DISCERNING THE UNDERSTANDING OF NOVICE PROGRAMMERS THROUGH EXAMINATION OF THEIR INTERACTIONS WITH CODE PUZZLES

KATRINA SARAH LYNNE JONES

Doctor of Philosophy

ASTON UNIVERSITY

August 2021

Aston University

Discerning the Understanding of Novice Programmers through Examination of their Interactions with Code Puzzles

Katrina Sarah Lynne Jones

Doctor of Philosophy, 2021

Thesis Summary

Programming could be viewed as a difficult discipline that some novice programmers (NPs) struggle to grasp, and the effect of this can be viewed in the inflated drop-out rates in Computer Science courses (Channel 4 News, 2017; HESA, 2020). The difficulty of programming can be partially attributed to the complexity of developing and applying effective computational thinking strategies, as programmers need to learn how to select, incorporate, evaluate and refine relevant programming constructs to create a program; implicitly requiring the knowledge of programming concepts, paradigms, execution models, and relevant domain-based knowledge. Due to this complexity, NPs can find it difficult to explain why they are struggling with programming to their tutors, making it difficult for tutors to identify the cause(s) of an NP's issues, leading to frustration and discouragement; consequently, the focus of this thesis is to explore a potential method for how to minimise this communication barrier.

This thesis documents the findings of three interpretivist, mixed-methods studies which asked 21 participants to rearrange modified 2D Parson's (referred to as Code Puzzles) into a working Java class while explaining their movements (think-aloud protocol) to an observer. Their times, movements, written feedback, and dialogue were recorded and analysed. For the secondary study involving 13 participants, the researcher discovered that 69% of participants believed the observer had correctly deduced their understanding of the programming concepts, and that 84% of participants believed the observer had correctly deduced their approach to creating a program, suggesting that using paper-based Code Puzzles may help tutors identify problem areas at a conceptual-based and approach-based level in a one-to-one setting.

Our approach led to the discovery of a novel area not incorporated into the original Parson's Problems design, known as the 'workspace', where participants grouped pieces together based on perceived similarity of the underlying programming concepts and/or context in which the pieces were used providing an insight into how NPs understand abstract programming concepts. This discovery led to the proposal of using the following in the context of investigating communication between programmers: think-aloud protocols, interpretivism over post-positivism research philosophies, cluster-based puzzles to analyse how NPs relate concepts and the proposal of a design of a diagnostic toolkit to analyse an NP's understanding.

Keywords: Code Puzzles, Parson's Problems, paper-based, Phenomenology, Programming, novice programmers, understanding, Java, Computer Science, Education, Cognitive Psychology

Personal Acknowledgements

This thesis is dedicated to the many persons who have aided in both my professional and personal development between the years of 2016 and 2021.

I would like to express sincere gratitude towards Aston University for providing me with this extraordinary opportunity to conduct research into an area that I feel passionately about, and for providing me with funding to support my endeavour.

I wish to thank my supervisor, BSc lecturer and friend Dr. Tony Beaumont, for his time, support and effort throughout my PhD journey. His vast expertise in the fields of CS and pedagogical research has aided me greatly during this piece of research. Not only has he selflessly contributed many hours of his time in giving me feedback during a viral apocalypse, his genuine, kind and caring demeanour has always supported me through my periods of ill health as well. Without his expert guidance, and continued support, it is likely that I would have been unable to complete my studies and for that I am, truly, indebted to him for this. I would also like to thank my associate supervisor, Dr. Alina Patelli, for her time – not only for myself, but my primary supervisor as well. I would also like to thank my ex- supervisor and MSc project supervisor, Dr. Errol Thompson whose passion for the world of pedagogical research inspired me to pursue my PhD. Despite unfortunately needing to leave my PhD journey in 2018, his kind, caring and genuine demeanour helped me through the early stages of my PhD.

With Tony's mind and spirit, and Errol's heart, I think we have created a piece of research that will hopefully be used in future work to help inspire and support novice programmers.

I also wish to thank my parents – Lynne and Peter – for their continued support of me throughout my studies; their love, patience, and support has been paramount to the completion of my research. I am forever indebted to them for such a wonderful opportunity! I would like to thank my brother Andrew, uncle Steve, and my late grandmother, Margaret for their support from afar. Similarly, I would like to thank my late and current feline companions for their continued affection and much needed distractions from my work.

I would like to thank a few of my research colleagues from the Aston STEM Education Centre – Dr. Sylvia Wong, Roger Howell and Andrew Kay – for all their feedback on my research throughout the years. As well as Professor Jo Lumsden and Dr. Aniko Ekart of the EAS department for their patience, kindness and support throughout the years.

Similarly, a huge thank you to all of my friends whose continued support, care, and understanding throughout the years has truly helped me through the toughest of times and has helped me remain (relatively) sane, namely: Sally F., Marcus C., Bethany H., Cameron E., Amanda H., Henry P., Tom M., John B., Rachel C., Kaylee S., Jess C., Jon E., Chris S., Jade M., Andrew D., Selina C., Jodie S., Megan B., Lizzie I., Jamie C., Matthew B., Priyanka J., Nisha J., Jordan D., Henrik H., Tien T., Asim M., Timea H., Tiffany W., Hayley C., and Alison D.. I hope you know how much I value you all!

And finally, thank you to all who participated in my research – without your feedback, there would be no thesis.

Collaborator Acknowledgements

This research was conducted by the research student and guided by the wisdom of her supervisor, Dr. Tony Beaumont, associate supervisor Dr. Alina Patelli, and former supervisor, Dr. Errol Thompson.

Related Dissemination of Work

This research was presented at a couple of conferences, listed below:

- Jones, K. S. L., Thompson, E. T. and Beaumont, A. J. (2018). *Can Learners' Interactions with Code Puzzle Pieces Accurately Match their Perceptions of the Related Programming Concepts?.* Higher Education Academy Conference. England: Birmingham. Programme Available at: <https://www.heacademy.ac.uk/system/files/downloads/Advance%20HE%20TLConf18%20abstracts_Day%203%2C%205%20July_STEM.pdf>

  The preliminary pilot study data was presented at this conference, with particular focus on approaching Code Puzzles from a differing perspective than in previous research.

- Jones, K. S. L., Thompson, E. T., and Beaumont, A. J. (2018). *Aston University EAS Postgraduate Research Conference*. Engineering and Applied Science. Aston University: Birmingham.

  This poster reported on the findings of the pilot study data and focused on the advantages and disadvantages of using Code Puzzles as a learning aide.

This research was also published as part of the Birmingham Digital Projects, available from their website as of 2020:

- Jones, K. S. L., Thompson, E. T., and Beaumont, A. J. (2018). *Can we use Code Puzzles to understand an NP's thought processes?* Birmingham Digital Project. Aston University: Birmingham. [ONLINE] Available from: <https://www.birminghamdigitalstudent.co.uk/single-post/2018/05/11/Learner-interactions-with-code-puzzles>

Ethical Approval Granted by Engineering and Applied Sciences Ethics Committee

This is to confirm that the EAS Ethics Committee approved and granted permission for Study #1115 to commence from April 2017-September 2020. If you have any concerns about the way that this research has been conducted, please contact the EAS Ethics Committee at the first possible instance.

List of Contents

List of Tables

List of Figures

List of Equations

Nomenclature

- **Workspace –** The novel, physical space that exists between the randomised puzzle pieces and the final constructed solution in 2D Parson's Problems. NPs grouped pieces together, providing insight into how related NPs feel pieces are to one another based on perceived similarity of concepts and/or context which the pieces were used in.

Abbreviations

- **CP1** – Code Puzzle 1 (2D Parson's Problems/One line of code per piece)
- **CP2** – Code Puzzle 2 (One segment/word/punctuation piece of code per piece)
- **CS** – Computer Science
- **EAS** – The School of Engineering and Applied Science at Aston University
- **EP** – Expert Programmer
- **IT** – Information Technology
- **M** – Mean
- **NP** – Novice Programmer(s)
- **SD** – Standard Deviation
- **SME** – Small or Medium-sized Enterprises
- **UK** – United Kingdom
- **QUI** – Quantifying Understanding Indicator

Chapter 1. Why is it important to understand Novice Programmers?

Novice Programmers (NPs) need to be understood and encouraged to succeed to remedy the growing shortage of programmers in industry and consequently reduce the digital skills gap. Computer Science (CS) and pedagogical researchers – such as the works of Benda, Bruckman, and Guzdial (2012), Griffiths (2014), Hu (2015) and Walters (2018) – have clarified that understanding, communication and effective teaching of NPs is crucial to bridge the digital skills gap and promote sustainable jobs.

Understanding the perspective and mind of an NP is challenging due to communication barriers that exist when a learner cannot explain effectively to their tutor what it is that they do not understand. This barrier likely contributes to the inflated drop-out and failure rates for CS courses. There are conflicting views on how to best to obtain, identify and represent an NP's level of understanding and whether there is a way to obtain this information in a way that can minimise the frustration caused by the process of learning how to program. This research aims to identify and evaluate whether it is possible to accurately discern an NP's understanding and, if so, how can such understanding be extracted.

This chapter provides an overview of the importance, background, and context of the research and clearly states the research: problem, question, aims, objectives and outcomes while clarifying the significance, scope and limitations of the research.

1.1 Why are programmers in demand?

As the significance and widespread usage of technology increases, so does the need for skilled programmers; businesses require programmers to utilise and effectively maintain an up-to-date virtual and social presence in order to stay relevant in modern times. Businesses often need to compete on a global scale, and the conception of the internet was the first stage of the dissolution of the importance of needing a physical and high-street based presence. According to Griffiths (2014), a third of UK SMEs had no website presence, and two thirds of UK SMEs did not sell or market their stock online despite contributing to 99.9% of the UK's private sector businesses. As a result, the SMEs' £1,606,000,000 turnover could have been boosted to £18,800,000,000 had they recruited programmers to digitise and streamline their front and back-end business processes (Griffiths, 2014). Programmers are therefore relevant, highly in demand and desirable to modern day employers.

However, companies specialising in recruitment – such as Robert Walters (2018) – reported that the UK has a Technology Skill Gap. Walters (2018) conducted a survey in collaboration with popular job

websites – such as Totaljobs and Jobsite – to question over 550 UK-based technology professionals on their experiences with finding and recruiting programmers. Walters (2018) discovered that: 70% of employers were anticipating a shortage of programmers in the near future; 24% of employers fear that a lack of programmers would greatly impact their business policies and financial gain; 11% of the employers did not believe the UK was prepared to "compete on a global scale in the technology industry" as a result of this skill gap; and approximately 55% of employers believed that the UK's political climate would cause an even greater technology skill gap – particularly in Yorkshire, London and the North of England where "programmer shortages were already great" (Walters, 2018). While Walters' (2018) research is self-published on their own website and cannot be found in an independently peer-reviewed source, other scholars have supported these claims with older sets of peer-reviewed data – for example, Golding (2008) expressed concern that recruiting programmers may not be an immediate problem in 2008, but that a technology skill gap was present and would become a future issue if not resolved. Sadly, this issue was not resolved as digital inequality and a digital skill gap are exacerbated according to April 2020 statistics; approximately 1,900,000 UK households were without internet, with tens of millions reliant on "pay-as-you-go services" to access benefits online (Pellini, 2020; Guardian 2020). These are people who could, with the right support and environment, become computer literate and even proficient programmers but are unlikely to attempt programming without having adequate experience with computers.

According to McDonald (2016), the combination of the UK Technology Skill Gap and UK Digital Skill Gap costed the UK economy £63,000,000,000 a year, making any effort to close the skill gap beneficial to UK taxpayers. But why do these costly skill gaps still exist in 2021?

1.2 Why is there a shortage of skilled programmers?

CS courses are attractive as programmers are in-demand, well-paid and can work on projects in a variety of different disciplines or choose to become aspiring entrepreneurs or inventors (Walters, 2018). Záhorec et al. (2020) highlighted the issues with the increased attraction to CS courses, noting that students are pressured to apply and excel in CS courses due to the perception that CS graduates will have plenty of opportunities for a stable, high-paying job to cater for the ever-increasing demands of the digital age. While Bennedsen and Caspersen (2007) found no notable difference between the emotional well-being of successful and unsuccessful CS students, Utting et al. (2013) suggests that the increased pressure to succeed may cause some CS students to misjudge where their skill level will be at the end of their first year of university. Záhorec et al. (2020) highlighted the issues with the increased attraction to CS courses, noting that students are pressured to apply and excel in CS courses due to the perception that CS graduates will have plenty of opportunities for a

stable, high-paying job to cater for the ever-increasing demands of the digital age; this perception is supported by the UK Government's report indicating that the "UKCES Sector Insights report predicts that by 2022 some 518,000 additional workers will be needed to fill the roles available for the three highest skilled occupational groups in the digital arena" (Shadbolt, 2016). While Bennedsen and Caspersen (2007) found no notable difference between the emotional well-being of successful and unsuccessful CS students, Utting et al. (2013) suggests that the increased pressure to succeed and obtain such jobs may cause some CS students to misjudge where their skill level will be at the end of their first year of university. Arguably, when a student's expectations mismatch their performance, this can lead to disappointment and discouragement which could negatively affect satisfaction metrics out of the control of the tutors. Yet, tutors may minimise the impact of this cognitive dissonance by addressing the realistic skill level that students will be at throughout the course and can assess whether their students are being realistic through satisfaction rates and surveys (Shee and Wang, 2008). Additionally, the marketing of such courses needs to ensure that inflated distortions of how well they will be able to program are not encouraged as this could further exacerbate the disappointment students feel when their expectations do not meet reality.

Despite the skill shortages, there are still record numbers of people applying to take CS courses (Záhorec et al., 2020), so there is no shortage in the quantity of candidates interested in pursuing CS. Yet retaining these candidates has been proven to be difficult; according to the HESA (2020) university-level CS courses had the worst drop-out rates in 2018 with 10.7% of undergraduates dropping out of their CS course before the end of their first year – this is 3% higher than the next course with the highest dropout rate, Advertising (7.7%) – and is therefore a significantly worrying statistic considering the UK Technology gap. Walters (2018) and Computer Weekly (2019) both cite growing concerns about the quantity of programmes dropping out of their courses. While NPs encounter similar issues like any other student from a differing field, the inflated dropout rate indicates a subject-specific issue alongside the usual reasons for dropping out (i.e., academic difficulties, personal, mental health, social, technical and/or financial issues as suggested by Luciana-Floriana et al., 2020), it is likely there is subject-specific issues present for CS specifically. CS can be considered a difficult discipline due to a "steep learning curve" (Ihantola and Karvita, 2011, and can be considered not an easy discipline to excel in as some graduating students are not up to employers' standards (Walters, 2018).

While Vahldick (2015), Hnin (2017) and Bosse and Gerosa (2017) have conducted studies on how retention rates could be improved in CS courses, collective causes of the high drop-out rates have not yet been established; this is because a collective study across CS courses to establish a universal cause for inflated dropout rates is challenging as CS courses vary in terms of programming languages,

entrance requirements, course advertising (i.e., students' cultures, backgrounds, interests, pre-conceptions about programming), programming paradigms, teaching styles, content and purpose making comparisons difficult.

1.2.1 The difficulty of programming

Programming can be difficult for NPs to grasp, and that NPs need to be understood to help reduce excessive the CS course failure and/or dropout rates courses (Lahtinen et al., 2005; Parson and Haden, 2006; Correia et al., 2015; Santos and Gorgônio, 2015; Vahldick, 2015; Hnin, 2017; Záhorec et al., 2020)– yet, to understand why programming is considered difficult we must first explore the definition. Guzdial et al. (2005) define programming as the process that generates "an executable segment of code that performs a required task" where the process of programming is language-dependent, and the generated flow of logic of the solution is language-independent. However, programming languages often were created with a specific programming paradigm in mind – for example, Java is traditionally object-orientated (Oracle, 2020) – leading to Thompson (2008) suggesting that programming paradigms influence the way a solution is constructed, and that the flow of logic is therefore language-dependent. Conversely, Pitta-Pantazi et al. (2007) note that mathematical equations are language-independent, as equations can be equated to the flow of logic of an algorithm. Therefore, the holistic process of programming is important when cross-examining studies.

Popularity of any given programming language may change over time, making direct course comparisons an issue if a language has fallen out of favour in syllabuses as it may not be as widely taught as it once was. The language itself may evolve and cause parts of the language to become depreciated in favour of new parts of the language. For example, in Java, the java.util.Date has now been depreciated and almost replaced by java.util.Calendar – which operates in a different way to Date due to Calendar being an abstract class rather than a concrete class which offers a different level of difficulty for an NP. Therefore, to achieve the same functionality in older versions of the language may be more difficult than present day if parts of the library relevant to the problem did not exist at that point in time, making course comparisons difficult if the years between the courses are drastically different. However, if researchers chose a more general aspect of the CS course – such as investigating which programming paradigm is optimal for NPs like Bosse and Gerosa (2017), then the research may not become so quickly outdated. Nevertheless, the selection of intended programming paradigm and programming language(s) for CS courses are crucial to an NP's experience of programming and perception of programming in general. For example, error messages, syntax, language documentation, and the development environment are key to influencing the NPs'

experiences. Hu (2015) noted that "poor[ly] designed computer languages" and IDEs cause additional barriers for NPs, and that language and IDE selection is therefore crucial to instil confidence in NPs and avoid cognitive overload. Many programming concepts are abstract, yet the selected programming language is the tangible learning element for an NP and their tutor. In a novice's eyes, writing a program is an authentic task that provides evidence of their capability as a programmer, and as a tutor, writing a program is evidence that the novice understands programming concepts and can apply them in relevant contexts. Consequently, if NPs are struggling to write code and debug effectively, they can view this as a reflection of their own lack of capabilities and potential to be a programmer and become frustrated and disheartened (Bosse and Gerosa, 2017; Giannakos et al., 2017). Lahtinen et al. (2005) surveyed 559 undergraduate CS students and 34 teachers across 6 universities in 5 different countries; as part of the survey, they were asked to rate the perceived difficulty of various aspects of their computing courses on a Likert scale, where 1 was 'very easy' and 5 was 'very difficult'. They discovered that the average student perception (2.8) was lower than that of their teachers (3.5) across all universities. On average, the students and teachers rated the same three programming concepts as the most difficult to learn, and they are (in the order of highest rated difficult concept to lowest rated difficult concept out of the top three). These difficult programming concepts imply that computer architecture and the way execution of a program works are trouble areas for NPs, which is interesting, as these are, arguably, tangible elements that work in the background of a high-level program execution but are integral to the way a program works rather than the abstract programming concepts proposed to be difficult by Bosse and Gerosa (2017). This discrepancy likely highlights that the 559 CS courses may have spent more time, on average, teaching programming concepts/language/paradigms than teaching about the integral process of how a program runs. These findings also indicate that topics that require a mixture of programming concepts, knowledge of computer architecture and practical implementation elements were deemed the most difficult; supporting the idea that programming is a multi-faceted, multi-dimensional and layered technical skill which contributes to the difficulty of programming. Sleeman (1986), Soloway and Spohrer (1989), du Boulay (1989), Kölling (1998), and Khazaei and Jackson (2002) support these findings by arguing that NPs struggle to understand what a program is, and how the compiler works. In addition, Lahtinen et al. (2005) discovered that students and teachers considered language-specific libraries and abstract data types to be very challenging thus supporting the claims of Thompson (2008) and Hu (2015). Finally, Lahtinen et al. (2005) found that students and teachers, on average, rated the same three programming concepts as the easiest to learn, and they are (in order of lowest rated easy concept to highest rated easy concept out of the top three): selection, repetition and

variable. – which suggests that the logic flow of a program is the easiest element to understand for NPs.

The programming language alone is not enough – there are now multi-paradigm languages (i.e., Python) that make direct course comparisons difficult without providing context in the way the language has been presented and taught, as two CS courses that provide the same language might not use the same paradigm. Research is inconclusive about the 'best' programming paradigm for NPs to learn first (Ehlert and Schlute, 2009; Bosse and Gerosa, 2017), and therefore there is no universally agreed upon paradigm or language that is optimal for NPs. Consequently, Kauffmann (2011) and Giannakos et al. (2017) acknowledged that the way programming is taught is likely "not optimal" and various pedagogical frameworks – such as Variation Theory proposed by Marton and Tsui (2004) – have been proposed to try and improve the quality of CS courses.

Even if the hurdles of programming paradigms, concepts and languages are overcome, the process of programming itself is challenging. NPs need to develop their knowledge alongside their critical thinking capabilities which can cause cognitive dissonance if they are accustomed to memorising facts as a form of learning – such as by rote learning – as knowledge recall alone is not sufficient to become a proficient programmer. Therefore, NPs may have to think in a different way to what they have been taught in other subjects (Bork, 1972; Desmedt and Valke, 2004). Bosse and Gerosa (2017) and Giannakos et al. (2017) supports that NPs struggle with the process of programming rather than the language, as they found that NPs can get frustrated with the process of debugging and with their inability to write "effective solutions". According to Lister et al. (2006), programming can be viewed as a "hierarchy of knowledge and skill", inferring that NPs likely require a 'good' understanding of the basic building blocks of how to build a program, before they can fully understand higher-level programming concepts such as software design patterns, which can contribute to the complexity due to the multi-level nature of the knowledge needed.

NPs need to develop and evolve appropriate strategies to approach programming (Thompson, 2008; Zarb and Hughes, 2015; Bosse and Gerosa, 2017). Beaubouef and Mason (2005) postulated that a lack of mathematical and problem-solving skills was a major difficulty for NPs. Spohrer (1989) created a document amassing older research which dealt with the preliminary issues presented by NPs, and, in this document, it is suggested that NPs typically have a "surface level" of knowledge regarding programming which is revealed when they fail to apply the concepts to a practical solution. Lahtinen et al. (2005) observed "the knowledge of novices tends to be context specific, and they also often fail to apply the knowledge they have obtained adequately".

Consequently, there is an emphasis on NPs developing effective problem-solving strategies and programming intrinsically requires critical thinking as a skill and, as such, a plethora of general approaches to software development and problem solving are taught in CS courses (Aston University, 2020). Three different approaches have been selected to serve the purpose of illustration the variations that exist within the field of CS and are commonly taught to NPs (Aston University, 2020); Polya's (1945), Vicker's (2009) and Stepwise/Top-Down process (2020).

There are obvious similarities between Polya's (1945) and Vicker's (2009) approaches, and Vicker (2009) does acknowledge influence from the traditional general-purpose problem-solving process and confirms its relevance for programmers. Vicker's (2009) evaluation stage is more formally documented, in the hopes that NPs will evaluate and reflect upon their own code and process to correct it for future coding exercises. Stepwise process is quite different and can be equated to a technique that a programmer may use to divide and conquer their code. Consequently, the Stepwise approach is designed to alleviate the intrinsic cognitive load by allowing programmers to focus on individual steps rather than experiencing cognitive overload and becoming unsure where to start constructing their solution and therefore acts as a framework to help shape an NP's computational thinking pattern. The generalness of these approaches garners criticism in the context of programming, as NPs are unlikely to be self-aware enough to devise a plan that is unchangeable – and while reflection does take place at the end of the process – it is likely an iterative process and 'trial-and-error' aspect is incorporated naturally when they first start programming. Similarly, Vicker's (2009) approach is very document-heavy; NPs may become overwhelmed by the sheer volume of reflective work they need to proactively participate in and become unable to concentrate on what they can surmise from their work. In contrast, the Stepwise approach is simplistic and does not indicate how they should divide the problem which can lead to confusion or creating too many sub-problems from a major problem resulting in the over-complication of a task and making redundant steps. In essence, NPs may resonate with one problem solving process over another based on their mental model and computational thinking pattern style at first, and then learn how to apply and use the appropriate process to the problem when needed once enough practice and experience has taken place (Kauffmann, 2011).

1.2.2 Student misconceptions about CS courses and programming

Not all CS failure or dropout rates are attributed solely to the CS tutor and/or teaching style – the way a course is advertised, and consequent preconceptions of a subject and course by the students, contribute to the dropout and failure rates. This is particularly true for CS, where Beaubouef and Mason (2005) discovered that some students believed IT was synonymous to CS, and where students

had unrealistic expectations about their ability to program by the end of their first year, contributing to cognitive dissonance.

Hu and Kuh (2000) suggested that the perceived relevance of a topic to the CS students' desired occupation is important to help retain CS students. Similarly, Hu and Kuh (2002) and Beaubouef and Mason (2005) argued that CS students joined courses with specific interests in mind – if the CS course spent little-to-no time on these interests, CS students are likely to lose interest and motivation to continue with the course. Inspiration and motivation are key – with a CS student's personal investment in a course and the importance of the topics being paramount to happy CS students (Shee and Wang, 2008; Alshammari, 2016). Motivation is important, as Bransford et al. (2020) contend that practice is essential to learn, it could be, therefore, that effective NPs automatically perform 'deliberate practice' – in other words, in other words, practice which focuses on areas that a learner is struggling with is required with adequate feedback – as opposed to ineffective practicing which includes non-focused regurgitation of knowledge, or repetitive focus on an area which does not need enhancing. Shaffer and Resnick (1999) also argued that the use of 'authentic tasks' – tasks that are realistic to the domain – are required to further improve the quality of the practice. However, an issue with using authentic tasks with NPs is that it runs the risk of causing cognitive overload if you were to, say, give them a requirements documentation and ask them to design and implement a program to meet the needs of a client. The idea of authentic tasks seems to conflict with the suggestions of Ihantola and Karvita (2011) who propose that the task should be set at the level of the programmer – for example, giving an NP who can barely trace or explain in plain English what a program does would likely struggle with a realistic task given to a professional programmer. Similarly, Kauffmann (2011) contends that NPs "bypass self-control" – in other words, NPs often went and created their own solutions rather than comparing and trying to match their solution to the sample one – and thus make their lives more difficult for themselves and the tutor. Perhaps, therefore, there should be certain aspects of a task that remain authentic, and tutors should ensure the CS students remain focused on the task, but the level of that task would likely be ineffective if the task is hyper-realistic as it would overwhelm an NP and they may not focus on the critical aspect of the object of learning. However, a balance needs to be struck – if an NP is too rigidly confined then inspiration and motivation may wane. Keeping students inspired and motivated is difficult; as noted by Kauffmann (2011), CS students struggled to get "the bigger picture" due to the modular-style approach commonly adopted to CS courses, which was found to contribute to CS students' struggles as they failed to relate programming concepts to one another, causing issues for CS staff in identifying the source of their poor understanding. Lister (2004) and de Raadt (2008) noticed that if students have a "fragile knowledge", where they cannot apply relevant knowledge to

achieve an unforeseen task, that this correlates to higher failure rates in practical undergraduate courses in comparison to theoretical undergraduate courses – this is applicable to NPs in CS courses, if they cannot identify relevant programming knowledge and apply it, they are likely to fail as CS is a practical course.

Cultural differences and expectations play a significant role in whether the CS student feels the quality of teaching is up to standard – a CS student who expects more practical-based application that is confronted with a traditional authoritarian lecture series is likely to feel dissatisfied and affects their personal behaviour in willingness to defer to authority if they need assistance as certain cultures may prefer to shy away from confronting figures of authority to ask queries. In essence, a disappointed, disheartened CS student is far more likely to drop out and/or fail than a content CS student – therefore, both CS tutors and CS course advertisements need to be carefully crafted to mitigate these risks for CS courses. Similarly, the previous subject(s) studied by the CS student are relevant – Bork (1972) postulated that NPs from science backgrounds may be more inclined to have fewer issues adapting to programming than those from the arts, as, Bork (1972) argued that the way in which science approaches subjects is like the "mindset required for programming".

1.2.3 Communication barriers preventing effective dialogue between CS students and tutors

Good communication skills are highly desirable transferable skills wanted by CS employers (Begel and Nagappan, 2008; Walters, 2018); therefore, it is important to investigate why poor communication skills might manifest so that NPs can achieve their potential successfully and harness a skill that is very valuable in the current climate.

1.2.3.1 Psychology of Communication Barriers and why they occur

From a psychological perspective, learners of any discipline struggle to express what they do not comprehend – this thesis dubs this issue a 'communication barrier' that is formed between tutor and student and prevents effective communication of programming issues due to a misunderstanding or mounting frustration that occurs during miscommunications.

Sweller et al. (1988) proposed Cognitive Load Theory (CLT) to "provide guidelines intended to assist in the presentation of information in a manner that encourages learner activities that optimise intellectual performance" and, partly, to explain the phenomenon of a learner feeling overwhelmed. In CLT, it is suggested that humans have a finite capacity to obtain and store information, and that too much information at any one time can cause information overload which is when a learner feels so overwhelmed by the information that they cannot process it effectively (Sweller et al, 1998).

Information overload is known as 'cognitive overload' and Sweller at al. (1998) advise against intentionally causing this state of mind as it can cause the learner undue stress and discomfort and may cause an inability to learn or engage with the content presented to them. Simplistically, cognitive load can be referred to as 'light' or 'heavy' – each linking to how strenuous the activity is on cognition.

Regarding programming, NPs have a heavy cognitive load as they have to focus on several different aspects of programming simultaneously (e.g., problem analysis, computational thinking, selecting relevant programming concept knowledge, thinking about the design of their solution, testing their solution, debugging) and can therefore easily become overwhelmed if the content is not tailored to their capabilities effectively. Cognitive overload is linked to poor communication or ability to cite the cause of the discomfort (Sweller et al., 1998) so NPs who are overwhelmed may say, for example, that they "don't understand Java" when it is only one aspect of Java that they might not comprehend – but they are so overwhelmed, that they don't know what to focus on to identify the issue. Humans are hard-wired to focus on negative aspects more so than positive aspects, and therefore it is no surprise that an overwhelmed learner is liable to focus more on what they cannot do as opposed to what they can do.

1.2.3.2 Reasons for CS Communication Barriers

CS is a terminology-heavy discipline that requires a motivated, inspired learner that is knowledgeable in abstract programming concepts, versatile in critical thinking, and willing to practice programming – involving both comprehension and literacy of programming language. CS companies, such as Microsoft in 2008, have rated "good communication skills" as a highly desirable transferable skill that they look for in programmers (Begel and Nagappan, 2008). However, Freudenberg (2007) noted that "the cognitive aspects of [communicative programming techniques such as pair programming] are seldom investigated and little understood", possibly because studies into programmers communicating with one another often use quantitative metrics – such as using the number of communication transactions per process and comparing it to the quality of an end goal (Zarb and Hughes, 2015), rather than investigating the form of interaction exhibited or the quality of communication. The quantitative metrics used for communication are not surprising, considering that measuring the quality of communication using qualitative metrics is often subjective by nature and that the form of communication can differ from all kinds of extraneous variables – such as the participants' temperament, personality, confidence and/or culture.

Communication barriers occur when there is a misunderstanding in words or phrasing, or when a learner does not understand the topic enough to know what they don't understand and are struggling with – this can result in the learner stating they don't understand the whole concept when it is just part of the concept they do not comprehend. Research has shown that NPs are not an exception here – Ragonis and Ben-Ari (2005) found that programming students answered generally – so, they could not specify what kind of encapsulation, modularity and data structures they were referring to in their answers. Similarly, du Boulay (1989) argued that NPs did not know what a program was, and in essence, did not know what a program was not implying that some NPs may not even know what they are meant to produce. Likewise, Garner et al. (2005); Benda et al. (2012) and Hu (2015) discovered that NPs sometimes did not know how to approach programming effectively and/or where to start with the process of programming – which these types of NPs would not necessarily know that their process isn't as effective as it should be or have the expert knowledge to be able to remedy this issue without assistance but explaining this issue would be difficult if NPs do not know where to start. Thompson (2008) suggested that NPs may struggle to identify the relevant domain and past knowledge required in order to accomplish the task; NPs would not be able to communicate without expert assistance as these types of NPs do not know what is and isn't relevant. Thompson (2008) and Kauffmann (2011) suggested that NPs struggle to relate CS concepts together, as NPs fail to see the bigger picture on how the logic flow can be transferred to another situation, suggesting NPs may 'know' a concept in one instance, but not in another, which can confuse NPs and their tutors. Bosse and Gerosa (2017) and Bosch and D'Mello (2017) highlighted that some NPs struggle with debugging which indicates that some NPs struggled with understanding the source of an error, comprehending the error message, and/or the internal computer's logic related to the reason for the error appearing but would likely not be able to express this to their tutors. Bosse and Gerosa (2017) elaborated on the importance of NPs not transferring programming paradigms too early, as this is likely to cause confusion, and NPs would not know why cognitive dissonance was present and may attribute such discomfort to not liking the language as they do not comprehend the abstract concept of programming paradigms. du Boulay (1989) postulated that NPs can misunderstand or have a shallow depth of understanding about the purpose of syntax and indentation, which can cause issues for NPs if their assumptions are challenged by evidence of the syntax or indentation being used differently to their expectations.

Due to the complexity and practical implications of programming, CS studies that investigate communication are typically linked to the study of programming or teaching techniques – such as pair programming – and therefore further analysis will investigate the frequent research areas that

are linked to the communication barrier than can occur between programmer and tutor, or programmer and programmer.

1.2.3.3 Frameworks for Analysing Communication

CS researchers in the area of analysing communication of NPs, particularly in the contexts of pair programming, argue that using a modified form of Straussian Grounded Theory is recommended (Zarb and Hughes, 2015) but what is 'Grounded Theory'? Grounded Theory is a sociology-based 'middle-range theory' – a framework that involves both theoretical and empirical research – that's primary purpose is to produce a theory driven by evidence 'grounded' in reality and can be supported successfully in similar research settings. According to Oktay (2012), Grounded Theory's "focus on the development of middle-range theory is the primary way that [it] differs from other qualitative methods" and, from the perspective of this research focus, is designed to help with" the 'meanings' individuals ascribe to aspects of their cultures or their lives (phenomenology)". While there are different versions of Grounded Theory, there are four key components that drive the formation of a sound theorem: theoretical sensitivity (the ability for the researcher to remove their biases from the situation and correctly identify the key aspects of the phenomenon that they are observing), constant comparative method (the ability for the researcher to treat participants as individuals and compare each instance with one another to develop concepts), theoretical sampling (where a sample is relevant to explore the selected phenomenon) and theoretical saturation (where no new categories are observed when more data is added). If all these criteria are met, then a theory (or set of theories) should be generated relating to the selected phenomenon.

Grounded Theory has been used successfully in the past; Bryant (2004) created a framework, primarily for the use of studying the dialogue between NPs in pair programming originating from Straussian Grounded Theory that consisted of the following steps to analyse the transcripts generated by the pair programming sessions: 1) Creating open codes; 2) Using the open codes on the transcripts; 3) identify patterns in the coding; and 4) interpret the coding. The key aspect here is that the researcher is immersed in the dialogue of the NPs and obtains the overall sentiment and feeling between participants' and the types of dialogue they use. Zarb et al. (2012) found that NPs frequently experienced several types of dialogue when communicating to other NPs: review – examining the code to determine if it is sufficiently achieving the intended task; suggestion – offering advice on the next step; explanation – explaining an aspect of the code or part of their previous dialogue, code discussion – general discussion about the code; muttering – where the dialogue is inaudible or nonsensical noises; unfocused – where the NP gets distracted from the task; and silence – where the NP is not speaking.

1.2.3.4 Pair programming as a technique to reduce communication barriers

Researchers such as Williams et al. (2000), Begel and Nagappan (2008), and Zarb and Hughes (2015) have conducted studies and discovered that pair programming does reduce communication barriers.

Pair programming is defined as "a software development technique where two programmers work together side-by-side on the same machine to achieve their goals" and has garnered popularity in recent years due to the potential benefits of the technique (Zarb and Hughes, 2015).

Pair programming has a vast array of benefits for both NPs and EPs, and has been reported to contribute to: greater enjoyment levels (Bryant et al., 2006) and time seeming to pass more quickly (Sanders, 2002), increased "knowledge distribution" (Zarb and Hughes, 2015), improved problem-solving capabilities (Williams et al., 2000), improved confidence (Williams and Kessler, 2000b), improved satisfaction of produced code (Kavitha and Ahmed, 2015), improved team effectiveness (McDowell et al., 2003), improved time management (DeClue, 2003), higher pass rates (Porter et al., 2013), improved comprehension of "unfamiliar topics" (Zarb and Hughes, 2015), improved bug detection (Hulkko and Abrahamsson, 2005), improved refactoring (Chaparro et al., 2005), self-sufficiency (Zarb and Hughes, 2015), and better quality code (Cockburn and Williams, 2001) when compared to traditional avenues. Hanks (2006) demonstrated that pair programmers still face the same issues as solo programmers, but that these issues were more likely to be overcome than when alone.

Yet, pair programming requires active, cooperative communication to succeed (Thomas et al., 2003; Begel and Nagappan, 2008; Zarb and Hughes, 2015), with Zarb and Hughes (2015) postulating that pair programming without the communication is merely "reviewing each other's code". In ineffective pairs, programmers were found to feel discomfort (Cockburn et al., 2001), frustrated, guilty, and feeling like "they had wasted their time" and that EPs benefitted significantly less than NPs (Thomas et al., 2003) which led to poor productivity and performance in Aiken's (2004) study. Likewise, Melnik and Maurer (2002) argued that trust is paramount to a successful pairing and that distrust among pairs caused significant issues.

The quality of communication is, therefore, an integral part to pair programming – but what differentiates a successful and a non-successful pair and can this be predicted? The answer lies in compatibility of communication and temperament. Williams and Kessler (2002) diagnosed that a long silence was a sign that a pair was not communicating effectively, although the exact length of time of the silence is debatable. Aiken (2004) agreed and suggested that "no more than a minute should pass without verbal communication". Bryan et al. (2006) did suggest that a ratio is more applicable –

suggesting that the driver, the programmer who is coding, and the navigator, the programmer who is reading, need to speak roughly 60:40, respectively. The exact length of a silence not being established as a universal metric is not surprising, considering that the cognitive load of the task and capability of the student needs to be considering when trying to calculate how long is too long a silence. Different programming tasks have been shown to require different amounts of cognitive load, which can naturally cause NPs to become lost in thought. Likewise, measuring the time or ratio of silence to speech does not consider the difficulty of the task, or the nature temperament or disposition of the programmer – some people are more naturally inclined to be introverted than extroverted, or may be hyper focused on the task and forget to communicate their thought processes to their partners. The form of communication is also considered to be important – Flor and Hutchins (1991) observed that there needs to be an exchange of ideas coupled with regular feedback in the form of debating when two programmers collaborate on software maintenance. This thesis argues that there are some missing components to this analysis – and that the quality of feedback and reasoning behind decisions must also be important factors that need to be acknowledged – if a pair are ignoring each other or refusing to question the others' reasoning behind design decisions, the discourse is likely to become stilted, and the learning process is likely suboptimal. Therefore, a mutual sense of respect and open channels of communication must be an integral part of successful pair programming even if these aspects have not been considered as essential – likely due to the difficulty in measuring what is and isn't 'good quality feedback' and 'good reasoning behind design decisions'.

Begel and Nagappan (2008) considered the required temperament of both programmers and suggested that NPs in pairs needed to exhibit certain qualities: good listener, good debater, articulate, logical, verbal, friendly, non-defensive, clear, inquisitive, and honesty. Unfortunately, Zarb and Hughes (2015) observe that scholars fail to "investigate how communication happens within pairs and how it is or is not effective" and this is further supported by the work of Sharp and Robinson (2010) and Stapel (2010). This raises the potential research question of how do programmers communicate, however, this question itself is too big a scope for a single researcher to comprehensively answer, and due to limited sample size and time available to the researcher, this research question was not deemed feasible – that said, it is acknowledged that the work on NPs' types of communication and the corresponding effectiveness of the communication is an under researched area and a systematic search revealed that there is no universal measurement for how effectively NPs communicate nor what causes communication to occurs. As suggested by Zarb and Hughes (2015) "a better understanding of communication within pair programming could lead to

improved teaching practices" which is why the study of NP communication is an important area to encourage research in.

Therefore, this thesis proposes that communication barriers in CS contribute to the frustration of learning how to program and need to be reduced where possible.

1.2.4 The way programming is taught and assessed is not optimal for NPs

While the way programming is taught and assessed is not a primary focus of the thesis, it is important to acknowledge that the way in which CS courses are taught and/or assessed may not be optimal and that the research needs to consider the level of difficulty and presentation/style of the tasks presented to NPs as this impacts the research design.

Janpla and Piriyasurawong (2018) argue that NPs failing to evolve their computational reasoning is a symptom of traditional lecture-based learning and that switching to problem-based learning would help to mitigate the issues. However, Kauffman (2011) observed that, when teaching theoretical programming concepts such as databases and database structures, students struggle with a notable "lack of knowledge transfer" if problem-based learning was used. However, Century et al. (2020) suggest that there needs to be a mixture of both lecturing material and tutorials as both the theory and application of the theory is required for programming an intended learning, although they did also support the notion that problem-based learning techniques were more effective than traditional modularised learning for CS courses.

Although the way the content is taught is important, the structure of the content itself needs to be considered. Ihantola and Karavirta (2011) argue that there are different difficulties of task: tracing – where a programmer needs to identify the source of a bug, which, by them, is considered a trivial task. Yet, Lahtinen et al. (2005) and Bosch and D'Mello (2017) disagree that this type of task is trivial as in the study conducted by Lahtinen et al. (2005) it was reported that errors were considered one of the most difficult to understand and resolve. Denny et al. (2008) analysed Java submissions made by first year undergraduates from the University of Auckland with the aim to catalogue the type and frequency of syntactical errors in submissions and discovered that syntax was a significant barrier for completing the submissions; even in the easiest version of the Java task, students made several incorrect attempts to compile and run the code before managing to successfully complete the task. Research has reported that the most common difficulties in terms of practicalities are: an inability to find the source of errors; an inability to create an effective solution for a given task; and modularise the code using elements such as methods, functions and procedures. Denny et al. (2008) also noted

that the programming concepts that caused the most difficulty were: functions, procedures, error handling and arrays which directly goes against the findings of Ihantola and Karavirta (2011).

Conversely, Ihantola and Karavirta (2011) postulate that harder tasks are explaining what a program does in plain English – considered 'intermediate' – and writing a program is considered the hardest task as they argue it requires both understanding of the language and how the compiler works.

Assessment of students' programming skills are typically based on criteria that requires the student to be able to create an executable segment of code, that fulfils the required task, with the appropriate quality of code (e.g., assessments used by Aston University, 2020). Therefore, the definition for a 'good level' of programming should be the creation of an executable segment of code, that fulfils the required task, with the appropriate quality of code – the highest level of difficulty of task, according to Ihantola and Karavirta (2011). That said, how long this takes an NP to achieve such a task is debatable and the complexity of the task itself needs to be considered, alongside the authenticity.

1.3 Research Motivation, Problem, Scope and Limitations

Programming can be difficult for NPs to grasp, and that NPs need to be understood to help reduce excessive the CS course failure and/or dropout rates. While there is a multitude of different ways that could be investigated – such as research into the reasoning behind CS course failures, the pedagogical designs of CS courses, and the expectations of NPs – reducing the communication barrier was selected. Studies into communication between programmers of any level is limited, which was acknowledge by Zarb and Hughes in 2015, and thus any contributing knowledge to this field is of importance as the communication barrier has been established to cause frustration for NPs as evidence in pair programming activities. Zarb and Hughes (2015) suggested that research had failed to illuminate how to facilitate effective communication between NPs, and that little research had been conducted into the analysis of NP communication patterns and their meanings. Consequently, this research focused on furthering the understanding of NP communication and chose to replace peer-interaction with the NP explaining their reasoning to a rubber duck (the observer) while using a puzzle-based medium as a talking point to study whether this is an effective form of communicating understanding. Yet, qualitative studies – which are required to address this research gap – are subjective in nature, and therefore the research uses a mixed methods approach to try and balance the limitations of quantitative and qualitative research while also offering a comparison point.

However, it would be an unrealistic goal for the research to presume it can eliminate dropout and/or failure rates. Instead, this research hopes to contribute knowledge to the fields of CS, CS psychology and pedagogy that can bridge the gap between tutors and NPs and allow for effective communication in the hopes that this can lead to making programming a less frustrating discipline to learn which may, in turn, have an impact on excessive failure and/or dropout rates in CS courses. In the scope of one PhD, it would be impossible to explore all possible paths associated to reducing drop-out rates in great depth for all levels of programmers. Therefore, the research chose to focus specifically on first-year undergraduate CS students due to the accessibility of this type of participant to the researcher and acknowledges that not all first year CS students have the same level of programming expertise – therefore there is an inherent assumption that first year CS students are NPs. That said, it can be assumed that programmers starting an undergraduate course in CS want to learn programming and are the target audience for trying to reduce the drop-out rates as the statistics provided by Shadbolt (2016) relate to first year undergraduate CS students.

There has already been a multitude of studies investigating: why programming is difficult; the issues surrounding the difficulty; and how practitioners can address the difficulty in their teaching practices that dates back to the 1980s to the 2020s (e.g., Sleeman, 1986; Soloway and Spohrer, 1989; Eckerdal and Berglund, 2005; Thompson, 2008; Utting et al., 2013; Alshammari, 2016; Záhorec et al.; 2020) and that only a sub-selection of relevant research has been documented thus far. With many of the researchers, like Garner (2005), emphasising the importance of understanding the difficulties of NPs to develop more effective teaching practices. This volume of research spanning decades indicates that a universal approach and/or universal starting language/paradigm has not yet been established or agreed upon, and that there are known issues with the difficulty of CS courses that are not easily identified and/or resolved. Programming is rapidly evolving area, and it is unlikely that a one-size-fits-all approach will remedy all issues with CS courses. That said, this research aims to focus on investigating the difficulties of NPs, rather than the noted programming difficulties explained by tutors and experts.

This thesis will refer to research publications, business journals and practitioners' pieces where relevant; as practitioners have an enriched experience of teaching in an authentic environment and know what is feasibly achieved in a classroom; while businesses have an insight into the quality of programmer they require; whereas researchers have a typically impersonal insight and may often have access to larger, peer reviewed datasets. Therefore, all entities have a valuable insight into the discussion of understanding NPs, why programming is difficult and what quality standard a programmer needs to achieve at the end of their course.

This research has obtained ethics approval under the Aston University regulations, and we will discuss briefly how the University ethics regulations impact on this research. Researchers cannot create alternative methods of teaching and divide up students to receive different approaches, and qualities of teaching – a student in one group where the teaching method doesn't work would be disadvantaged when compared to students in a group with a more suited teaching approach. CS students who participate having an advantage and/or disadvantage over their colleagues is forbidden due to ethical and legal implications; this meant that the research design could not use a control group as a core part of the research and adopted a more interpretative philosophy to research as it is difficult to generalise or normalise the differences between student perceptions of programming.

Therefore, this research had to rely solely on undergraduate volunteers from CS disciplines. Additionally, lecturers cannot be made aware of the participants that have participated in any research studies as this could influence the lecturer to either give them positive or negative attention in comparison to their peers. Therefore, the datapoints presented in this thesis are anonymised with the video and audio recordings available on request through Aston University protocols. This meant that in comparison to other studies in the related areas, the sample size is relatively small. Moreover, aside from the issues mentioned being out of the scope of the thesis, the recruitment process did mean acquiring a very large sample size was difficult to perform.

The justification for using undergraduate students, who were at the end of their first year CS course, is that this is a form of control – this meant all of the participants were exposed to the same modules and content for one academic year; this was planned as part of the research methodology as the researcher could not feasibly control participants' backgrounds, or access their student records, as that could compromise their identity protection which would be against ethics guidelines. This is possibly why many of the published research studies that observed participant interactions with code puzzles were taken from naturally occurring data as part of university courses (e.g., Helminen et al., 2012) but this was not possible under the University regulations.

As it was not possible to do a longitudinal study with the cohort or allow them to learn a second language in order to control the environment more, Java, which is an object-orientated language, was chosen in order to help form part of the nature control that appears in students at the end of their first year of undergraduate study (as they have all, at least, participated in learning Java for approximately 15-20 weeks). The datapoints achieved from the three experiments still are significant and the level of control did mean that the Code Puzzles could be set at roughly the correct level based on my ability to access the content that they had learned from.

1.4 Research Purpose and Question

The purpose of this research is to explore a potential avenue for accurately diagnosing the understanding of NPs in order to reduce the impact of a communication barrier (Zarb and Hughes, 2015). While it would be unlikely that one tool would completely solve the issue of inflated drop-out rates in CS courses, it would be a good first step if practitioners had a diagnostic tool to help identify their students' issues.

Therefore, this thesis proposes to use Code Puzzles as an understanding diagnostic tool to help reduce the difficulty of programming for NPs.

The researcher chose an exploratory research question rather than a confirmatory research question as, in order to understand understanding, we need to delve into the areas of ontology – capturing participants' versions of realities – and epistemology – understanding a particular phenomenon which is understanding their perspectives of programming using Code Puzzles as a medium (see Table 1).

Research Question
Can we accurately discern an NP's level of understanding through the examination of their interactions with Code Puzzles?

Table 1: Research Question

1.5 Research Aims

The aim of this thesis is primarily A-1, with several secondary aims (A-2 to A-8) investigated alongside the primary aim (see Table 2). It should be noted that 'to understand' is not a measurement by itself – for example, someone could say that they understand a topic when they do not. As a result, for all of the aims (see Table 2), the term of 'level of understanding' will be measured by whether the understanding of the participant reflects that of the expert analysing their data, and, whether the participant agrees that the representation of their understanding as described by the expert matches their own perceptions of their understanding or whether this is untrue. Level of understanding is an ambiguous term, but based on the discussion of why programming is difficult we can conclude that there is potentially a hierarchy of understanding – where certain programming concepts are required to be known prior to being able to understand others – and that there needs to be exploration into what these concepts are that NPs need to encounter in our chosen language – Java – and at what point they 'understand' a concept fully without any lack of understanding or misunderstanding, or whether this could even be achieved. After all, it is unlikely that NPs have a universally applicable clean point where everything suddenly clicks in their minds. Marton and Tsui (2004) spoke of threshold concepts when discussing CS – and how there are certain critical aspects of an object of learning that if an NP learns can deepen their level of understanding to a degree that the NP cannot default back to the previous mode of thought after learning about the threshold concept. Even so, it is unclear whether threshold concepts are down to the individual's perspective of programming formed by how they relate programming to their own constructed worlds or whether there are certain aspects of programming that can substantially shift the viewpoint of all NPs at a certain stage

of their learning. The main issue with measuring understanding is that, according to phenomenology, we are observers of the symptoms of understanding as we cannot possibly represent understanding in its true form (Thompson, 2008), after all, we are not in the minds of the NPs and can only experience the NPs' worlds through their interactions within the scope of the observer's world. Therefore, when this thesis discusses the aspect of 'level of understanding' it means the degree of understanding demonstrated through the NP interactions (movements and dialogue) of Code Puzzles. The degree of understanding is assessed based on the metrics of whether the Code Puzzle pieces are used in the correct context based on the final submitted solution (as, we found, that measuring the time taken to place paper-based pieces was problematic, and also trying to assess whether the piece is placed correctly in correlation to the pieces already placed but prior to submission was also extremely tricky to assess) and based on the dialogue and tone of the NPs when placing the pieces – for example, using the correct terminology for the piece placed and their action matching their description of that action.

A-1 focuses on whether understanding can be quantified and measured, or, whether the concept of a 'level of understanding' is meaningless or unfeasible to represent, whereas A-2 to A-8 assume that A-1 has been achieved and consequently aim to analyse the usefulness, effectiveness and accuracy behind the term.

Aim ID	Aim Description
A-1	Discern an approach to identify and represent an NP's level of understanding of programming concepts and the computational thinking strategies they used.
A-2	Evaluate the accuracy of the level of understanding of an NP by comparing the observer's interpretation of the level of understanding to the perceptions of the NP's understanding of their understanding.
A-3	Determine the best practice for representing the level of understanding.
A-4	Compare whether learners of a similar level of understanding share characteristics in the way they interact with Code Puzzles (learner interactions).
A-5	Discover whether a learner's conceptions and misconceptions about a programming concept can be identified purely on their learner interactions.
A-6	Discover whether a learner's level of understanding about a programming concept can be identified purely on their learner interactions.
A-7	Determine whether a learner's perception of their own computational thinking correlates to their actions and thought processes while interacting with Code Puzzle pieces.
A-8	Determine if there is any correlation between the types of interactions performed and the NP's level of understanding.

Table 2: Research Aims

A-1 is the overarching aim – it aims to discover the best way to obtain, represent and use the level of understanding identified from NP interactions. The secondary aims (see Table 2) aim to further dissect this aim and create more of a specific focus.

A-2 focuses on which approaches to data analysis can give us the most accurate representation of their understanding. It explores whether we can simply assess their understanding accurately by looking at the breadth and depth of their explanations, or, whether a more hierarchical approach is more suited to more accurate and effective readings. This objective also aims to explore whether we can represent the level of understanding by comparing the pre-defined intended aspects of the

object of learning and seeing whether the NP also comments on the same aspects when looking at a specific piece.

In comparison, A-3 focuses on discerning what is the best way of representing level of understanding data – whether it is quantitatively, or qualitatively and what type of data is the most useful. For example, is it better to represent this using discrete or continuous data? It will also explore which current data structures can be utilised to help represent the level of understanding, should this be translated to a software system. This objective also explores whether we need a holistic view of all the interactions, or does it only require detailed analysis of a selection of interactions in order to gain an accurate representation of understanding.

Furthermore, A-4 explores whether the datapoints collected from each participant are comparable to one another – for example, it will focus on whether there are common characteristics or movements across different users when they interact with the puzzles. It also focuses on whether these interactions can be mapped accurately to the representation of their understanding of a related programming concept – if the interactions are meaningless, difficult to categorise, or, whether all movements and comments weigh the same in terms of gaining information about their understanding. This objective also raises the query of whether we can classify movements as an overall pattern, and, whether the overall approach that the NP takes reflects on their understanding of the associated concepts. There is also the other issue of how we identify a characteristic or commonality between two learners – and whether we can, for example, group learners by commonalities as no two learners are the same despite us recruiting participants who have similar exposures to programming. This aim also asks us to explore whether types of movements can be directly compared to another – are there similar reasons for performing similar movements of pieces or are the reasons all different?

A-5 relates to the idea that, in theory, if we can identify an NP's level of understanding we could identify specific gaps in understanding and/or misconceptions of related programming concepts. This is separate from A-6 as, if the research cannot identify level of understanding accurately, it might still be possible to identify various aspects of understanding and concepts related to programming. This aim explores whether an interaction that reveals a misconception would be the same as one that reveals an understanding – after all, a misconception could be argued as an incorrect understanding which differs from a lack of understanding of a programming concept. Additionally, this aim focuses on whether dialogue is more effective than simple piece placement – or whether both ways combined are the best approach for communicating misconceptions or lack of understanding of topics.

A-6 determines whether learner interactions alone are enough to be able to determine the level of understanding of an NP or whether there needs to be an accompanied explanation from the NP on the meaning behind their interactions. If, for example, all learners who place Code Puzzle pieces incorrectly in the final solution area have issues with the related concepts, or, if learners who swap pieces have issues with mixing up concepts, then, there would not need to be dialogue in order to determine their understandings or misconceptions. However, if the level of understanding cannot be purely classified by movements alone then what type of dialogue and what movements best reveal their understanding needs to be explored

A-7 assesses whether NPs have an accurate perception of their own computational thinking thought processes; in the secondary and tertiary study, participants were asked to complete a background questionnaire which asks about the steps they go through to create a program – as the study observes their interactions with Code Puzzle pieces – their thought processes and computational reasoning behind the Code Puzzle movements can be compared to their preconceived notions of how they write programs. While it could be argued that there is a difference between completing a Code Puzzle and writing a program using a development environment, this aim attempts to explore whether there are similarities between how the NP interacts with Code Puzzle pieces and how they program. It should also be the case that if they are generally talking about writing a program, they should at least follow some of their computational thinking processes that they identify when trying to write a Java class.

A-8 explores whether the interactions with Code Puzzle pieces can be classified generally or specifically, and whether these classifications can be directly mapped to either positively or negatively impacting their representation of their level of understanding – in other words, whether the movements themselves are unique enough in a pattern that they could be artificially interpreted with a high level of accuracy.

1.6 Research Objectives

To achieve these aims, and answer the subsequent research question, the following objectives are defined (see Table 3).

Objective ID	Objective Description	Aims Covered
O-1	Survey the current literature on knowledge representation, understanding and the psychology of learning.	A-1 -> A-2
O-2	Document how to represent a level of understanding of an NP.	A-1 -> A-3
O-3	Design an experiment to assess whether the level of understanding of NPs can be identified using Code Puzzles.	A-1 -> A-9
O-4	Recruit and conduct a controlled observation study with at least 20 participants.	A-1 -> A-9
O-5	Record the types of interactions that are performed with Code Puzzles by the participants.	A-1 -> A-9
O-6	Transcribe and analyse the audible dialogue obtained from the experiment using Bruner's functional approach to narrative analysis and phenomenology-focused thematic analysis to evaluate whether using Code Puzzles is more effective than using a traditional approach (questions) to form understanding representation.	A-1 -> A-9
O-7	Evaluate the usefulness, effectiveness and accuracy of using Code Puzzles as a way of obtaining information about an NP's understanding.	A-1 -> A-9
O-8	Assess the reliability of the algorithms for detecting understanding in NPs.	A-1 -> A-9
O-9	Design a software architecture and algorithm for how this could be translated to a piece of software.	A-1, A-3
O-10	Disclose the potential implications and loss of data if there is a shift from paper-based puzzles to software-based puzzles.	A-1 -> A-3

Table 3: Research Objectives

1.7 Research Outcomes

These objectives form the basis of the of the chosen approach, identified in 1.4 and the consequent thesis structure, identified in 1.5. O-1 correlates to the Chapters 1 to 3, O-2 to O-6 correlate to Chapters 4 to 7, and O-7 to O-10 correlate to Chapters 8 and 9.

These objectives led to the research outcomes (see Table 4).

Research Outcome ID	Research Outcome Description
RO-1	Design of a diagnostic toolkit that can be used to analyse the understanding of an NP with a high degree of accuracy (above 60%). This design includes important modifications to the usual format of Code Puzzles.
RO-2	A list of recommendations for how to extract the level of understanding of programming concepts from NP interactions
RO-3	A series of software requirements for how this diagnostic tool could be transferred to a virtual environment.

Table 4: Research Outcomes

1.8 Brief Overview of Research Approach and Chosen Methodology

This research implemented an interpretivist mixed-method approach to designing the studies and collating and analysing the data.

This research recorded scripted, controlled, observations that incorporated a think-aloud protocol with 21 participants over the course of three studies which monitored their interactions with two 2D Java Parson's Problems' Code Puzzles. The scripted, controlled observations utilised passive participation from the observer – where the observer is a bystander and only interacts with the participant when asked a query.

Participants in all three studies answered the same questionnaire, that consisted of two scaling questions and two open-ended questions, after each Code Puzzle was completed. However, in the secondary and tertiary studies, participants were asked to complete a background questionnaire that consisted of open-ended questions on what it is to be a programmer, and a scaling question on confidence; a questionnaire that consisted of scaling questions before each Code Puzzle on their perceptions of the task; and a final questionnaire asking dichotomous questions on how accurate they found the study with the opportunity to write freely at the end of said questionnaire. Additionally, the secondary and tertiary studies also incorporated an immediate follow-up feedback session which employed an unscripted, participant observation that embraced active participation protocols from the observer in order to present real-time feedback on the participant's difficulties and query movements that did not make immediate sense to the observer after the puzzle experiment had taken place.

The researcher transcribed audible feedback and puzzle placement for each of the participant's successful recordings (with one video from the pilot study being corrupted due to technical difficulties). These transcripts were then analysed using phenomenology-focused thematic analysis in order to organise and categorise the data into recurrent themes, and Bruner's functional approach to narrative analysis which aimed to categorise the overall 'story' of the participant and attempt to place context on their speech patterns.

1.9 Thesis Structure

Chapter 1 was an introductory chapter intended to concisely set the scene, and lay out the motivation, scope, focus, aims, objectives, outcomes, approach and research question clearly before the literature review chapters (see Figure 1).

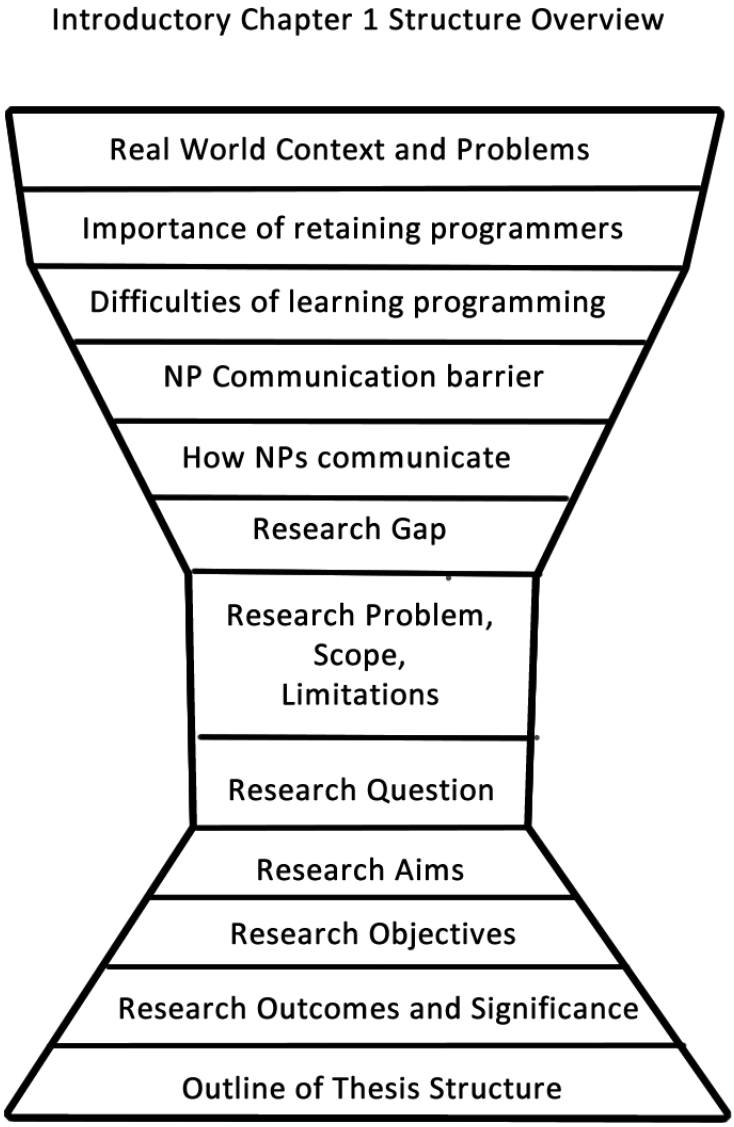


Figure 1: Introductory Chapter's 1 Structure Overview

Chapter 1 identified a research gap involving how programmers communicate and the issues around detecting an NP's understanding of programming (see Figure 2).

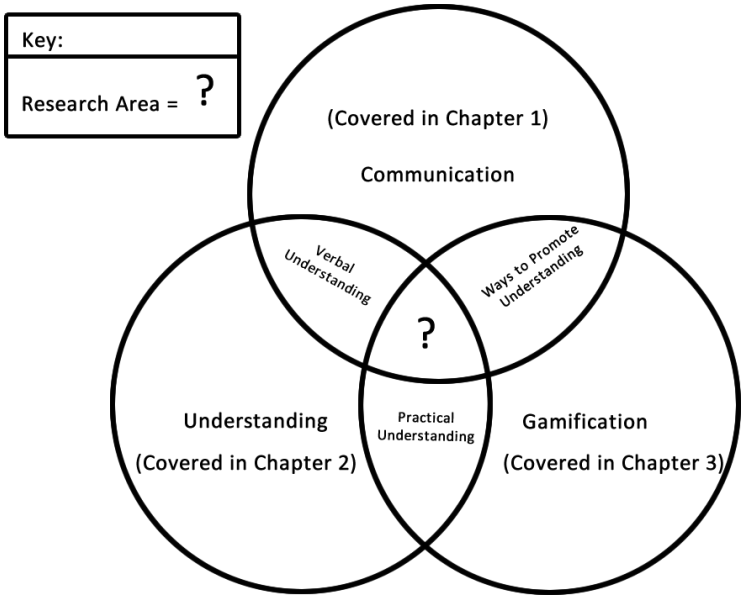


Figure 2: Literature Review Chapters' Focus

Chapters 2 and 3 are the secondary literature review chapters based on Figure 2, that analyse relevant scientific literature associated to the research problem and aim to define, examine and critically evaluate researching into understanding and gamification. Each chapter starts with the general background followed by a refined focus on NPs – where relevant, research that borders on the communication of NPs in these aspects is discussed and clarity on how the methodology was influenced by this past research is specified (see Figure 3).

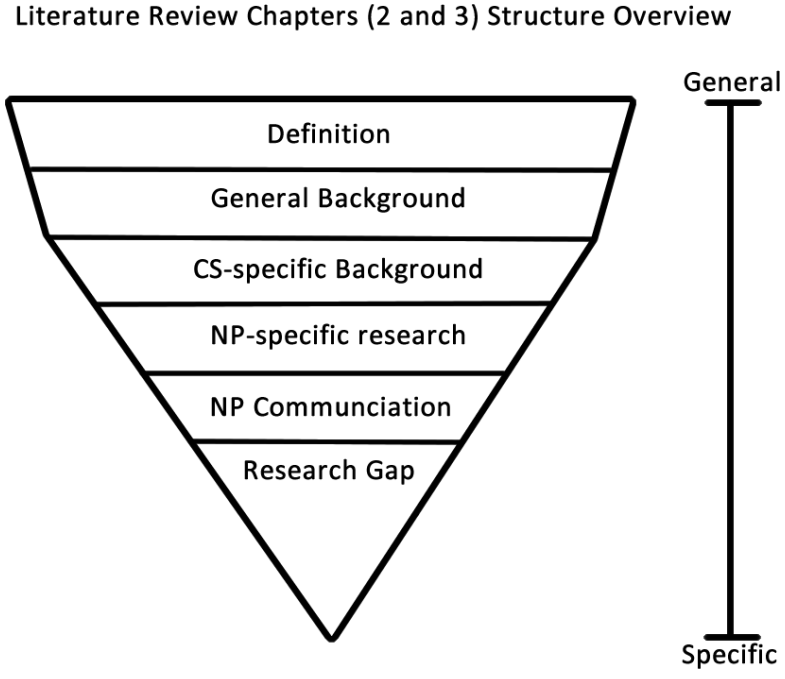


Figure 3: Literature Review Chapters' – 2 and 3 – structure overview

Chapter 4 is the research methodology chapter (see Figure 4) which outlines the general procedures following in the studies written up in chapters 5, 6 and 7 – this chapter identifies, explains and defends the choice of research philosophy, methodology, process and ethics.

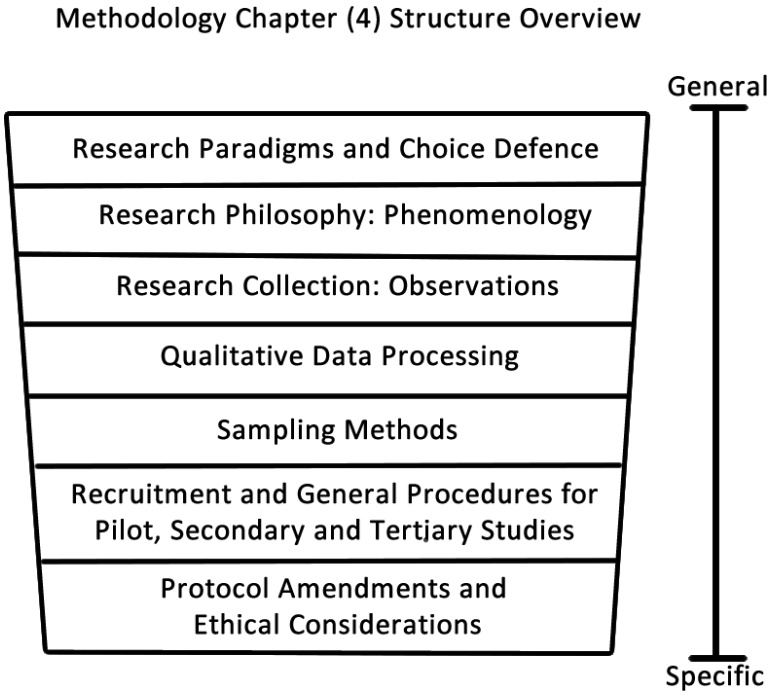


Figure 4: Research Methodology Chapter 4's structure overview

Chapter 5, 6 and 7 are the research study chapters (pilot, secondary and tertiary studies respectively, see Figure 5).

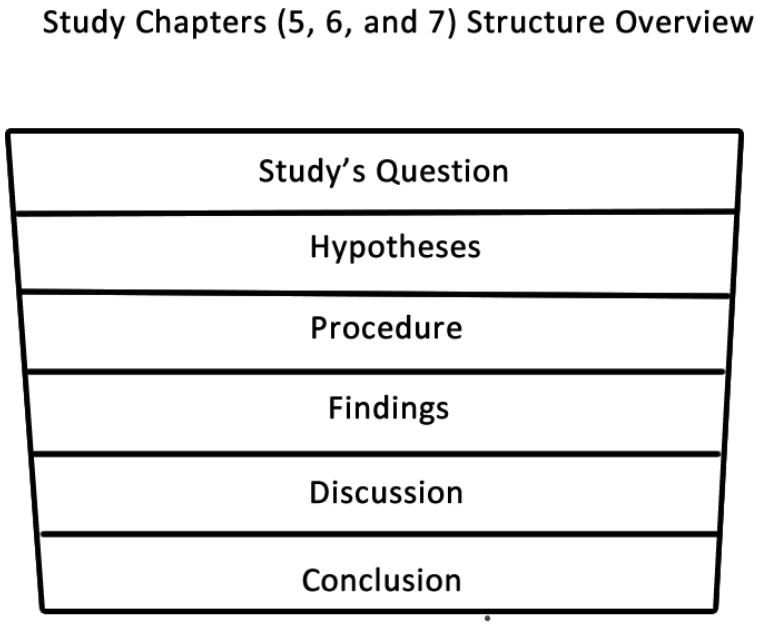


Figure 5: Study Chapters' – 5, 6, and 7 – structure overview

Chapter 8 is the collective discussion chapter – this chapter collectively analyses the findings of the three studies (portrayed in Chapters 5, 6 and 7) and draws conclusions from the findings. Chapter 8 primarily focuses on the findings considering the research question and aims, the focus is heavily based on the explanation of the novel finding – the workspace – alongside the proposal of a list of requirements for a software-based diagnostic toolkit based on the results of the studies (see Figure 6).

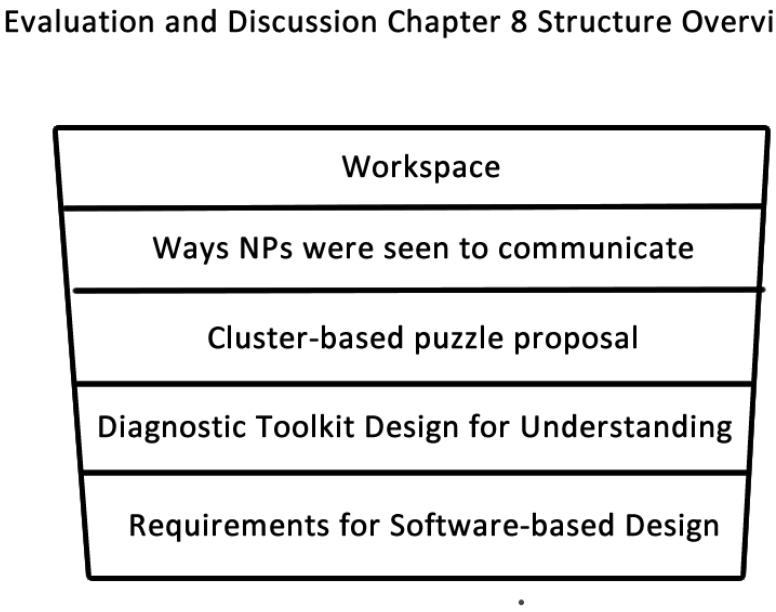


Figure 6: Evaluation and Discussion Chapter 8 Structure Overview

Chapter 9 is the concluding chapter – this chapter reflects on the extent to which the aims and objectives highlighted in Chapter 1 have been accomplished, alongside the intended outcomes of the research using supporting evidence from previous chapters. Chapter 9 highlights the contributions of the studies' findings to the field of CS, NP psychology, and CS pedagogy and reveals the implications for both practitioners and scholars. Chapter 9 reflects on the limitations observed in the studies and propose suggestions for future work based on the analysis of the collective findings and discovery of the novel workspace in Chapter 8. Finally, chapter 9 answers the research question proposed in Chapter 1 and concludes the thesis (see Figure 7).

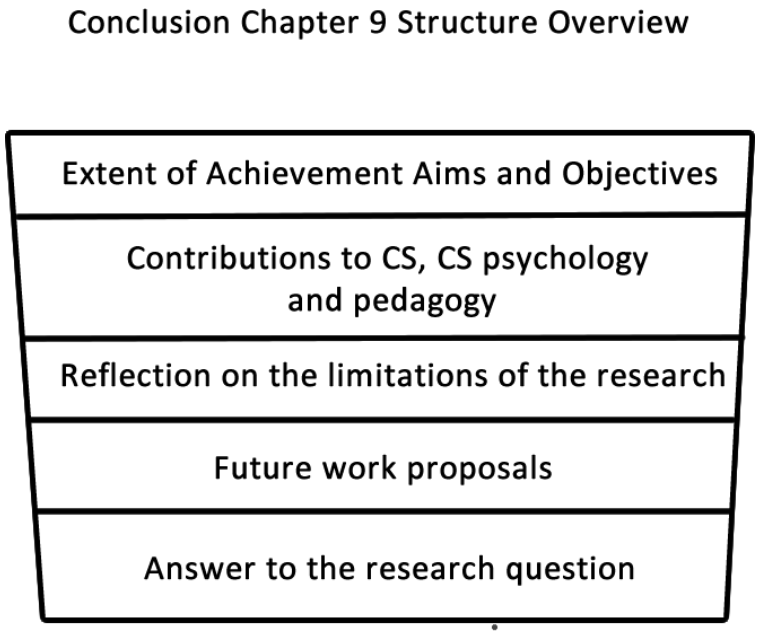


Figure 7: Conclusion Chapter's structure overview

Chapter 10 consists of the bibliography and references, while Chapter 11 consists of the appendices.

1.10 Chapter 1 Summary

This chapter set the foundation for the thesis; it clarified the motivation, scope, question, aims, objectives, and outcomes for the consequent research and ended with a brief description of the subsequent thesis structure.

This chapter started by elaborating on the current issues associated to the UK's technology gap and the worries that the UK will not be able to compete in a global market. It then went onto to explain that, while CS courses do have proportionally higher numbers of students applying to them, the excessively high drop-out rate when we are suffering from such a skill gap must be resolved. While there is not a one-size-fits-all solution to this conundrum, and there will always be some level of people dropping out for non-subject related issues, there are specific issues related directly to the subject of CS that need addressing before this issue can be resolved. We also expand on the natural difficulties of programming, and define what programming is and how each aspect contributes to the overall difficulties encountered by NPs. As a result, this thesis hopes to present research conducted into the specific issue of students being overwhelmed, and consequently frustrated by the prospect of programming – this causes a natural communication barrier to occur between tutor and NP which can contribute to students feeling inadequate and dropping out as a result of their experience with programming and their inability to identify what precisely they do not know.

Using Code Puzzles, which is a common tool that NPs are likely familiar with if they have studied this subject at high school before, we can determine if we can use a friendly environment to detect the NP's understanding and address any lack of understanding or misconceptions present. This medium, if proven successful, has a great potential to bridge the gap between student and tutor without adding to frustration which, could overall, decrease students dropping out of courses for that reason.

With the motivation and context of the situation defined, this thesis identifies the objectives we need to complete in order to make a substantial contribution to this interdisciplinary field of research before explaining the outcomes and the subsequent remaining thesis structure.

As part of each chapter summary that relates to the literature review or background of the research, this thesis will have a summarising diagram to help provide a quick reference to the important aspects portrayed in the chapter overall.

Chapter 2. The Challenges of Identifying the Understanding of NPs

NPs need to be understood to identify and remedy issues that occur from poor understanding. To understand NPs, we need to explore what 'understanding' is and how mental representations of coding are formulated. Therefore, this chapter will start by exploring what understanding is and how mental representations are formed in learners in general and – where possible – these theories will be applied to examples of programming with NPs. The chapter will then discuss the findings of the studies that have investigated understanding NPs and the produced mental representations from those studies in order to determine how this research should discover the understanding of NPs. This chapter ends with the thesis' interpretation of the definition of understanding in the context of NPs and what characteristics of mental representations of programming will be focused on during the investigations.

2.1 The Philosophy of Identifying Understanding and Knowledge

According to Soanes and Stevenson (2005), the term 'understanding' can be defined as a noun or adjective whereby a person has "insight and good judgement" on a given subject, and is also used to imply "sympathetic awareness or tolerance" in regards to another person's situation; however the verb, 'to understand', can be defined as the "correct perception of the intended meaning of the words that the speaker used" or to "interpret or view a subject in a particular way" (Soanes and Stevenson, 2005). Interestingly, most of these definitions imply that the person who understands a subject will have the 'correct' perception of the subject matter, suggesting that in order to be classified as understanding a subject, one's perception of that subject should co-align to the expectations of the one delivering the subject matter. However, the secondary definition of the verb form, implies that there are different ways that someone can 'understand' a subject and that these ways also contribute to the term of understanding.

Therefore, it can be determined that the actual meaning behind the word 'to understand' is not a simple task to comprehend in itself; according to Scardamalia and Bereiter (2006), true understanding is only obtained when a person can think about, and use concepts from that subject, to deal with relevant or connected situations where knowledge of that subject would be useful. However, this definition only refers to the outcome of understanding, not the process of creating understanding itself.

For many centuries, philosophers and educationalists have debated what the essence and structure of knowledge, thought and understanding is, and these debates have formed the basis of psychological and sociological research which needs to be briefly explored in order to understand the

origins of how identifying understanding came to be. Aristotle explored the concept of knowledge and understanding in his works of The Concept of the Mind, The Problem of the Four Universals and The Four Causes in his book, 'On the Soul' (translated by Smith, 2016). In summary, he postulated that in order to understand the mind, one needed to understand the soul; the soul Is defined as "the internal principle of motion or change in a living being" and that there are different levels of the soul which indicate the capabilities of the host.

In Aristotelian philosophy, the concept of understanding originates from the rational part of the soul – and the part that allegedly differentiates us from animals – in the way that we can form societies and formulating our own understanding of the world based on our lived experiences. This philosophy forms the basis of sociology, and the chosen focus for this research where we are attempting to extract and identify the understanding of NPs. One issue with the concept of the mind is that it does not consider the age or experience of the human being which does affect their ability to use their rational soul. For instance, an NP is likely to have more difficulty identifying and understanding extracts of code than an adept programmer. Therefore, it is important to remember that in the context of this research we are interested in their first actuality – that is, the specific conditions of their current state, i.e., being a novice – as opposed to purely their first potentiality – that is, their type of being, i.e., being a programmer. The concept of the mind also presents an interesting notion: that for the rational part of the soul to develop, the vegetative and sensitive portions of the soul require nourishment; when examining the capabilities of an NP – or any learner, for that matter – their background, health and outlook can reflect on their ability to process information from the surrounding world. Therefore, it is important to treat NPs and learners as individuals when studying them as each student is likely to have different experiences and conditions in their souls. Aristotle also considered aspects of how a person could learn concepts; in his book 'On the Soul' he determined that a learner would need to understand a universal you needed to understand the origin of the universal. A universal is a characteristic, or aspect, of the subject that should be learned and is similar in concept to Marton and Tsui's (2004) Variation Theory regarding identifying the 'critical aspects of the object of learning'. Variation Theory is a proponent of the idea that for effective learning to occur the teacher needs to alter the way the learner views the topic – known as the object of learning – through focusing on 'critical' or important aspects of the object of learning (i.e., the desired parts of the topic that the teacher wants the student to learn about). In essence, Marton and Tsui (2004) argued that without variation of each of the critical aspects there is "no discernment". Aristotle emphasised the importance of also understanding how the universal came into existence, and argued that there needed to be comprehension of the four causes of a universal in order to understand the concept itself: Material – the features that make up the object, for

example, the programming language of code; Formal – the design or form of a concept, for example, a quick sorting algorithm; Efficient – the implementation of the concept or if the concept was not immaterial the driving force behind its creation, for example, using a development environment and relevant programming language documentation to type the code relevant to the sorting algorithm selected; and the Final State – when it has fulfilled its intended purpose, for example, the code has been deployed and is being actively used in a wider part of the program. This can be transferable to programming, as shown, and highlights the value of an NP needing to understand the programming language, programming paradigm, how to design their code, how to implement their code, how to test their code and how to check whether it fulfils its intended purpose. The Four Causes argument is not perfect; Aristotle himself even noted that not all perceivable objects have a final cause – such as a never-ending event or scenario such as war. In the context of programming, many online systems undergo maintenance and updating of legacy code, arguing that there is never a true final, unchangeable, state of a program. However, studies such as Pérez-Álvarez (2017) have utilised Aristotle's Four Causes as a way of examining psychology phenomena. Plato argued that only generalisable, always true, knowledge is valuable – such as facts – and that applying the Four Causes to specific situations did not yield this type of knowledge as there would only be understanding of that specific instance of the concept, for instance, in the example of implementing a quick sort algorithm in programming if the programming language changed – as in the material state – the programmer would not be able to claim that they understand quick sorting algorithms, according to Plato. However, this thesis argues that the true essence of understanding is understanding each part of the example in order to become a proficient programmer – so variations in the way a critical aspect is perceived is necessary (Marton and Tsui, 2004). Aristotle formed the basis of the term 'mental representation' by thinking about the concept of the mind and how it stores information.

2.2 Cognitive Models: How are mental representations formed?

In the field of CS, cognitive models are defined by Lane (2012) as "descriptive account[s] or computational representation[s] of human thinking about a given concept, skill, or domain" whose main purpose is to help explain reasoning behind actions and thoughts on subject matter. Bayman et al. (1983) further defined mental models as "the user's conception of the 'invisible' information processing that occurs inside the computer between input and output" in other words – mental representations are how NPs view programming. Mental representations are formulated by experts identifying characteristics present in their data. Typically, interpretivist research attempts to extract the 'true essence' of a participant's viewpoints, which are then categorised into characteristics.

However, Cropley (1967) warns that studies focused on 'cognitive styles' are concerned with how a participant thinks (the logic behind their actions) rather than what a person thinks (their perspectives on the world). Warr et al. (1970) agreed with Cropley (1967) and proposed that studies should carefully select phenomena to explore so that participants could draw upon their knowledge (what they think) and demonstrate their actions (how they think) toward the phenomena. While the distinction between knowledge and actions is appropriate for topics that do not require a practical application, discerning an NP's understanding requires identifying their thoughts and actions. Consequently, an amalgamation of Bayman et al. (1983) and Lane's (2012) definition of mental representations will be used in the context of this thesis as it is more broadly applicable to programming than the other definitions.

Bieri et al. (1966) suggest that it is possible, and even necessary, to separate the content of the cognition from the underlying structure of the mental representation, and that if this separation is achieved, the "knowledge of cognitive structure implies that predictions can be made of the way in which the person copes with his environment". This suggests that research into mapping the interactions of learners can be classified as mapping their cognitive styles, and from such data, perhaps anticipating their approach to other programming problems.

In the fields of sociology and psychology, multiple scholars support the importance of researching and documenting the differing cognitive models in order to derive understanding. For example, Scott (1963) suggested that 'cognitive styles' were crucial to understanding in order to derive meaning from the world around us. Therefore, recording characteristics and deviations of mental representations is crucial for researchers who wish to explore understanding how participants view phenomena.

2.3 Mental Representations found in Programmers

Wiedenbeck et al. (1993) proposed that there were five characteristics of mental representations evidenced in the experts they studied, namely: hierarchical structure, explicit mapping of coded goals, recognising recurring patterns, knowledge, and interpreting the program text. Wiedenbeck et al. (1999) compared the findings of the mental representations extracted from expert programmers in the Wiedenbeck et al. study in 1993 to NPs in the Wiedenbeck et al. (1999) study and found that the experts' responses showed signs of all given mental representations, whereas NPs generally lacked some of the characteristics or had poorly developed versions of the characteristics identified. Bayman et al. (1983) arrived at similar conclusions during their study where they asked programmers with different levels of experience to construct BASIC statements and noted that NPs "possessed a

wide range of misconceptions concerning the statements they had [self-]learned" suggesting what may seem obvious – that NPs are more likely to have poorer mental representations of programming than more experienced programmers.

Which starting programming language and paradigm best encourages the development of 'good' mental representations for NPs is a topic still subject to debate. Wiedenbeck et al. (1999) conducted two studies on second year undergraduate CS students who had learned languages in procedural and object-orientated paradigms to compare the quality of comprehension between the two cohorts; in their study, programmers were given pre-constructed segments of Pascal code and asked to answer questions on the code. Wiedenbeck et al. (1999) discovered that object-orientated participants' responses in the short version of the programme showed signs of superior mental representations for the subject of functions when compared to procedural participants' responses. However, for longer programs no notable difference in quality of mental representations was shown and in all other questions aside from those about methods, procedural knowledge was considered slightly superior. Wiedenbeck et al. (1999) concluded that their data suggested that the object-orientated paradigm had a steeper learning curve than procedural.

More recent research has suggested that NPs starting with imperative and procedural programming paradigms contribute to the formation of 'good' program models, but that NPs starting with an object-orientated programming paradigm may develop better situational models (Alardawi and Agil, 2015). These findings in past works are relevant to note when investigating mental representations, as the NPs investigated in this thesis' research had supposedly learned just Java; an object-orientated language, however, the survey responses from the secondary and tertiary studies suggest that some participants in this research had learned multiple languages with differing paradigms. The different impact of programming paradigms on mental representations for NPs can be explained by Alardawi and Agil's (2015) observations of how "different programming styles have different effects on the mental representation constructed by subjects during comprehension process" and in turn, the mental representations created by the NPs can consequently vary based on the programming paradigms they have previously had experience with.

Mental representations in programming, therefore, require context; while most of the research tends to focus on procedural programmers, it is relevant for the researcher to be aware of the mental representations produced by these studies. Mosemann and Wiedenbeck (2001) suggested in their research that both the mental representation of a programmer and the way they comprehend the task is crucial in order to understand the way a programmer views the world – in their study they explored how procedural programmers navigated through a program to determine which forms of

representing the program's internal logic were most crucial for facilitating programmer comprehension. Mosenmann and Wiedenbeck (2001) found that sequential flow of the order of which commands are executed in a program was superior to showing a data flow – how data changed states. This suggests that programmers are more focused on the logic behind the statements in a program rather than the data itself. In more recent times, Scott (2010) supported the concept of using flowcharts with NPs as an effective strategy for improving their mental models of a program. However, no known research has been conducted into the effectiveness of using game-based puzzles to determine the mental representation of an NP. Another noted issue is that cognition is based upon perception of the subject matter itself, and this can be explained by the philosophy behind phenomenology and phenomenographic analysis. The principle that a person can view the same phenomenon as another person and interpret it in a different way, thus forming differing mental representations from the same stimuli is widely accepted among practitioners of phenomenology.

2.4 Pedagogical Frameworks (aimed at influencing and examining mental representations)

Pedagogy is referred to as the field of studying educational practice, and cognition is consequently an important topic for practitioners in the way that the student's mental representation of programming concepts is what educational CS practitioners want to improve the quality for, and consequently assess.

Pedagogists explain and try to improve how learners create mental representations of the knowledge from their environment, or space of learning. According to Ihantola and Karavirta (2011), it is "commonly agreed that students' active participation in exercises are essential for learning programming" arguing that the student needs to be a co-creator of constructs and involved in the process of generating a mental model of the intended object of learning. This links to Aristotle's idea of the concept of the mind, in that the student needs to have their needs satisfied and distracted as little as possible in order to be able to use their rational soul. However, tutors do not have enough time to be able to provide individual feedback to every NP they encounter, which is why automation of any part of the learning process is considered desirable (Leong, 2015). The idea of automating the learning process goes against the concept behind interpretivism – where every person views the world differently and therefore has a different way of interpreting the environment around them which affects their cognitions. That said, the whole premise of assessing students based on their understanding implies that there, realistically, needs to be a way in which understanding can be amalgamated as practitioners and employers need a way to measure the understanding of programmers in order to be informed about who has a 'good' level of understanding, and who will be

employed out of a series of candidates. Therefore, while the individual has a potentially unique perception of reality and differing degrees of understanding of that reality, there has to be a way for us to be able to assess whether those perceptions are as close to the true meaning of the phenomenon as much as humanly possible.

In programming, understanding is often viewed from the perspective of having a hierarchy of knowledge and skill; for example, Lister (2004) explained that tutors and students need to be aware that programming consists of a "hierarchy of knowledge" which the tutor must be consciously aware of to overcome threshold concepts – also known as concepts that once learned promote a deeper level of understanding – in order to teach programming effectively. This hierarchy of knowledge can be related to the depth or level that a characteristic of the mental representation is recorded at. Lister (2004) provided a concrete example of the programming knowledge hierarchy in that, before an NP can write code (i.e., 'Can you write me a method that calculates the price of the sale?'), they need to have the ability to trace code (i.e., 'what value does this method output when the code is executed?'); and that before they can trace code, they need to be able to read code (i.e., 'can you explain, in English, what this method does and why?'); and before they can read code, they need to understand the relevant syntax (i.e., 'can you explain why the semi-colon is needed at the end of the line?' and so forth). Lopez, Whalley, Robbins and Lister (2008) supported Lister (2004) and argued that understanding of programming was hierarchical and that practitioners needed to begin from a starting point of reading the code, then explaining the code, and then writing the code. Lister (2008) repeated the study five years later and found similar conclusions. Originally, Lister (2004) further added that more research is required into streamlining the process of learning to program, and current data on the drop-out and failure rates of programming provided in Chapter 1 suggests this research is still relevant in modern times.

The purpose of SOLO Taxonomy is to provide a framework for practitioners to assess the quality and levels of work by learners (Biggs and Tang, 2011). SOLO Taxonomy can also be used to guide instructors on how to structure their object of learning in a way that allows the learner to understand the relationships that exist between programming concepts (see Table 5 for an example based on Biggs and Tang, 2011).

SOLO Taxonomy Classification	Example of the type of queries an NP of this classification could say:
Pre-Structural	"What is programming? What is a program? I see code, I think… but I'm not sure what language that's in".
Unistructural	"This is clearly a program because it is written in Java".
Multistructural	"This is clearly a program written in Java, the program sorts a randomised array of numbers, outputs them in order of size and then the GUI updates".
Relational	"So, in order to re-order the IDs of tasks linked to a GUI in the program, I need to arrange the numbers in order with a sorting algorithm and write the code in Java, and then, when the output is given update the GUI so that the user can see them in the order selected. As, from my experience, reordering of tasks based on priority and order of creation would be useful and I can see the purpose behind this function".
Extended Abstract	Same example as relational, except this is added: "I wonder what sorting algorithm would be most time efficient? Perhaps I could create different sorts of functions and analyse the time complexity. It's not specified in the task to do this, but I am curious and as a user I would want the screen to reload quickly".

Table 5: An interpretation of SOLO Taxonomy (Biggs and Collis, 1982; Biggs, 1995; Biggs and Tang, 2011) applied to NPs

Ihantola and Karavirta (2011) equate the process of teaching NPs how to program to teaching a learner how "to see trees that make up a grand forest" – this beautiful imagery can be related to how understanding has different levels. Lister et al. (2006) supported the idea that programming concepts can be viewed hierarchically and postulated that SOLO Taxonomy can be equated to the metaphor of the forest – where unistructural answers may only appreciate seeing part of a tree (i.e., the branch or leaves) without seeing the bigger picture, whereas a multi-structural understanding may see individual trees but not understand the necessity for diversity or their role in the ecosystem, and relational and extended abstract can truly appreciate the varying depth and roles of individuals of the trees in the forest. Ergo, it is possible that the understanding of a programming concept for an NP may be at a deep level, but if their appreciation for the wider context is lacking then their ability to apply and adapt a programming concept to be relevant in a situation requires a deeper understanding making SOLO Taxonomy a relevant structure in programming.

Consequently, it can be suggested that programming tasks may have a hierarchy of difficulty depending on the type of question asked, and the programming concepts themselves can also be represented hierarchically based on the depth of knowledge. The difficulty of the task in programming as well as the required knowledge to achieve that task, therefore, is crucial in order to enhance understanding and not overwhelm or scare the programmer – but what about the reverse – extracting understanding? As this research aims to identify understanding, effectively reversing the

equation. In theory, assessments aim to measure the understanding of a candidate; if an assessment was aimed at the correct level for the anticipated learner, then, it should be possible to gain evidence of that level of understanding based on the learners' responses. To identify understanding we must first look at travelling up the hierarchy – so ask tracing, explanatory and writing questions in that order. However, examining their answer, by itself, would give little understanding of how they arrived at that answer without the accompanying explanation of how they are tracing the code – which, arguably, gives more insight into, why, for example, they arrived at the wrong value for the method-based example question. Imagine, for instance, that an NP is stuck and unable to answer a simple tracing question or cannot explain why they arrived at the answer they got for the tracing question – how would the tutor be able to extract misunderstandings or misconceptions from a learner's thought processes if they were unable to vocalise what those thought processes were? It is possible, therefore, that the tracing and explanatory questions could be merged into one; where an NP traces while explaining their thoughts out loud. On the other hand, this may actually cause their cognitive load to drastically increase due to the amount of mental effort required in order to both trace and explain their processes at the same time, with participants focusing on the step-by-step process rather than considering their reasoning behind why they are approaching such a process in such a way.

There are numerous differing types of assignments and assignment frameworks which are based on this theory of hierarchical difficulty – with some tasks, such as fill-in-the-blank worksheets and simple tracing questions – aimed at the lowest difficulty. Some assignments, particularly assessments that contribute to an overall mark, usually attempt to use a mixture of low-level, intermediate-level and advanced-level questions in order to assess whether a programmer can understand, explain and write in the language that they are being assessed on (Ihantola and Karavirta, 2011). Written examinations usually contain a mixture of open-ended questions for this reason, and more recently, tutors have been looking into alternate ways to examine programming – as, for example, examining the quality of code that a student can write off-hand is at an advanced-level and may not be possible for an NP that is already having difficulty describing what a standard program does.

Scholars such as Eckerdal and Berglund (2005) and Hsu and Wang (2014) support a framework based on this interpretation of cognition, known as Variation Theory, arguing that it is a "beneficial" framework for encouraging the structuring of subject material in a way to enhance the learner's potential to experience a given subject matter, known as the enacted object of learning. As observed by Alardawi and Agil (2015), some of the studies that were conducted specifically into object-orientated programming were in agreement that it was considered vital for teaching practices to understand both the origins and formation of NP's perspectives on learning, and through this

appreciation, they can begin to understand how and why misconceptions and misunderstandings arise and use these models to help inform their teaching practice.

2.5 Thesis' Definition of Understanding

In the context of this thesis, the researcher has determined that the definition of understanding of NPs shall be defined as: the level of correctness in the way in which an NP perceives, approaches, comprehends and chooses perceivably relevant programming concepts based on their own mental models of programming and how they apply these concepts in practice, i.e., to create a program for a given task.

2.6 Research Gap: Can gamification be used to quicken the process of identifying understanding accurately?

There is a notable research gap identified in this literature review; as no research was discovered on whether these cognitive representations could be extracted from NPs using gamification – i.e., programming using fun, popular activities such as creating programs from arranging a series of code blocks such as code puzzles – as the discovered studies asked their participants a series of questions regarding a concept rather than observing practical implementations. This thesis argues that programming, as defined in Chapter 1, has a process inherently inbuilt into it and in order to understand an NPs' perspective there needs to be some appreciation for how they approach and solve a program alongside their understanding of programming concepts. This research, therefore, will explore whether gamification can be used to extract understanding accurately to address this gap in research knowledge.

An investigation into whether observing the way NPs construct code will reveal their understanding is warranted to deduce whether NP's mental representations can be generalisable enough to be compared to one another. Scholars such as Wiedenbeck et al. (1999) and Alardawi and Agil (2015) indicate that the NP's starting programming paradigm may affect their mental representation, and the definition of what a 'good' mental representation is regarding programming is subject to debate still. This research only had access to participants who were guaranteed to know Java (an object-orientated language) – not necessarily other programming paradigms – but had no control over what other languages the participants' cognitions may be influenced by; whether the mental representations present in the data were like that of Alardawi and Agil's (2015) work would help to support or disagree with the current thoughts about the effect of programming paradigm(s) on an NP's psyche.

2.7 Chapter 2 Summary

This brief chapter aims to explain the concept of what understanding is from a relevant psychological perspective (specifically to programming, as, beyond the topic of programming is outside of the scope of the thesis). We identified how important mental representation is for understanding NPs, which is crucial to explore and compare our findings to these mental representations to see which representations we have supporting evidence for, and which ones we have supporting evidence against.

Chapter 3. The Potential of Using Gamification to Identify Understanding

Gamification is defined as the practical application of game design features in non-game domains (Deterding et al., 2011). According to Almadia et al. (2021), the term 'gamification' "remains inconsistently used [,] a general theory of gamification is yet to be developed [and there are] so many different definitions, discrepancies, distinctions, and discretionary delimitations [in observed gamification papers]" that it makes the topic difficult to explore. Almujally and Joy (2020) argue that gamification can be used in the context of Knowledge-based theory to promote sharing of knowledge in the knowledge management process and that higher education institutions need to enhance their knowledge sharing in order to provide an effective teaching experience to their students.

There are some recent studies that have focused on how gamification benefits students of all disciplines, as well as novice to advanced programmers. For example, Ahmad et al. (2020) focused on whether using group-based gamified activities was an effective strategy for teaching higher level CS students, with a particular focus on whether students retained long-term understanding and were satisfied with their learning experience. Ahmad et al. (2020) concluded that gamification was an effective tool for "tough courses", but that group size affected the quality of the learning experience. Zhang et al. (2020) also proposed that gamification can be used to achieve satisfactory results for teaching "abstract and uninteresting" and yet important concepts in CS. Carmo et al. (2020) investigated the student's behaviour and impact on performance through a gamified tool named 'learning paths' which are "sequences of learning objects followed by students" they found that students who interacted more with the course had better grades. Although there is likely a prevalence between hard-working students and being more engaged with the course, Zhang et al. (2020) note that "students pay more attention to the fun and ease from the game" implying that gamification could be a way to motivate interaction with the learning content for students struggling with traditional text-based content. Chou (2019) further supports the idea that gamification is designed to be "fun and engaging" with good gamification concentrating on "Human-Focused Design" to motivate and reward the user. According to Almadia et al. (2021), gamification became popular when some educational institutions adopted the mantra of "fun at work" in the early 2000s – but are games meant to be 'fun' and should learning be 'fun' all the time? Almadia et al. (2021) highlighted that when an education-based game is done 'right' then "applying gamification to education and learning systems represents a promising means to allow educators to make learning fun, contextualize learning quickly, speak the language of young people, and directly deal with soft skills, [and] improv[e] education quality", however warn that if done incorrectly can cause "harmful effects". Interestingly, Almadia et al. (2021) discovered that the most common subject for negative

effects of gamification to be incorporated was 'Computer Science' (with a total of 27 negative papers) followed by medicine (with a total of 6 papers) however this may be easily explained by computer science being the most closely related subject. Yet Ivanova et al. (2019) argued that gamification is "extremely effective with school students and standard subjects, such as STEM, when applied correctly" which suggests that 'correctly' is important and that perhaps delving into the documented ways on which it has been unsuccessfully applied may shed light onto how we can avoid replicating the same issues. Almadia et al. (2021) explored papers on the potentially harmful effects of incorrectly applied education-based gamification and discovered that "the impact of gamified interventions was found to be positive by 59% of the [77] papers reviewed, with effects including empowerment, motivation, health monitoring, and more healthy habits taken". Zhang et al. (2020) further add that if done right, gamified elements may "inspire [students] to think, discuss and innovate in the topic taught by the game. However, Almadia et al. (2021) state that "41% – a significant portion of the studies [from the 77 papers reviewed] – reported mixed or neutral effects" suggesting that the benefits of gamification were not seen in all contexts – including the aspect of 'fun at work'. Hammedi et al (2021) even suggested fun should never be "mandatory" unless the designer wishes for the game to fail by causing users to become disinterested in the game. Almadia et al (2021) observed that: badges, competitions, leaderboards, points, challenges, achievements, quizzes, experience points and levels were the top ten named educational game mechanics that were reported to have negative effects in the papers reviewed, but that a large majority (59 mechanics) were labelled as 'others' so it is difficult to tell whether Code Puzzles rank among the failed instances as they are not recorded on the list. Let us take the example of badges – the most frequently negatively reported mechanic – the main observation for badges were that they were not observed to improve motivation, learning quality and were a source of technical difficulties making them irrelevant and potentially distracting to the user. These findings may implicate that competitive and process-monitoring gamified elements are not suited to educational contexts – which is surprising, considering that many educational systems revolve around an assessment-heavy, mark-orientated approach to learning and perhaps reveals the way in which we teach our subjects (computer science) included may be detrimental and impede a student's desire to naturally engage with the content. Almadia et al (2021) documented the common negative effects were seen across at least 5 papers (with the first negative effect being most common in the list): lack of discernible effect or impact in comparison to other tools; lack of observed learning enhancement and lack of a noticeable increase in understanding from playing the game; irrelevance of the game mechanic to the topic; users finding a lack of motivation to use the game; users losing motivation to use the game after a while due to it losing its novelty; loss of performance; users feeling the need to cheat to get,

say, a better score, or learning how to "game the system". While these effects are negative, the context where these effects arose would be needed to discuss why these effects were occurring which the Almadia et al. (2021) paper does not disclose as they chose to focus on the frequency of the effects, yet these effects are likely linked to the concept that gamification influences human behaviour and that if, say, a game mechanic was influencing the player to 'win' then some players may be prone to trying to cheat to feel the benefits of the reward without the effort of learning.

The design of gamified tools can be challenging; researchers like Voit et al. (2020) warn that the "ineffectiveness of many gamification projects [originate from] wrong decisions made during the conceptual design phase, especially in the selection of game design elements" and have consequently been using machine learning techniques to extract game design techniques from over 30,000 board games to help game designers design tools more effectively. As such, this thesis will investigate popular forms of gamification tools used in the domain of CS that most closely correlate to the perspective of gaining knowledge from the student's interactions rather than enhancing the student's learning experiences.

However, there is limited research on using gamified tools to 'translate' understanding directly to a tutor of any discipline. Studies from a tutor's perspective tend to focus on whether gamification can be used to disseminate knowledge to other staff or to students effectively rather than the reverse. For instance, Almujally and Joy (2020) wanted to see whether a gamified tool could promote knowledge sharing between staff and discovered that the "quality and amount of knowledge [the 20 CS staff members] shared strongly depended on the feedback they obtained from the gamification mechanisms which were provided" and concluded that gamification was a way to encourage staff to interact with others, but this research does not relate to the student experience.

There is also limited research into the effects of removing gamified elements, and there is no universally agreed methodology for how to optimally decrease gamified elements to readjust the NP to a more realistic interface. Seaborn (2021) noted that "a small corpus of 8 papers [have been] published between 2012 and 2020" on such a topic, which raises the question of how tutors can best ease NPs into authentic tasks from gamified tools. While this thesis agrees that this is an underdeveloped area that requires research, we argue that programmers will naturally encounter different development environments and tools that will mean that they have to adapt to interacting with code in a flexible way and should not devalue the benefits of gamification as a tool for both learner and tutor. For example, Scratch (2020) and BlueJ (2020) have quite a different interface to IntelliJ (2020), Eclipse (2020) or Netbeans (2020), which has quite a different interface to Jupyter labs (2020) but all of them 'interact' with code and require an adjustment to a new interface.

Researchers have discovered that using code puzzles, a form of gamification, as a way of assessing understanding is more time efficient than alternates. For example, Denny et al. (2008) suggested that they correlate well to traditional exam-style questions with less effort from the marker and are assessed at a similar difficulty level. This chapter explores potential alternatives that participants could use as talking points for communicating their understanding of programming to the observer.

Code Puzzles are gaining popularity, particularly among younger programmers, as a fun, engaging, alternative way to learning programming where a learner utilises active participation in order to learn programming concepts.

3.1 Parson's Puzzles

Parson's puzzles are a type of Code Puzzle where lines of code from a pre-made solution are translated into moveable code blocks and are presented to the user in a randomised order (Parson and Haden, 2006). Anecdotally, the name of these puzzles varies from source to source – while the original author's surname, Parson, is singular many articles and papers have used Parsons' instead of Parson's when referring to the same form of puzzle. Hybrids of this origin have appeared, including Parsons' puzzles, Parson's puzzles, Parsons problems, and even just 'Parsons'. As the original paper cited the name as Parson's puzzles, this thesis will endeavour to call them this from now on.

There is evidence to indicate that students find such puzzles engaging, and in Parson's and Haden's 2006 study, 82% of 17 undergraduate students indicated that Parson's problems were "useful" or "very useful" in a post-study survey. Ihantola and Karavirta (2011) argued that such puzzles could help learners recognise common algorithms, as well as proposing that different varieties of Parson's problems may be used to increase the difficulty of the task. This evidence was further supported by the theory that NPs lack the mental representations necessary for programming and benefit from a more scaffolded approach (Winslow, 1996). Morrison et al. (2016) argued that Parson's problems are suited for NPs as they contain "correct syntactic constructs and impose low cognitive load" on NPs. Fabic et al. (2019) observed that "Parson's problems provide scaffolding helpful for novices, unlike in traditional code writing exercises where the only scaffolding is the problem description".

While the exact hierarchical place, in terms of programming difficulty, for Parson's problems has been widely debated, there is some suggestion that Parson's problems may be lower-level than tracing exercises in terms of the difficulty hierarchy (Lopez, 2008; Fabic et al. 2019), whereas other research has suggested they are similar to advanced-level tasks as it requires the programmer to arrange pieces as if they were coding it themselves (Denny et al., 2008). Some researchers, such as Ihantola and Karavirta (2011) are inconclusive about the precise level of Parson's problems in terms

of the learning hierarchy and believe that such a hierarchy is affected by the task difficulty rather than the design of the Code Puzzle itself. Lopez (2008) did suggest, at the time, that the data collated from their research may be affected due to the differing complexities of the tracing tasks they administered to the participants. The original design of Parson's problems was to provide immediate feedback to the user, which, while good for users who are struggling may be not as beneficial for those who are merely relying on the computer to tell them if the ordering is incorrect. This can cause the opposite phenomenon to active participation, where the learner is merely attempting to complete the puzzle in the quickest time possible without engaging in the core computational thinking processes required to understand the puzzle itself. This is further supported by the findings of Helminen et al. (2012) where they found that student's retention of knowledge over several weeks was far from optimal, with many failing to remember core concepts associated to the practice puzzles that they had engaged in.

The original purpose of Parson's problems were that they were developed in order to provide an engaging, automated, learning environment with immediate feedback to students and have gained popularity with interfaces such as Scratch (Scratch, 2020) gaining some inspiration from this style of puzzle. These types of puzzles are widely used and available, in a variety of different languages with institutions such as MIT providing example snippets of Code Puzzles and libraries in order to create your own puzzles. The beauty of Parson's puzzles is that the code does not need to be executed in order to check whether the code works, it simply checks whether the code is in the correct order or slot with many interfaces choosing to give feedback through a change of colour or pop-up notification.

There are different variations of Parson's problems. Ihantola and Karavirta (2011) produced an overview of different styles and features of Parson's Problems, including: extra lines or distractors which, as per the original Parson and Haden (2006) paper, are meant to be added to increase the difficulty of the puzzles; user-created blocks which, if adopted, typically mean users can indent or insert braces into their code, which, according to Denny et al. (2008) increases the complexity as well; and context which is provided in order to arrange the code blocks – the difficulty increase or decrease is inconclusive on this particular aspect. In Parson's Problems, extra lines (or red herrings) are called 'distractors' as they are segments of code that could potentially fit but on closer inspection would not fit the context of the solution. If the researcher chooses to use distractors, there are two common ways they are incorporated: the first way is to initially present the pieces in the incorrect order but have lines that look similar to one another – with one correct piece surrounded by distractor piece(s) – placed together in a 'group' so that the user can easily compare pieces of a similar nature; and the second way is to have all pieces initially presented randomly – so distractors

may be separated from the correct version of the distractor as no groups would be present due to full randomisation of the pieces. According to Ihantola and Karavirta (2011) the latter approach reduces the cognitive load of the programmer whereas randomising the distractors in the code causes more of a mental strain. The number of pieces available to the user likely affects the difficulty of the puzzle, as if there are more pieces available there is a higher chance of their being too much choice for the user, which can potentially cause analysis paralysis (where the user cannot decide what piece to play because there are so many options to choose from) or possibly overwhelm the user due to there being too much information displayed to them at once which increases the risk of cognitive overload. If the pieces were semi-randomised instead, the difficulty may be reduced if pieces of a similar nature were placed next to each other; for example, if there were pieces that were associated to fields – such as 'private int numberOfPotatoes;' and a distractor of 'private double numberOfPotatoes;' – were grouped together this may help the user to focus on a more abstract level reducing the amount of mental strain as they would not need to re-order the pieces as part of their process of deducing how to create a solution. Despite extensive searches, there does not appear to be a usual version of Parson's problems that incorporate the delimiter to be anything different to a new line – for example, there is not usually a cluster of lines in one piece, nor is there one word in one piece – it is always, seemingly, after every new line with very few exceptions.

3.1.1 Parson's Puzzles Tools

Parson and Haden (2006) originally created a drag-and-drop exercise framework called 'Hot Potatoes' which allows you to export custom exercises to HTML web pages that make use of a Javascript library called JMix. This interface, while free and useful, does present its own challenges – it can be difficult to insert a piece between two already existing pieces and would mean you need to shift existing pieces down to create room for another piece – this would be tedious for longer, more complicated, tasks that require more pieces by nature. Additionally, there is only one kind of error message, and it appears to check whether pieces are out of order – for example, if an NP placed a piece too low down but the general order of the piece was correct in terms of logic, it would still raise an error. On a positive note, the interface still works well in modern times, and because there is a manual 'check' button, it will allow for students to only gain immediate feedback if they so desire to – it could be that the pressing of a check-button part way through an exercise could indicate some uncertainty from the learner if they are wishing to double check their solution at that point in time, additionally, it will discourage learners from relying purely on visual, instantaneous feedback that could make them rely on the interface to tell them when they are done. Additionally, distractors are supported in Hot Potatoes as well as, potentially, grouping units based on small extracts of code.

Kaila et al. (2008) created ViLLE which is a Java applet originally designed to aid program visualisation, but 2009 versions and beyond allow for the creation of Parson's Problems due to that kind of Code Puzzle's popularity. It is freeware, for non-commercial use. While distractors are not supported, there are different types of errors that can be generated – so there is a distinction between an error that would cause the code to not run, an error that generates a bug and an error that has logical issues.

While these tools are still downloadable today, the source code for either of these products is not freely available, and while the general premise of Parson's problems is utilised, there are some differing aspects between interfaces that suggest that there is not one standard way of presenting Parson's problems. Like some researchers before us (e.g., Ihantola and Karavirta, 2011) non-accessible code did cause issues initially – which is why the research used paper-based solutions for the pilot study as it was envisioned that from the results of the paper-based product we would be able to create an interface and test it. As suggested by our research and the consequent discovery of the workspace, these findings did come about due to the freedom involved with using paper-based pieces without a restricted virtual environment.

Ihantola and Karavirta (2011) created JSParsons using Javascript widgets that can be embedded into HTML pages and is based primarily on their experience with both Hot Potatoes and ViLLE and from advice and feedback from their students' experiences of using the tools. Their tool is open source under the MIT license and is still available today as one of the only open-source Parson's problem generators currently. JSParsons has two modalities – one for distractors and one for what they call a 'basic' mode with simple sorting of pieces. While this tool has a combination of both Hot Potatoes and ViLLE features, it is primarily focused on portraying Python – a language that Aston University first years were likely unfamiliar with. The reason this is an issue for us to translate to Java is, because JSParsons relies on indentation to check whether a coded extract is correct and does not check for brackets. While it could be said that a separate piece could be made for closing brackets, or, pieces need to append the closing brackets to the end of the line it is not easily translated as it is not essential for Java to be indented. Instead of pop-up boxes for feedback, lines are given colours – green for correct, red for incorrect. If the user has forgotten to indent the line, it highlights this.

3.2.1.2 How effective are Parson's Puzzles at detecting difficulties or issues with NPs?

Many studies have been conducted on investigating the effectiveness of using Parson's problems in a computational way in regards to improving student learning and highlighting issues with student understanding (Parson and Haden, 2006; Helminen et al., 2012; Harms et al., 2016), and there has

been indicative evidence from these studies that it is possible to both classify what NPs are stuck on, and, that the state of their final solution does reflect their understanding. For example, in Ihantola's and Karavirta's (2011) study they found that some struggling students were typically making getting stuck on some portions of the task, such as recursive coding, and repeatedly submitted the same incorrect solution. But is this truly the same as understanding their understanding of, say, recursive coding and how can we check whether the measurements are in any way reliable or accurate?

3.2 Chapter 3 Summary

This chapter introduces the concept that games can be used as an alternative to traditional programming to help NPs learn. From Chapters 1, 2 and 3 we have concluded that a series of questions need exploring in order to successfully bridge the research gap of how we can effectively diagnose an NP's understanding without them being able to explicitly state the precise part of programming they are struggling with. Therefore, we conclude that these queries will form the essence of what this research aims to answer:

- How can we identify NPs' understanding?
    - What is the best way to measure NPs' understandings of programming concepts?
    - What is the best way to measure NPs' choice of computational thinking pattern?
- How do mental representations vary between NPs?
    - Are there finite versions of these mental representations, or are they potentially infinite?
    - Can we quantify understanding?
- Can NPs display symptoms of their understanding?
    - How do programmers explain their thoughts?
    - Do these thoughts match their actions? (I.e., is it possible to apply meaning to the actions without the accompanying words?)
    - Does the style of puzzle affect the way they explain their actions?
- Are NPs aware of their own issues with programming?
    - Do these issues manifest in their actions/words?

Chapter 4: Research Methodology

Previous chapters have highlighted the complexity of diagnosing NPs' understandings of programming concepts and cognitive representations associated to the process of programming. There is a variety of ways an NP may view programming, and in order to extract information that can help us derive their point of view, the research methodology needs to mitigate leading the participant while also not causing a cognitive overload. Extracting programmers' 'true' mental representations would be inherently difficult, as per the ethos of phenomenology, we are only an observer and as such can only detect the 'symptoms' of understanding. However, this thesis argues that teachers can only be observers of the symptoms of understanding, and that this statement reflects the reality of teaching programming, and that an effective tool that accurately diagnoses the level of understanding would be beneficial in order to minimise the communication barrier between tutors and NPs.

It therefore becomes apparent that the choice of research paradigm and consequent approach needs to be analysed carefully, and must consider the following questions in order to address the main research question which is 'Can we discern the level of understanding of NPs through examination of their interactions with Code Puzzles?'

In order to minimise observer bias and to reduce the cognitive load of coding the task from scratch, research was conducted through observation of a think-aloud protocol where participants explained their movements of different styled Code Puzzle pieces (based on 2D Parson's Problems) to the observer. In the pilot study, unstructured observation was performed as it was anticipated that the observer may need to ask queries to the participant about the meaning behind their movements. However, it became apparent structured observations were required to avoid the observer influencing the participant's vocabulary, movements or perceptions of movements. For example, if the observer queried a participant with the phrase "Why are you doing that?" the participant could misconstrue the meaning behind the message to mean 'I am performing an odd action' rather than what the observer intended. Likewise, the participants were noted to show symptoms of authority bias as they were aware that the researcher was a PhD student in the field of CS and therefore did attempt to ask queries about the meaning behind pieces. In the structured observations, the observer was instructed by the script to reply with "It is up to you to decide the meaning, I do not wish to lead you" which separated the observer from the participant as much as possible. As such, each study has its own version of the research methodology where the differences have been highlighted in the relevant chapters.

This chapter will therefore focus on justifying choosing the interpretative research paradigm of phenomenography for this research problem, and consequently choosing a mixed methods approach – consisting of collating and analysing qualitative and quantitative data – and analysing these datapoints using Straussian Grounded Theory from the perspective of phenomenology.

4.1 Philosophy and Choice of Interpretivism

Interpretive research aims to understand social reality. As it is typically sociology-centric, this flexible research paradigm encourages the use of qualitative data to capture and frame the reality of human beings. While this does co-align with the goals of the research, which is to frame cognitions of NPs regarding programming which falls nicely under this definition, it is important to consider more than one research paradigm before proceeding.

Previous researchers using Parson's Problems chose to conduct their research from a positivist perspective, which proposes that observations and reason are the means of understanding human cognition and actively rejects the usefulness of qualitative data as this is classified as inherently subjective. The drawback of this approach is that the as the human element of having participants' explanations of the meaning behind their movements or thoughts on the programming concepts has been removed and replaced with the researchers' opinions of why a participant would be moving a piece in a particular way. During the pilot study, following that approach stopped us categorising and quantifying the movements of participants to arbitrary categories – such as 'swap', 'move', 'remove', 'correct placement', and 'incorrect placement'. Instead, this thesis proposes that when participants' movements did not clearly reveal the intentions behind that movement, we would try to identify those intentions by collecting qualitative data. Therefore, after the pilot study, we diverged from the positivism approach. Likewise, other previous researchers in the field preferred to use a pragmatist paradigm – which is focusing on what can be 'proven' or what works and has the philosophy of arguing that a single researcher cannot possibly learn the perfect, universally applicable truth about reality. Pragmatism, in research design, revolves around the core belief that of finding the answer to 'what will work best in this situation' rather than the generalisability of the solution to other situations in a given research area; for example, this research focuses on NPs in a CS undergraduate degree at the end of their first year studying Java, if a purely pragmatic approach was adopted, even if the solution was not applicable, to say, NPs in a different degree course (i.e. business degree undergraduates who are studying Java on the side) or in a different language (i.e., CS undergraduates studying Python instead of Java) then from the viewpoint of a pragmatic researcher is that the original solution would still be considered successful as it works for CS undergraduate degree-studying Java NPs and was not designed necessarily for other situations. Therefore, pragmatism has

the advantage of not allowing the researcher to be caught up in philosophical debates about whether the outcome of the research is a success, as if – say the diagnostic tool that this thesis proposes – is deemed to be effective for the selected participants and could be replicated in other CS first year undergraduates studying Java, then it would be deemed to be successful even if the tool is niche. That said, this thesis is more interested in whether it is possible to quantify or measure the understanding of NPs by observing the students completing Code Puzzles while trying to get them to explain their intent. As a result, this research took more of the philosophy of interpretivism than pragmatism despite the points for both being relevant. It was rare for previous researchers in this field to use a constructivist approach – which rejects the primary principles of positivism and argues that there is no objective, universal knowledge that can be retrieved from reality and that the researchers' values and disposition affect the knowledge obtained. This is because the whole principle of constructivism argues against the possibility of being able to extract the understanding of an NP, and while it is valid to suggest that it is impossible to replicate the exact cognitive frameworks that an NP has from merely observing their interactions with Code Puzzles, this thesis argues it should still be possible to gain enough evidence to be able to determine their understanding to the degree that a tutor would be able to use such information to guide their approach towards the NP. In that way, constructivism was rejected as a potential approach for this research ideology. Likewise, while a transformative research paradigm could have been selected – which follows the principle that knowledge is a social construction and is formulated through the lived experiences of humans – to properly create a transformative piece of research there would need to be in-depth case studies into individual participants that would breach ethics through being so specific as to identify the participant.

The final research paradigm considered was post-positivism – and it is the case that the philosophy behind post-positivism has influenced the construction of this research methodology and as such is worthy of note alongside the interpretative paradigm. Post-positivism gives equal value to both qualitative and quantitative data. In order to gain a full picture of the participants' understanding of a programming concept and computational thinking, the data needs to be view holistically rather than in pieces. It does follow the philosophy of positivism that observations and reason are the means to human cognition – but post-positivist theorists argue that we can only ever observe imperfectly and probabilistically. As a result, researchers of the post-positivistic paradigm are encouraged to mitigate the influence of their perceptions as much as possible, even if the philosophy suggests their influence can never be ruled out entirely. Post-positivism encourages observers to exert as little influence on participants as possible to capture the closest approximation to the truth possible, i.e., in the context of this research framing an NP's reality.

Interpretivism was chosen over post-positivism after the pilot study as the findings of the pilot study suggested that the research philosophies of past works – such as Helminen et al. (2012) who chose to focus on cursor movements and the number of clicks as metrics as opposed to more qualitative means – were potentially incorrectly chosen if understanding of an NP's perspective was needed, it was consequently decided that an interpretative approach would be taken to explore the realm of Code Puzzle interactions. Interpretivism also has multiple research methodologies and frameworks orientated around the principle of discovering theories – such as Straussian Grounded Theory, Classical Grounded Theory and Constructivist Grounded Theory – which would be suited for entering this novel territory. Interpretivism promotes the idea that a researcher is like a social-actor – someone who observes and explores the participants interacting and experiencing their surroundings. In order to generate symptoms for diagnosis of understanding, the NPs need to be observed from this perspective in order to capture and analyse the data obtained from performing an authentic task. While the experiment chose to use puzzle pieces in order to lessen the cognitive load of participants, the task that they were performing – i.e., constructing a program for a task description which was purposely written in a similar format to what they could experience in a Java examination – was authentic due to the realistic aspect of the task itself. While the philosophy of post-positivism does co-align to the research objectives and end goal, interpretivism matches the goals of the researcher's perspectives more – and that is that interpretivism believes every participant's experience and thought processes are unique and worthy of note. The literature review conducted in the earlier chapters demonstrated that this was true about programmers; that there were so many aspects to programming that it is likely that programmers would have differing levels or degrees of understanding which would mean that the research needed to take the stance that each individual was potentially unique. Likewise, research methodologies that are grounded in interpretivism do not exclude the possibility of having quantitative data alongside qualitative data – as long as the data measures a participant's natural interactions to phenomena and allows for contextualisation of data. This field of philosophy explicitly mentions that context is very crucial and that without it the data can become meaningless as it becomes possible for the data to lose its true meaning if the correct context behind the data is not included. This strongly links to the notion of an NP being an individual, and that the previous researchers who took the positivist approach were using a less than ideal methodology for the context of the research problem.

We conclude that interpretivism is a flexible paradigm that promotes the generation of data around the genuine behaviours and reactions of participants, while also promoting minimal researcher impact that can allow for participants to freely explain their perspectives and clarify their own perceptions of reality. This approach is likely to lead to a large amount of data naturally due to the

promotion of qualitative data collection. That said, this paradigm can be time-consuming, laborious and requires a lot of effort from the researcher in order to encode and interpret the data accurately. Participants are required to be willing to be observed – which may deter participants of more shy dispositions or those who do not feel confident in their ability to program, and therefore the nature of this type of research could naturally favour more confident, proficient NPs who may be able to express themselves and under-represent less-confident NPs. During the recruitment process, efforts were made to tailor advertisements so that there was an emphasis on the benefits of participation – the possibility of participants gaining feedback on their work – in order to entice non-confident participants to volunteer. Additionally, interpretivist research is characteristically difficult to replicate due to participants being naturally different – this can mean the research methodology could be taken and performed with little-to-no changes and still get a vastly different outcome to the one produced in this thesis. While the philosophy behind interpretivist research is that every participant's natural experience is valid, it is a possibility that not every experience imaginable could be covered in the sample collated. Another notable issue with interpretivism is that a researcher can easily perform information bias or intentionality bias if they are not careful – many of the concepts in sociology imply that participants typically know what they are doing and why, however, this might not be the case – some movements or actions could be accidental or meaningless and can provide obstacles in gathering meaning from generalised data sets. This is why context and bracketing in interpretivism research is important in order to reduce these biases, and also why, large data analysis that attempts to, say, identify the types of words spoken by a participant may lose its meaning. Interpretivism naturally doesn't have hypotheses, so it becomes difficult to assess the full benefit of the data collated if the wider context and purpose of the research is also not considered; this is why this thesis presents hypotheses based on the literature review findings and uses them as discussion points based on the outcome space generated.

With these points in mind, the interpretivism approach was chosen for this research.

4.2 The Philosophy of Phenomenology: Husserl's Transcendental Phenomenology

Phenomenology is an interpretivism-based, subjective qualitative research philosophy primarily rooted in the fields of psychology and sociology that has been successfully implemented in some educational contexts across a range of different disciplines. The main purpose behind phenomenology is to discover the 'pure' meaning of phenomena; that is, how others interpret and find meaning in the same phenomenon. Consequently, phenomenologists study and document the conscious experiences of a selected phenomenon. However, the way in which this 'pure' meaning of phenomena is extracted is different depending on the exact branch of phenomenology used; for

example, phenomenologists, such as Edith Stein, argued that researchers can only extract the 'pure' essence of phenomenology through having empathy with those experiencing the phenomenon (Sawicki, 2012), whereas other phenomenologists, such as Georg Hegel, would argue that logic and reason are the attributes that must be utilised to keep the purity of the meaning behind the studied phenomenon (Young, 1996).

Phenomenologists can approach the analysis of how others view phenomena in differing ways depending on their school of philosophy, for example, in the context of this research we want to generate a pure description of a lived experience – as in, we want to document and view the NPs' natural dialogue associated to how they view and interact with the phenomenon of programming. However, we also wish to understand how these NPs experience programming in general – not just for our specific task – and whether these datapoints can be interpreted to a kind of experience relevant to the wider context of programming. So, this research needs to document two types of experiences – the 'lived experience', as in how the NP views programming and views the concepts behind programming, alongside a 'kind of experience', as in how the NP approaches a programming task and constructs a solution. The 'form' of the experience then needs to be analysed, for example in this research, how the experience differs between NPs. From this data this research can suggest how this 'studied' phenomenon may be viewed by NPs.

Husserl introduced the concept of phenomenological reduction (Zahavi, 2003); whereby the researcher needs to acknowledge that 'natural' knowledge is just an appearance that can be used to find the pure meaning behind phenomena. It is therefore considered impossible to extract the raw, pure, meaning from dialogue alone and that the way in which someone – such as an NP – expresses their thoughts and feelings exhibits a way in which we can understand the essence of the way in which they interpret the phenomena (Sawicki, 2012). It is therefore paramount for the researcher to conduct qualitative research and gather dialogue, as the essence behind the philosophy of phenomenology is that statistical analysis is not necessarily enough to be able to gain an insight into the true meanings of phenomena. There are two primary deviations in the way that two prominent branches of phenomenology handle researcher bias – Husserl's branch of phenomenology believes the researcher should place their preconceived notions and bias aside during the course of their research, whereas Heidegger's Hermeneutic phenomenological approach believes that it is impossible to truly separate ourselves from reality and suggests that interpretation should be used as revision of the data analysed – also known as a hermeneutic circle (Sawicki, 2012). For the basis of this research, it was deemed more appropriate to use Husserl's Transcendental Phenomenology in order to be mindful of how the questions for the questionnaire were structured and attempt to remove the researcher's influence as much as humanly possible. This thesis will delve further into the

how the research applies Husserl's Transcendental Phenomenology using sources such as Schutz (1967) for brief explanation.

In more recent times, Schutz (1967) proposed the notion of phenomenology of the social world which helped to define Husserl's concept of phenomenological reduction more clearly; in essence, Schutz argued that every person in the world takes their perception of the world to be 'natural' – as in, they believe that they view the 'real' reality of phenomena rather than consciously believing that the world around them is nothing but a façade and that the meaning behind everything they encounter is not the 'pure' meaning of the phenomena. This is logical, as if someone did believe that everything was not 'real', or, that their perception of reality wasn't the 'real' version of all the phenomena they have ever encountered they would likely suffer because of such an ideology. Therefore, Schutz (1967) proposed that everyone acts as though their own version of the world is largely the 'natural' version of the world, and that for a person to deal with the world they naturally form stereotypical expectations of conditions which result in the person generating formulae for how to handle these conditions. For example, in the context of this research, if a programmer – say, called 'A' – suggested that they were struggling to programmer 'B', a 'natural' reaction may be for B to be sympathetic towards A and supportive or motivating towards them. These social patterns are formulated when conditions are met, and, as a result – both programmers react in, potentially, predictable ways. For example, it would be unusual if A told B that they were struggling, and B told them how happy they were that A was struggling. These predictable formulae associated to patterns of conditions are called 'first-order constructs' in Schutz' concept of phenomenology of the social world, which are defined as general members of the public. However, Schutz (1967) argues that scientists investigating phenomena should go beyond 'first-order constructs' and ensure that a healthy amount of continuous doubt is inserted into the research via the process of bracketing – whereby the researcher must consider their own background, purpose, beliefs, biases, personal interests and philosophical paradigm and practice reflexivity to ensure that researcher and innovations biases are mitigated as much as possible during the conduction of data collection and analysis. Scientists that practice bracketing and show continuous doubt of their research are referred to as 'second-order constructs', which Schutz (1967) argues phenomenologists should be. Therefore, before this research commenced, it was important for bracketing to be performed in the form of dialogue with the supervisors and memoing during the study procedures (see Figure 8).

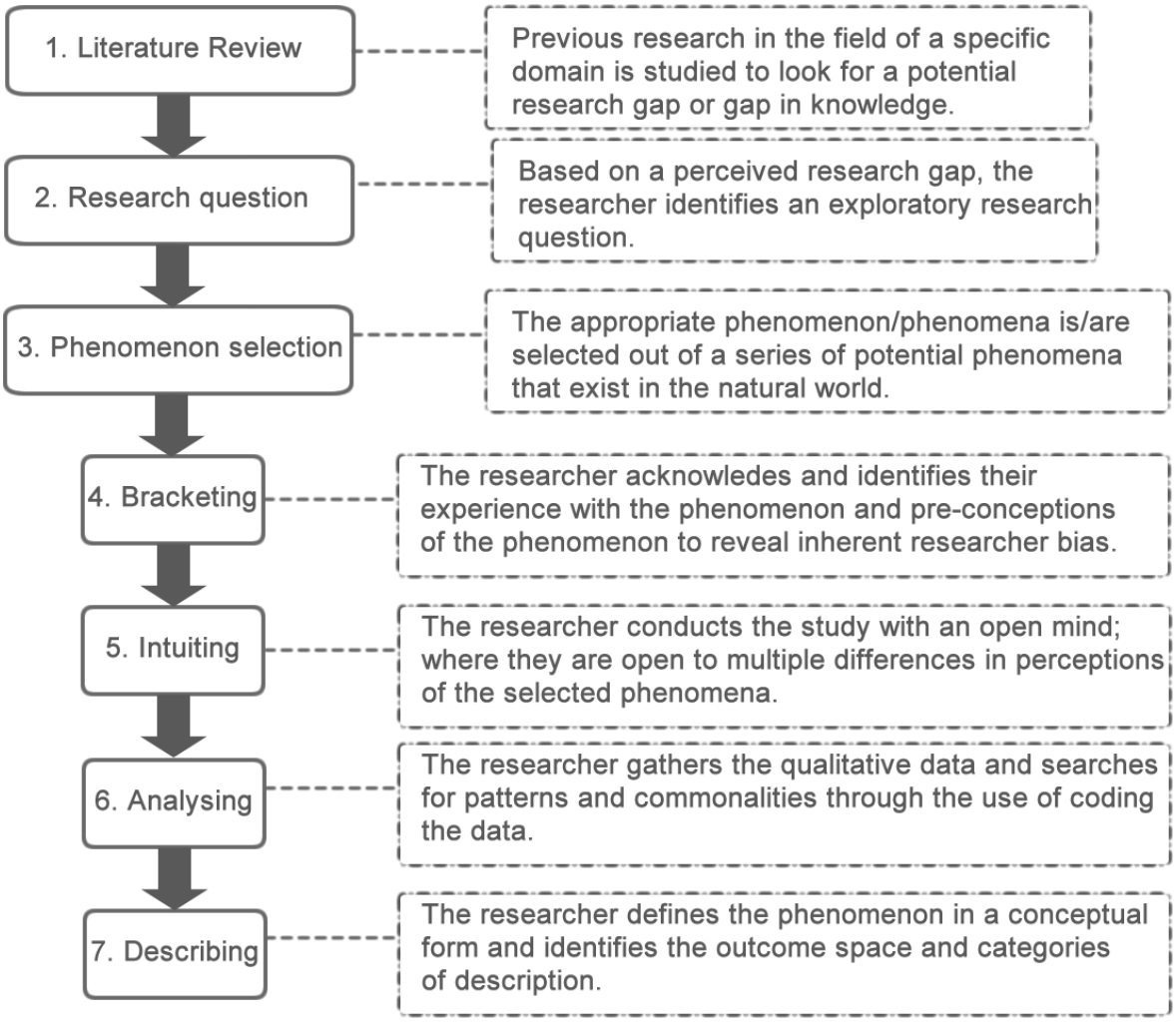


Figure 8: General process of conducting phenomenological research

Phenomenological research can be conducted and analysed through a variety of different qualitative methodologies; however, the principles of phenomenography were created for the implementation of describing the phenomenon in a standardised way and will be defined here. Phenomenography is where a researcher creates categories of description, which when combined, represent the phenomenon as experienced by the observer about the data collated (Marton, 2000). The categories of description can be grouped or related to each other to create an outcome space and can be distinguished by the identified critical aspects of their descriptions. Marton (2000) argues that the outcome space is an alternative way of conceptually portraying the definition of the phenomenon alongside the underlying structure of that phenomenon.

Consequently, phenomenography has been a way to implement phenomenological research and has been successfully used in previous exploratory research in the field of CS education with goals like ours (e.g., Marton and Booth, 1997). Primarily in previous research, phenomenography has been used to identify how programmers of different backgrounds and levels perceive the same

phenomena differently from one another. While the scope of this research differs from this as the research focuses purely on NPs from one UK-based university, we also wish to observe how they understand and view the concepts around programming through their interactions with our planned experiments.

Therefore, the philosophy of phenomenology of Husserl's Transcendental Phenomenology – the idea that phenomenography is based on – was chosen alongside phenomenography.

4.3 Collecting Data Using Observations

Interpretative research has a vast array of different methodologies that recommend differing ways of collecting data. While conducting interviews with participants is frequently advised in interpretative research design – either face-to-face, via electronic applications or through a focus group – it was determined that this would not be ideal for the research question. Previous researchers that used Code Puzzles elected to obtain minimal qualitative data as participants were already interacting with the Code Puzzles themselves and could become distracted, so the thought of attempting to ask the participant questions as well as asking them to complete the puzzle seemed to be ill-advised. Likewise, the concept of the focus group could easily result in obtaining the most confident participant's understanding, or a meshed version of all the group's understanding, which also went against the purpose and motivation behind this research. As such, a formal interview of any kind was not considered beyond the planning stages of the research. In comparison, another frequent recommendation is to perform participant observations – covert, scripted or unscripted – in order to view how a participant naturally interacts with their environment. This seemed the most appropriate way to conduct the research with the purpose in mind – that the research wished to obtain information about the individual's level of understanding rather than the observer's or a group's understanding – this was initially unscripted for the pilot study due to us not knowing whether the observer would need to ask about specific movements during the experiment itself in order to gain a better insight into the perception of phenomena in comparison to a restricted observer – but, this was changed to scripted to try and mitigate the observer's influence as much as possible. Finally, interpretivism design encourages the use of questionnaires and surveys – using open-ended and/or close-ended questions of differing styles – as a way of obtaining written data. As it was anticipated that the researcher would have ample transcribing to perform from the observations, it was determined that using questionnaires would be an effective way to record quantitative data. Participants were therefore asked to rank the difficulty of formulating a solution, as well as their level of confidence as to whether the solution would work or not, on a Likert scale for ease of analysis. The Likert scale was modified to include text-based descriptions in order to try and promote the potential

to compare different participant's experiences to one another based on the participant's recorded reaction. To offer further opportunity to gain qualitative data, open-ended further comment boxes were included after each query.

4.4 Qualitative Data Analysis using Coding and Straussian Grounded Theory

As the thesis proposes an exploratory research question that has aspects of examining the ontology of participants – i.e., capturing the participants' realities in view of a phenomenon – and the epistemology – i.e., understanding the differing ways in which a phenomenon can be interpreted – this research was deemed fit to utilise a form of Grounded Theory as there were no clearly defined expectations for the outcomes of the secondary and tertiary studies after the pilot study revealed unexpected results. This was an ideal situation for Grounded Theory, where the data drives theory creation. However, in order to mitigate potential observer bias, the researcher needed to become aware of, and consequently bracket, their influence when examining the data. There are multiple factors that can influence the way an observer or researcher views their own data which is why bracketing is also important for conducting this type of analysis. For most forms of Grounded Theory, it is encouraged to produce memos – documents that reveal the thoughts and reflections of the observer during or immediately after the experiments have been conducted. During the experiment, the researcher needs to ask themselves about what they are observing, their interactions with their participants and their experience of the process as a whole – for example, what could be improved or what worked well for that participant. During the data analysis phase, it is also important for the transcriber to make notes – such as about the whole data analysis process – what codes have been selected for what parts of the transcripts and their respective meanings, and the identified relationships among codes, categories and themes that are produced as a result of the selected Grounded Theory process. This has the disadvantage of requiring a lot of time and effort, as well as producing a lot of documentation associated to a data set, so, arguably, smaller data sets work better with using Grounded Theory. While unknown at the time of procedure selection precisely how many participants would volunteer for the studies, the sample size selected – 21 participants – was considered fairly large for this form of methodology, especially as manual coding was chosen overusing a Computer-Aided Qualitative Data Analysis Software primarily due to license issues.

In most forms of Grounded Theory, coding the data to make it comparable is an important processing step. Coding is the process where the researcher 'makes sense' of the data collated by classifying extracts of transcripts. For example, if an NP said, "I find constructors hard" this sentence could be coded by classifying it as "difficulty – hard" and "class constructs – constructors". There are many different types of coding, the example illustrates a form of coding known as descriptive coding

– where topics are assigned to aspects of the data. Table 6 documents the relevant types of coding used in this research for the first cycle of coding which takes placed after the transcripts have been generated in order to understand a phenomenon and compare data with each other as participants talk about elements in a unique way.

Code Name	**Where it is used in this research**
Attribute Coding	Background Questionnaire (participants level of knowledge and confidence – secondary and tertiary studies), pre-code puzzle questionnaires and post-code puzzle questionnaires (participant's confidence).
Emotion Coding	Anonymised Transcripts of CP1 and CP2 for each participant.
Descriptive Coding	Background Questionnaire, Pre-Code Puzzle Questionnaire, Post-Code Puzzle Questionnaire, Post-Study Questionnaire, Anonymised Transcripts for CP1 and 2.
Evaluation Coding	Linked with Narrative Coding – the evaluation part of the individual participant summaries uses + and – to indicate what programming concepts participants audibly make incorrect or correct comments about.
In Vivo Coding	During the memos.
Narrative Coding	In individual participant summaries linked to evaluating both CP1 and CP2 transcripts.
Process Coding	In the movement data transcripts and sometimes in the anonymised transcripts for Puzzles 1 and 2.

Table 6: How this research utilised the different types of coding (based on advice given in the works of Elbardan and Kholeif, 2017)

After the first cycle of coding is performed, sorting needs to occur – this is where the generated codes are categorised in order to produce themes. Themes can be established by identifying similar relationships between codes – in other words, can the codes be classified under one larger code? The relationships can be determined by similarity, but also conceptual similarity even if the inherent meaning is different. In our example with difficulty, another NP may have commented "I find constructors easy" which, while the emotional coding behind the phrases is different, there is an underlying theme of difficulty meaning that two participants in our example commented on difficulty. The occurrence of the code is also of interest; if the majority of participants use a code then it may be worthy of a theme by itself. The chronological ordering of codes is also important, if applicable, and in the context of this research process coding was used to help document the movements but was also used with grouping how participants audibly approached solving the puzzle. After codes are grouped into themes, the themes need to be clearly defined and judged for the underlying essence of the theme – for example, are all the codes portraying the same underlying meaning, or different meanings? And from this, the findings are presented in the form of categories or themes with evidence from the data.

However, this description of coding has been generic when the main way that the three forms of Grounded Theories differ is on the aspect of how to approach coding as well as what is the necessary format of data to be in before coding can begin.

All forms of Grounded Theory aim for the same goal – the production of a theory, that is driven from data and are primarily useful for when no adequate theory in relation to the question can be established. Grounded Theories aim to define a social construct or process and uses an inductive approach to garner meaning through qualitative data. However, there are several versions of Grounded Theory, each with their own perceptions of what is necessary in order to reach a theory and with differing levels of detail documented in the differing Grounded Theory approaches.

Straussian Grounded Theory was considered first for this research; originating from the original works of Glaser and Strauss (1967) which documented a full procedure for how to formulate a theory from data obtained. In Straussian Grounded Theory it is argued that the researcher should have limited access to previous research conducted in the area so as not to influence their perceptions of the data – for this research, bracketing was used in order to mitigate these issues as the researcher had read previous works in order to develop a research question for this thesis. That said, the purpose of Grounded Theory appealed regarding data analysis as previous works had failed to identify how understanding could be obtained from NPs' interactions with code puzzles so a way of performing data analysis that allowed the data to lead to the theory was desirable for us to answer the research question. Straussian Grounded Theory is well documented, explaining step-by-step how to perform Grounded Theory, which was attractive to the researcher as they had not previously had background in qualitative research and had been previously unfamiliar with Grounded Theories. Straussian Grounded Theory has the criticism of having unnecessary detail and steps that make that version of the theory tedious; even the original co-author of Grounded Theory stated that "Strauss' method of labelling and then grouping is totally unnecessary, laborious and is a waste of time. Using constant comparison method gets the analyst to the desired conceptual power quickly [and] with ease and joy. Categories emerge upon comparison and properties emerge upon more comparison. That's all there is to it" (Glaser, 1992). Glaser further argues that a variety of methods is possible and that the process of axial coding is only one of multiple comparison methods that would allow the researcher to arrive at an appropriate theory. Glaser (1992) focused on a concept, known as theoretical sensitivity, and suggested that all collated evidence is data and possibly relevant to generating a theory. However, Glaser's (1992) methodology is vaguely outlined as it has a far more flexible approach in comparison, and this is primarily why the researcher chose Straussian Grounded Theory as it felt more structured for a novice researcher. Constructivist Grounded Theory was considered, as it offers a mixture of Straussian and Classical Grounded Theories – it has more

structure than Classical Grounded Theory, and more flexibility than Straussian Grounded Theory. According to Thornberg and Charmaz (2012), Constructivist Grounded Theory supports the idea from sociology that the researcher is a co-constructer of meaning as they are the interpreter of the data and the developer of the theory. The focus of the theory, however, is quite general – it tends to focus on the context, description and complexity of the data and accounts for the possibility of multiple theorems being developed from a single source of data which Straussian Grounded Theory emphasises only one theory should be identified from selective coding. While Constructivist Grounded Theory could have been used, the guidance for implementing it was less clear than Straussian Grounded Theory, therefore, this research chose Straussian Grounded Theory.

4.5 Sampling: Purposeful and Convenience Sampling

Purposeful and convenience sampling was selected for this research, the reason for this is that purposeful sampling is a common sampling for phenomenological research – and as we are attempting to study a specific phenomenon (how NPs portray their understanding of programming) that is experienced by a particular group of people (i.e., NPs who are more specifically enrolled on a first year undergraduate CS course, who have taken and completed at least one iteration of the same core module in Java) - purposeful sampling needed to be used. Convenience sampling, where a participant is selected based on their availability, was enacted due to the restrictions of the ethical procedures not allowing for the study to be integrated into university degree courses or modules. The ethical procedures outline that research needs to be kept separate from modules to ensure that students do not feel coerced to participate in research, and to minimise the possibility of complaints from other cohorts as, say, the tool was proven to be useful to the 2018-2019 cohort but was later removed from the 2019-2020 version of the module due to ethics permissions only being granted for the duration of the PhD, then the 2019 cohort may complain about unfair disadvantage of not having access. Similarly, if the tool was shown to increase the average marks of the 2018-2019 cohort, then it was possible for the 2017-2018 cohort to complain also about having an unfair disadvantage. The reverse would also have been an issue – say the tool had been a distraction for students and resulted in lower-than-average grades when compared to other cohorts, it would run the risk of the researcher getting into trouble for giving unfair advantages. This meant that all volunteers needed to be treated as equally as possible and ensure that advertisement was sent to all students at the same time with specific instructions to say that it was not related to the Java module that the announcement was sent from.

4.6 Recruitment, Conduction and General Procedure of Studies

Before considering the research design, the purpose and anticipated outcomes of each of the three studies needed to be identified; the pilot study explored the feasibility of categorising 'levels' of understanding based on the movements of the learner with Java code segments and aimed to mimic previous studies in the area by classifying the movements using 'swap, add, remove' actions in order to form a baseline reading for future automation. It was anticipated that a series of numerical scores based on the likelihood of the NP having that depth and breadth of understanding about particular programming concepts would be produced in a similar manner to previous research (e.g., Helminen, Ihantola, Karavirta, and Malmi, 2012).

In short, participants were asked to re-arrange two types of paper-based and Java-based Code Puzzles into runnable solutions. CP1 was in the style of a 2D Parson's Problem – with one piece being one line of code – while CP2 was in the style of individual segments of code – with one piece being one word or piece of syntax.

```
1. public class PotatoShop {
2.  private int totalPotatoesRemainingInStore;
3.   private double priceOfPotatoes;
4.   private int numberOfPotatoesSold;
5.   public PotatoShop(double price, int totalPotatoesInStore)   {
6.      priceOfPotatoes = price;
7.      numberOfPotatoesSold = 0;
8.      totalPotatoesRemainingInStore = totalPotatoesInStore;
9.   }
10.   public double sellPotatoes(int numOfPotatoes) {
11.     if(numOfPotatoes <= totalPotatoesRemainingInStore){
12.         numberOfPotatoesSold += numOfPotatoes;
13.         totalPotatoesRemainingInStore -= numOfPotatoes;
14.         return calculateSale(numOfPotatoes);
15.     }
16.    else {
17.       return null;
18.    }
19.  }
20. public double calculateSale(int numOfPotatoesSold){
21.    return priceOfPotatoes*numOfPotatoesSold;
22.  }
23. }
```

Table 7: CP1's Model Answer

```
Numbered pieces are separated by the regex pattern of: [num].[piece]
1.|import| 2.|java| 3.|.| 4.|util| 5.|.| 6.|Date| 4.|;|
5.|public| 6.|class| 7.|Potato| 8.|{|
   9.|private| 10.|double| 11.|weight| 12.|;|
   13.|private| 14.|Date| 15.|expiryDate| 16.|;|
   17.|public| 18.|Potato| 19.|(| 20.|double| 21.|weight| 22.|,| 23.|Date| 24.|expiryDate| 25.|)
|  26.|{|
      27.|this| 28.|.| 29.|weight| 30.|=| 31.|weight| 32.|;|
      33.|this| 34.|.| 35.|expiryDate| 36.|=| 37.|expiryDate| 38.|;|
   39.|}|
   40.|public| 41.|Boolean| 42.|isFresh| 43.|(| 44.|Date| 45.|currentDay| 46.|)|  47|{|
     48.|if| 49.|(| 50.|expiryDate| 51.|.| 52.|after| 53.|(| 54.|currentDay| 55.|)| 56.|)| 57.|{|
58.|return| 59.|true| 60.|;|
      61.|}|
     62.|else| 63.|{|
          64.|return| 65.|false| 66.|;|
     67.|}|
   68.|}|
   69.|public| 70.|double| 71.|getWeight| 72.|(| 73.|)| 74.|{|
    75.|return| 76.|weight| 77.|;|
  78.|}|
79.|}|
```

Table 8: CP2's Model Answer

While this research aimed to mimic previous research, this research focuses on whether Code Puzzles can effectively be used to discern understanding of NPs.

While rare for previous research to use paper-based prototypes, Denny et al. (2008) did do so to investigate using 2D Parson's Problems instead of traditional exam questions to make marking turnover easier for staff members and concluded that they were more efficient than traditional means. Helminen et al. (2012) wanted to use software to see if using 2D Parson's Problems could help NPs self-learn and help lecturers automate feedback; they focus on the quantitative data collected in these studies – such as the quality of code produced, length of time taken to create the solution, mistakes made, cursor mapping and the classifications of individual movements that a user made. These two previous research studies formed the main basis of the procedure for the studies presented in this thesis.

The subsequent studies' methodologies aimed to minimise the potential research bias present in previous research; for example, Helminen et al. (2012) used experts to interpret the reasoning behind cursor movements, whereas this thesis' studies collected qualitative data on what the NPs' explanations for their own movements were. An observer can only ever observe the symptoms of understanding, and while an expert is likely to determine the intended meaning behind movements, it is closer to the philosophy of interpretivism for the researcher to allow the participants the ability to explain their own intentions.

For each participant recruited, the studies followed the same stages of the process – they were through study design, ethical procedures, advertisement, recruitment, getting informed consent from the participant, arranging a mutually available meeting in a secure location, re-clarifying consent by walking through the participant information sheet, setting up the experiment equipment, completing the puzzles and a follow up session whose purpose was to provide individual feedback to the participant as per the advertisement (see Figure 9).

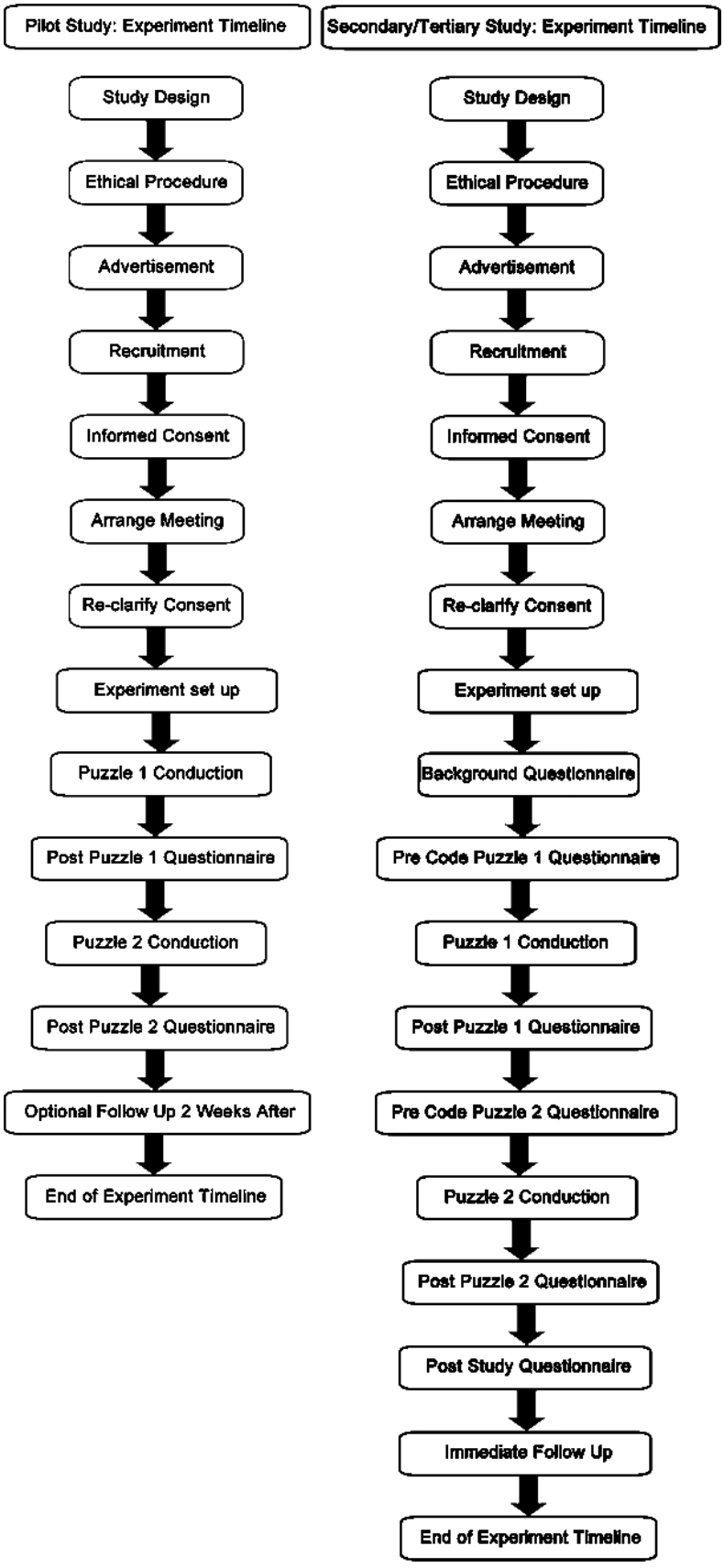


Figure 9: The study timeline from the participants' perspective; a general overview of the difference in procedures between pilot and the secondary and tertiary studies

4.6.1 Advertising and Recruitment Process

The recruitment process involved advertising through posters placed around the main building and via first year CS module announcements (see Appendix). All studies required participants to have completed, or be at the final stages of, a Java foundations module and be enrolled in either a combined honours or honours degree that had a CS component to it. The reasoning behind this was, in part, to ensure that each participant had been exposed to the same material in regards to Java and allow the research to create a baseline for the knowledge used in the puzzle pieces. In the pilot study, no incentive was offered other than the notion of gaining feedback from a follow-up meeting; in the secondary study, a £10 Amazon voucher was offered to participants who attended their scheduled session – regardless of whether their solutions worked or not and regardless of whether they completed all the tasks; in the tertiary study, no incentive was given as due to COVID-19 restrictions it was not safe or efficient to handout or post vouchers. Participants who were interested in participating in the study contacted the researcher directly to ask for more information, to which the researcher replied with a participant consent form – which they needed to sign prior to meeting – and a participant information sheet which explained the research purpose, procedures, time, incentive, appropriate contact details to relevant governing bodies, risks involved and a reminder of participant's rights (including the right to withdraw data from future publishing).

Participants needed to provide *informed* consent, therefore, the observer and participant read through the consent form and participant information sheet prior to the study commencing to ensure the participant fully understood the contents of both documents – this occurred even if the participant pre-signed the consent form prior to attending the study. Participants were reassured that they could withdraw from the study up to six weeks after the experiment took place, but none of the participants withdrew from any of the studies. Participants were also informed that they could ask any questions during the investigation, with the purpose behind this procedure being that participants could reveal their knowledge and understanding through the questions asked to the observer. In all studies, participants were told that they could stop the experiment whenever they wished to – and the observer encouraged them to stop if they felt too frustrated to continue or could not see any more changes to their puzzle that they could make. All participants were required to have engaged in a first-year undergraduate module which focused on the foundations of basic Object-Orientated programming concepts (e.g., objects, fields, methods, inheritance) using Java; this module assumed that the students were new to programming and taught them the required Java needed to create classes.

Participants were informed that they would be recorded by two different mediums – via a microphone, located on the desk connected to a laptop and via a handheld camera. For the tertiary study alone, participants were told that Blackboard Collaborate Ultra was recording both their movements and sound simultaneously. While numerous technical issues were present throughout all three studies, the primary issue was linked to the quality of video recording captured by the handheld camera – in the pilot study, the school's camera provided to researchers would frequently fail, resulting in the researcher using their own camera which was not high definition. This made the transcription of pieces and movements challenging, and audio and visual as well as observer notes were needed in order to create adequate translations of the movement data. All participants were told that an algorithm would be used to analyse the data to produce a representation of their understanding; in the pilot study this was performed after transcription, in the secondary and tertiary studies, the observer acted as a medium for this and in the immediate follow-up feedback sessions produced feedback based on the real-time observed data without pre-processing analysis. However, this recorded feedback was compared to the generated results produced by the same method as the pilot study to see analyse similarities or differences in the information given.

4.6.2 Observation Room Set Up and Procedure

In the pilot and secondary studies, the observation room was set up with two, conjoined desks with the original perception of the participant using the left desk for the final solution, and the right desk for randomised pieces. Participants would need to wait for both the camera and microphone to be set up and were warned of this – in the secondary and tertiary studies the phrase "Alright, begin!" was used to indicate participants could proceed.

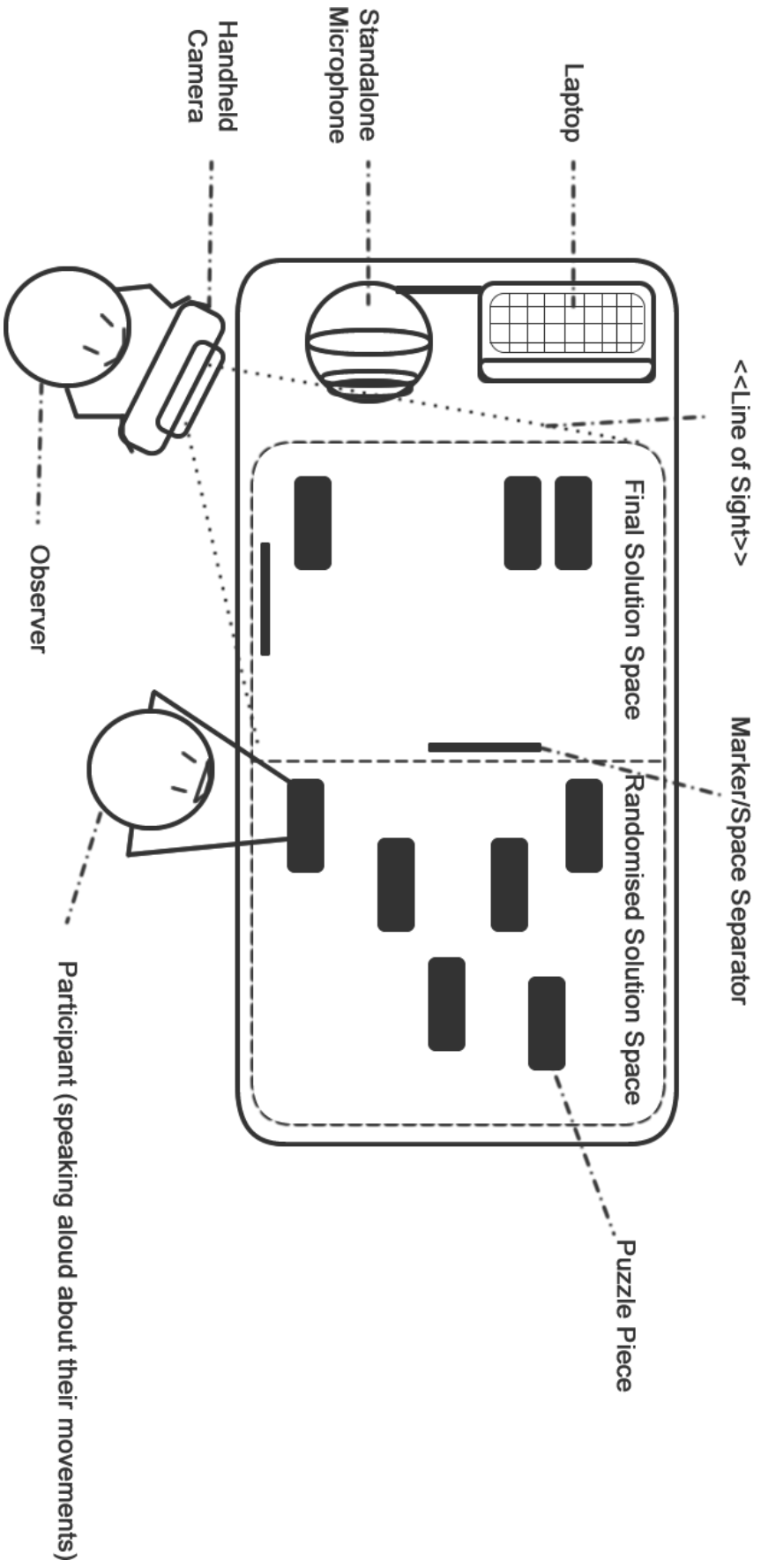


Figure 10: Diagram of the study set up.

The observer in all studies was tasked to ensure, primarily, that the microphone was in working order, followed by the camera being in focus, followed by attempting to listen and note down any observations recorded. This was difficult for the observer to handle; the ideal experiment would have had at least two observers, but this was not possible to do.

Participants were told to speak their thoughts using a think-aloud protocol, which was explained during the participant information sheet phase. While, ideally, participants would be 'trained' to speak aloud, as this is an unusual skill to master, it was deemed more realistic if participants were told more generally what to focus on so that the explanations could reveal what they viewed as important rather than the researcher's expectations guiding their explanation. In the secondary and tertiary study, for participants who fell silent, the phrase "Remember to explain your movements" was vocalised by the observer after either: a) the participant had not spoken a word for over 20 consecutive seconds but had been interacting with pieces or b) the participant was performing an action that had not been previously, or clearly, explained prior to the action. This thesis supports the view that think-aloud protocols do cause heavier cognitive load as participants were found to become quiet and it became difficult to comprehend when they were thinking over when they had forgotten to vocalise their thoughts in the pilot study when no set phrase had been allocated to be used in these situations.

In all studies, CP1 was a traditional Line-By-Line 2D Parson's Problem that involved the same PotatoShop task, and CP2 was a Piece-By-Piece Code Puzzle that used spaces, punctuation and new lines as delimiters for the same Potato task. In the pilot study, participants only needed to complete two post-puzzle questionnaires; in the secondary and tertiary studies they completed background, two pre-puzzle, two post-puzzle and post study questionnaires. The tertiary study had to display the pieces in a static format as Blackboard Collaborate Ultra did not allow for the movement of pieces (see Figure 10, Figure 11, Figure 12, Figure 13 for what each experiment looked like).

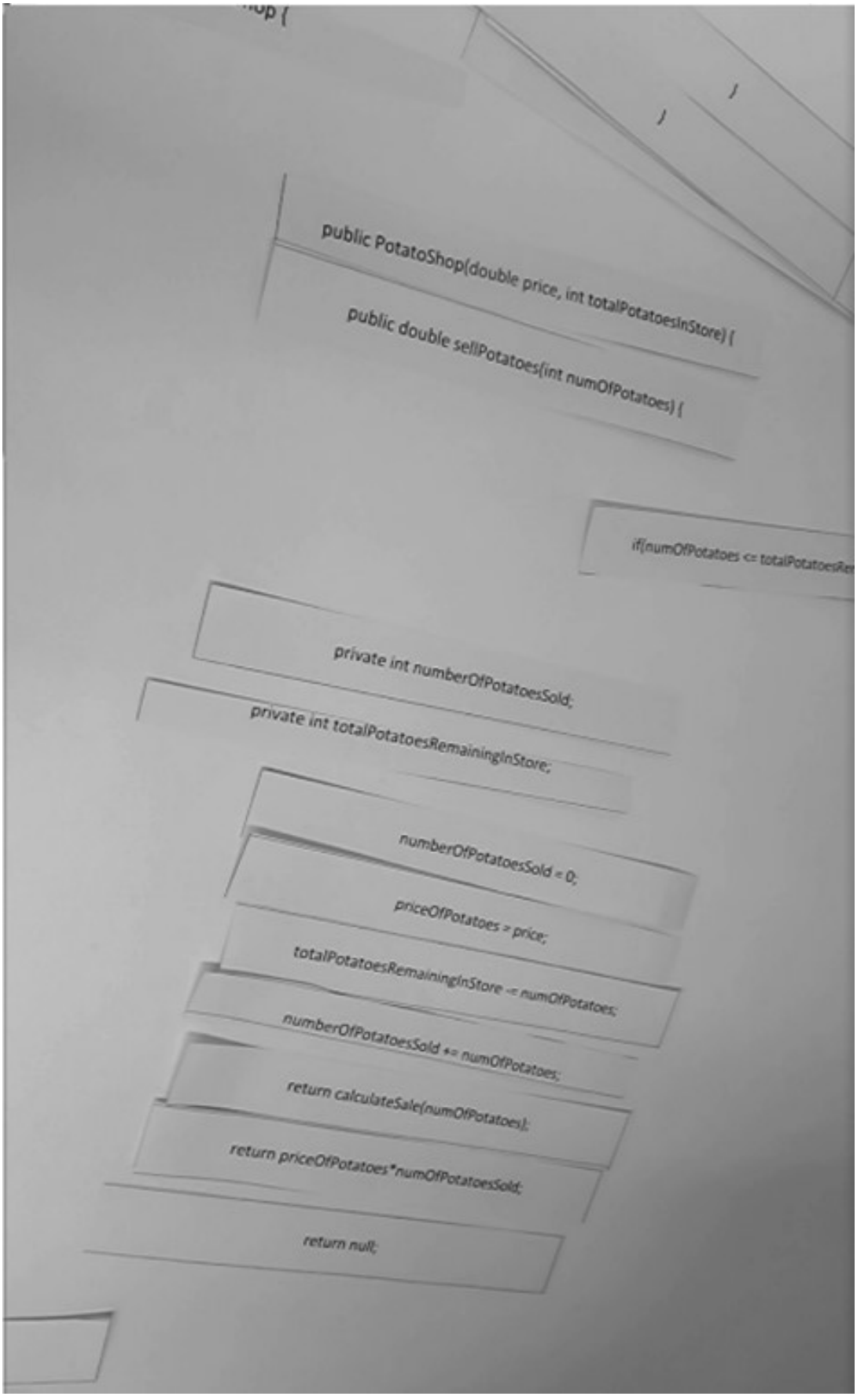


Figure 11: Photo of the CP1's pieces in the pilot study.

Figure 12: Photo of the setup of the secondary study; example of the camera and microphone set up.

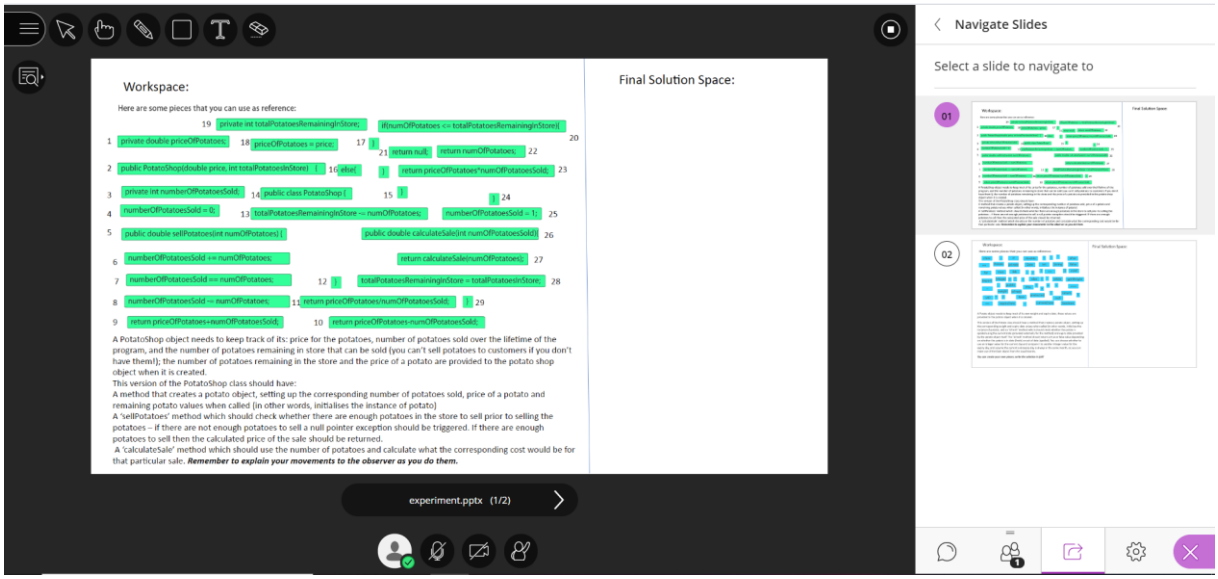


Figure 13: Tertiary Study: How pieces were displayed to participants.

After a participant had completed their puzzle, either by vocally announcing in the pilot and tertiary studies or via showing a red card in the secondary study, they were given time to complete a post-

puzzle questionnaire while the observer set up the next puzzle on a different desk (where possible). After participants had completed their final puzzle, they were thanked for their time in all studies.

4.6.3 Follow-up Procedure

In the pilot study, participants were not given immediate feedback as it was intended that the data would need to be judged retroactively before a conclusion on their understanding could be made. However, in the secondary and tertiary study, an immediate feedback session followed so that participants could remember their movements and give a more informed perspective on whether the observer had analysed their understanding accurately or effectively.

4.7 Pre-Processing Procedures and Consequent Data Analysis

After each experiment, audio and video transcripts were produced. The audio recordings were sound-boosted and cleaned before being manually transcribed into 'raw' time-stamped, voice-to-text transcriptions with transcriber notes (where relevant) and corresponding video identifiers and timestamps (where appropriate). The video recordings were used to transcribe the movements of the participants into anonymised time-stamped movement patterns where a card would be allocated to an 'order number' (i.e., the order in which the participant placed the pieces) and the time it was placed down by the participant. A piece would only be considered 'placed down' when the participant had removed all their fingers from the piece. The 'raw' audio transcripts were cleaned to change words that were deemed sensitive (i.e., may identify the participant) or inappropriate (i.e., vulgar) to '[REDACTED]' to create anonymised audio transcripts.

4.7.1 Qualitative Data Analysis Procedures: Using IPA and Straussian Grounded Theory

The anonymised versions of the transcripts were then coded using Straussian Grounded Theory; open coding was performed to generate a series of codes, which were then grouped via axial coding in order to generate categories of description which provided an outcome space and the formulation of new theorems. The anonymised transcriptions also were examined using interpretative phenomenological analysis (IPA) in order to generate a different form of coding which "aims to capture and explore the meanings that participants assign to their experiences" (Reid et al., 2005). The vocal data was also analysed for signs of frustration present in the participant – e.g., sighing – and other body movements and noises were also documented and studied for potential meanings – e.g., the participant tapping their finger on the desk while thinking, 'hm', 'um', 'uh', 'er', and 'dududu'. Similarly, a ratio of words spoken by the participant verses the observer was also generated, alongside information relating to the types and content of questions that were asked

among participants. The questionnaire data collected from all studies was examined from the same perspective as the audio for the open-ended questions.

4.7.2 Quantitative Data Analysis Procedures: Time Analysis and Movement Classification

The quantitative data collated consisted of numerical counts based on the audio and all the questionnaire questions. Numerical data focused on counting the types of movements performed as it was anticipated, based on previous research, that the participants' movements that could be translated directly to action statements – such as 'swap', 'remove', and 'add' – and that these movements would correlate to a specific formula that would affect the level of understanding of concepts associated to that puzzle piece (see Table 9)

Action Title	Description	Indicator
Remove	A pre-placed piece was taken by participant from the final solution area without replacing it with another piece within 10 seconds of the removal.	Participant is unsure
Add	A piece was moved into the final solution area and did not equate to swap.	Participant is sure
Swap	A pre-placed piece was taken by participant from the final solution area and replaced with another piece within 10 seconds of removal; both pieces would have the 'swap' movement added to them instead of add/remove.	Participant has confused pieces

Table 9: Anticipated Movement Types and Indicator

The movement logs would record the order of the pieces, the time the piece went onto the final solution space and how many movements there were – these would then be analysed for the time taken, number of movements for each piece and how long was spent on each part of the class.

4.8 Protocol Amendments and Ethical Considerations

The pilot study attempted to simulate as much as possible with the equipment available at Aston University to previous studies with the intention of forming a baseline, that said, the major difference was that the software used in previous studies was not available to the researcher, so paper-based cards were used. Similarly, due to the ethical procedures present at Aston University, it was not permitted to run this alongside the Java Foundations module as a mandatory component as this would breach the ethical protections for students – therefore, they had to volunteer and be ensured of their rights prior to participating in the study. Participation bias and participant selection biases are likely present in the sampling as the sample may not represent the entire cohort – participants were noted in the secondary and tertiary studies to be "fairly confident", on average, with their coding capabilities meaning that less confident programmers were not involved in the research. That said, due to the restrictions, it was not possibly to mitigate this other than in the

careful construction and wording of the advertisements. However, the outcome showed participants performing different movements and it was determined that what they explained did not always correlate to the anticipated reasoning behind the type of movement performed and revealed a form of anchoring bias presented in previous studies due to the restrictions of interface design being loosened due to the natural nature of a paper-based study.

As the secondary study yielded similar results to the pilot study but had not yet achieved data saturation, a tertiary study was performed with similar sentiments. But, due to the restrictions of COVID-19, a paper-based study was no longer feasible and therefore the procedure was moved onto a virtual learning environment – Blackboard Collaborate Ultra. That said, Blackboard Collaborate Ultra did suit the style of experiment due to the restrictions of the interface design not allowing for pieces to be moved manually, and consequently there was a poor uptake of participants.

Originally, three tasks were envisioned to be performed in an hour session – however, the pilot study revealed that this was not possible unless the participant was unusually quick at completing the puzzles. As part of our pilot study, three different types of paper-based 2D Parson's problems were created – varied by splitting the solution code using one of the following templates: splitting the code after each line (known as Line-By-Line), splitting the code after each space (known as Piece-By-Piece) and after each space but also including alternative pieces, some of which are incorrect (known as Red Herring Piece-By-Piece). In the pilot study itself, students were spending almost an hour on the first two puzzles and therefore the third puzzle – with the red-herrings – was omitted. However, NPs were determining how to complete classes using the remaining pieces of the Code Puzzle. As noted in feedback by participants in the pilot study, such as "read through the spec, it defined all I needed" and "the instructions explained in a lot of detail that some parts of it a could have been assumed", it was decided that the instructions did need to be modified in a way that participants who explicitly followed them instead of using the pieces as a guide would be highlighted with more contrast than they were in the pilot study. Likewise, participants were noted to be using the pieces remaining as a guide for what to do next, and therefore a red herring that could be comparable to the pilot study without adding complexity to the solution was therefore used in the secondary and tertiary studies alongside a modification of the observer explaining to the participants that "not all of the pieces need to be used and user piece creation can be done if there are pieces you feel are missing and need are not present". Therefore, in the secondary and tertiary studies, participants were given red herrings in both puzzles and told that they can create their own pieces. Participants in all studies were informed that they would need to construct, to the best of their ability, a working Java class using the paper-based Java code puzzle pieces available.

Therefore, only two tasks were given to all participants as no one managed to complete the third task in the pilot study, so it was combined with the second in the secondary study – the first task was to arrange paper-based 2D Parson's Code Puzzle pieces into a working version of a 'PotatoShop' Java class, and the second task was to arrange paper-based individual segments (where the delimiters were: a space, new line or end of a variable name) in order to construct a working version of a 'Potato' Java class. The reasoning for choosing Potato shop and Potato for the two tasks was to establish a concept of the program they were attempting to build, and for them to link and draw upon the two classes. Likewise, participants were given two separate task descriptions that purposely omitted technical terms, such as 'constructor', 'accessor method', 'parameters' and 'fields', in order to try and not influence the participant. Aside from this exception, the task description was written in a way that was similar to their Java Foundations module examination question style – this was due to the research wishing to establish a baseline by not surprising participants with a new style of query as we intended to draw upon their understanding which inherently relates to their previous experiences of programming. That said, this surprisingly caused issues as participants were used to seeing the technical terms used and some were quoted to be deciphering what the meaning of the task descriptions were, likewise, multiple participants claimed they were not familiar with java.util.Date which they had explicitly learned in their Java Foundation module. Furthermore, it was discovered that most participants tended to struggle with reading extracts from official Oracle documentation despite this also being covered in the module. Therefore, despite the research design's intention of creating tasks that would be in line with what participants experienced, this was not the case for all participants – this meant that we could analyse the effectiveness of extracting information from both NPs who were familiar with parts of the task, and, for NPs who were struggling with parts of the task. While all participants were seen to complete Task 1 – even if their solutions were flawed – one participant was noted to 'quit' for Task 2 and knowingly hand in an incomplete solution for Task 2; this participant is key for analysis, as before they submitted their solution there is ample text transcribed for that excerpt that helped to reveal the issues that caused frustration with alleged 100% accuracy for that participant.

As noted previously, this research incorporated an interpretivism approach to research design – and therefore a think-aloud protocol was adopted in order to make participants voice their thoughts and feelings about their movements to the observer. While the think-aloud protocol is difficult to perform, particularly while performing a task, it was deemed important for this study to be distinctly different from previous research in the area by purposefully obtaining the present thoughts and feelings of the participant rather than, purely, the observer's observations. This was to reduce: choice-supportive, outcome and hindsight biases as the participants were commenting on their lived

experiences during the present time of performing their action; courtesy bias as the participants were likely already focused on attempting to complete the task and due to the nature of the think-aloud protocol, more likely to have a relatively high cognitive load so would not be as likely to try to alter their words during the experiment to please the researcher; intentionality bias as participants were audibly communicating to the observer what their intentions were about an action or piece; self-serving bias as participants would be naturally commenting on both successes and failures during the course of the experiment and researcher bias as the observer was instructed to ensure that they remain as unintrusive as possible.

That said, there was the provable presence of shared information bias present in the transcripts as a result of the think-aloud approach – as participants often commented on their actions for both tasks as reading the piece and saying that they were going to place it in a specific position. That said, this thesis argues that this is also a necessary part of understanding how participants view phenomena and how they rank the importance of the experience of constructing coding – it may seem obvious, but the participant commenting on the placement of a piece in a given scenario indicates to the observer that they were concentrating on the process and positioning of the piece, and, due to the nature of paper-based studies it is important to understand the intended location and purpose of the piece with many participants demonstrating natural indentation of the pieces. While not always the case, participants often commented on placement and movements before explaining their actions or revealing their perception of the concept behind the piece. Therefore, in this instance, this thesis argues that such a bias being inherently present is necessary for the observer to gain an insight into a participant's understanding and reasoning.

In the pilot study, participants were asked to attend a 40-minute session – however, for the secondary and tertiary studies this was increased to a 1-hour session. On average, participants took around 10 minutes (for the pilot and secondary studies) and 45 minutes (for the tertiary study) to complete Task 1; and around 15 minutes (for the pilot and secondary studies) and 33 minutes (for the tertiary study) to complete Task 2. For this reason, this thesis will group 'pilot and secondary studies' and 'tertiary study' in separate sections when discussing the data holistically as the pilot and secondary studies were paper-based and performed as intended by the research design, but the environment used for the tertiary study was less than ideal and did not work well for the intended purpose meaning that the data for the final three participants is only comparable in terms of the solutions produced and the commentary rather than for the timings.

For the pilot and secondary studies, it was specified that an office space needed to be booked far enough away from CS staff so as not to accidentally breach the confidentiality clause for the

participant. That said, as audible on the recordings and during the transcripts, the available rooms were less than ideal, and disruptions were frequent during experimentation which were noted to break the participant's immersion and train of thought. These distractions ranged from small – where it was simply corridor noise or the next classroom being audible – to moderate – where the provided camera failed to work as intended – to severe – where the participant and observer is forcibly removed from the room during experimentation due to a misunderstanding about booking rights. As a result, the research methodology incorporated, as part of the analysis, a distraction penalty to the time it took to complete a given solution depending on the noticeable length, effect and severity of the distraction on the participant's words. For the tertiary study, this was not an issue, however, Blackboard Collaborate Ultra did cause similar distractions – including causing the loss of a participant's entire solution due to the rubber icon being mistaken for a rubber rather than a 'wipe all screen' function. Similarly, moving or dragging pieces in the environment was cumbersome and, instead, text typing was used which took longer. Distractions weren't recorded in the same way for the tertiary study, but a note was made for the affected participant who screen wiped their solution before completing it.

While the pilot study anticipated it would be easy for the observer to answer queries and understand when the participant had finished, this was not found to be the case. Therefore, the secondary study made use of a red card (a submission button. or for when the participant wants to stop the experiment) and yellow card (a question button for when the participant wants to directly ask a query to the observer) – this was to avoid the observer imposing more so on the experiment itself and to give a clear indication of when the participant was happy with their solution as asking them if they were content with their final product could lead them to doubt their solution. Despite 'testing' phase of program development not being anticipated to be revealed, many participants displayed a need to check their code prior to submission and this yellow-red card dynamic helped to flag what participants needed, albeit some naturally forgot about the cards. The observer was also given a stricter observation script as it was found that in the pilot study participants were asking themselves rhetorical queries during the think-aloud protocol. While the observer script did encapsulate a lot of scenarios, participants still found differing ways of forgetting previously discussed information and this is evidenced when the observer goes off script during the observations. In the tertiary study, the script needed to be altered and participants were primarily asking queries associated to the operation of Blackboard Collaborate Ultra itself which detracted from the task at hand. Tertiary study participants were encouraged to type in the chat, but few did and most naturally chose to vocalise their query.

Finally, questionnaires differed between studies – in the pilot study, it was intended that enough information would be garnered from the movements alone that more than two questionnaires – the same questionnaire asking about the difficulty and perceived confidence in the solution working after Task 1 and Task 2 – would be sufficient. However, the pilot study revealed that this was less than ideal as the movements themselves were not the greatest indicator of the participant's understanding. Equally, assuming that all participants had the same baseline due to them sharing the same module and degree programme for one academic year was not enough to establish a true baseline – as a result, the secondary and tertiary studies incorporated a background questionnaire which was completed prior to attempting or seeing the puzzles where the participant would clarify: their level of confidence in programming in Java; what languages they were familiar with (in order to obtain their experienced programming paradigms); what they found difficult; what they found easy; and how they believed they approached programming and a documentation of their processes. This was performed also to reduce the social desirability bias as the pilot study received, off-recording, unusually positive feedback about the accuracy of the findings and this thesis wished to establish whether these participants were viewing their results in the same way as one could view a horoscope. Having a documented and established way of them approaching an authentic task meant that their word alone could be supported by a questionnaire conducted prior to the study itself, and, to see whether the NPs did have an approach to these tasks that could be equated to their perceived approach to programming tasks on a computer. Likewise, the perceptions of the task itself were not easily identified during the think aloud protocol of the pilot study, and therefore a two-question pre-puzzle questionnaire was completed based on the perceived difficulty of the task itself to separate the difficulty of the puzzle from the difficulty of the task.

Additionally, a final questionnaire was completed at the end of the follow-up feedback recording for the secondary and tertiary studies – unlike the pilot study – these follow-ups were done immediately after the second task's post puzzle questionnaire was completed and was not an optional add-on that happened a fortnight after the event occurred. The reasoning for this change was that participants' uptake on the non-recorded feedback sessions that occurred a fortnight afterwards was poor as the time commitment of meeting for a separate session after a first was deemed a lot of effort. From the participant who did take up the follow-up meeting, they had forgotten about their movements due to the gap and therefore could not properly assess their opinion on the feedback presented by the researcher. While the immediate follow-up sessions meant that the observer needed to insert themselves into the equation and eyeball the data, they had collated to assess whether they could identify the understanding of the NP, this was deemed to be appropriate given the drawbacks of the follow-up meetings conducted in the pilot study. Furthermore, the

methodology was altered so that these feedback sessions were recorded as a major issue with the pilot study was that these feedback sessions were not allowed to be recorded according to ethics, whereas, on the secondary application of ethics this was universally granted presuming that the participant wished for immediate, but possibly imperfect feedback – which all participants agreed to. This thesis also argues that the observer using the data that they had noticed in the session was also a realistic metric in its own right – for example if this diagnostic tool was used it would be unlikely that the observer would have noticed all of nuances of the participant's symptoms of their lived experiences in real time. This also meant that the observer's responses were analysed in comparison to the answers given in the post-study questionnaires as the post-study questionnaires were likely influenced by the feedback session.

There were two ethical applications to the EAS ethical board, and one amendment of a previous application related to this research. The pilot study was conducted in 2017-2018, the secondary study conducted in the April-May 2018, and the tertiary study was conducted in July-September 2020. Ethics was originally granted in 2017 for a study that researched whether a decision tree algorithm could be used to determine the level of understanding of an NP based on interactions with Code Puzzles. However, it became apparent from the unexpected results that the purpose changed, alongside the aforementioned protocols, and therefore a new ethical application was created and approved in early 2018. Due to issues with ill health prolonging the research, a change of Ethics Board members and COVID-19, the 2018 was further amended to accommodate for the current issues.

4.9 Bracketing (for Pilot, Secondary and Tertiary Studies)

Bracketing is a crucial stage of Husserl's Transcendental Phenomenology and is the first step towards analysing the epistemology of the study procedure. For example, it is important for the researcher to identify their own thoughts and feelings towards programming, how they construct programs, how they view the phenomenon of a program and what their expectations are of the study. It is also important to determine exactly what subjectivity is in the context of these studies and consequently analyse the subjectivity of the researcher. To clarify, the researcher, the observer, the transcriber and the data analyst for all three studies was the same person.

In order to achieve bracketing, the researcher needed to practice reflexivity – a form of reflection which analyses how the researcher views the world of research and perceives the examined phenomena – which in the case of this research, is how NPs interact with code puzzles and whether this reflects on their understanding. There are multiple parts to the phenomena, so the researcher

needed to identify each individual part that comprised the network of phenomena being observed, as only through separation of the components can true reflexivity be achieved. The researcher deduced that it was necessary to perform reflexivity on: their perceptions of a program; their pre-conceptions of how to create a program; their perceptions of the underlying concepts of the program; their perceptions of what each piece translated to; their perceptions on what data would be collected from the phenomena which formed the basis of hypotheses as many of these perceptions were based on reading about previous research; how difficult the tasks would be; and on how participants would audibly describe their interactions.

Bracketing was practiced before each study commenced, and during each study memos for each of the participants were attached to the anonymised transcripts in order to capture the perceptions of the observer at the time of data transcription. While, ideally, memos should be creating during live observation the researcher needed to hold the handheld camera and had their mind busy with the process and being 'present'. The observer then transcribed their thoughts after the meeting with each participant to capture their reactions.

Bracketing requires the researcher to perform necessary forms of reflexivity. There are three main types of reflexivity: personal (reflexivity as a confession, Van Maanen, 1988) – about the researcher's own background, interests and motivations in the research; methodological – about the researcher's planned procedures and the research philosophy; and theoretical – about the theories or thoughts they had on potential outcomes of the research. Each of which was categorised in the memos associated to each participant post study.

Van Maanen (1988) showed that there was a difference between confessional tales and realist tales; realist tales identified the objective interpretation of the natural world; however, confessional tales were identified as the researcher discussing about how they perceive the world and documenting their lived experiences of the world and typically involve details such as their background overall as a researcher and their perspectives on why the topic they are researching is important to them personally. The reason that confessional tales are important is because they demonstrate how the researcher could influence their researcher by their own unconscious bias and allow for the researcher to check whether there is innovation bias present in the way they have collected or analysed the data. It can also be used to determine where the researcher's strengths and weaknesses are, and from this, determine what parts of the research procedures are likely to change depending on their experience level. Van Maanen (1988) argued that confessional tales served two primary purposes – downgrading authority or upgrading authority. Authority is the concept of how the researcher perceives themselves and how that perception can influence the way they conduct,

analyse and/or interpret the output of their research; to 'downgrade' authority the researcher needs to place their research in a more critical, negative light to see the limitations of their findings and 'upgrade' authority has the researcher place their research in a more positive light choosing to focus on what the data contributes to their knowledge or what they have learned from downgrading their authority. For example, a researcher downgrading their authority may say "as a non-expert in the field of phenomenology, this influenced the way I approached x" whereas a researcher with an upgrading authority may say "for someone with little background in phenomenology, this was the only perceivable way x could be accomplished". However, this thesis argues that it isn't surprising that researchers tend to claim authority more so than downgrade their authority; that said, it is a useful warning that a lived experience can be portrayed in different ways and that the tone of the researcher during the personal form of reflexivity influences the way they could interpret that data or objectively assess the procedures implemented (see Table 10).

Study	(Summarised) Personal Reflexivity	Value of Personal Reflexivity
All	Researcher acknowledges that there is a personal investment in producing something that is useful, as, they came from a background where they studied programming at undergraduate level without any previous exposure to programming and know of the personal difficulties encountered by NPs. As part of their teaching, seeing how NPs were deterred from programming made them feel as if the research mattered and that they wanted to find a way to potentially help struggling NPs communicate more openly with their lecturers.	The researcher needs to use that passion to keep determined during the course of the research, however, they need to step back and not project their investment or expectations based on their own past experiences onto participants actions. Therefore, the participants need to be focus of the data and the more structured Straussian Grounded Theory is the way to analyse the dataset.
Pilot	Researcher had limited experience of the interpretivism philosophy and conducting a study as they had come from a background with little previous experience. Researcher felt anxious about recruiting enough participants with no given incentive, and also felt anxious about whether they would be able to conduct a neutral observation even with using dialogue from supervisory meetings to try and frame and mitigate researcher bias. The researcher was anxious about being able to convert the quantitative data into meaningful analysis as they had limited experience with that form of research as well. Researcher was also nervous about influencing participants and how to conduct proper unstructured interviews as they possess social anxiety.	Researcher needs to improve their capabilities as a researcher by ensuring they have read enough literature on how to conduct observations and avoid leading questions. The researcher ensured collection of quantitative data and qualitative data to allow them to be able to change research philosophies should there be an issue with the quantitative data alone. The qualitative datapoints are also useful to cross-check whether the data indicates similar inferring of understanding.
Secondary	Researcher felt slightly more confident after conducting the pilot study and using their	Researcher needs to keep their positivity and ensure that they

	previous reflections on the methodology and theory to generate structured procedures. Researcher felt they still did not fully comprehend the reality of the philosophy of phenomenology, and the more they read, the less certain they felt about their understanding. Researcher was hopeful and enthusiastic, especially with the prospect of potentially gaining more participants and seeing whether their theory of the previous study was supported.	interact with participants as little as humanly possible. The reflexive methodologies amalgamated from the memo data allowed for the procedures to be altered to accommodate the change from post-positivism to interpretivism philosophies.
Tertiary	Researcher felt roughly the same as pilot study in terms of confidence as confidence had dwindled due to needing to transfer the procedures to Blackboard Collaborate Ultra because of the pandemic. Likewise, the researcher was affected by their illness and felt as if their passion to do research wasn't as good as previous studies. Researcher felt mentally low; their perceptions of how well the experiment would work were poor but did not have more time to go through ethical procedures again to change them to something more suited or have enough time to build and deploy an online Code Puzzle software piece.	Researcher needs to find value in all data collated; this is an opportunity to grow as a researcher and view the data from a different perspective. Just because it was collated in a different way does not mean that is not useful, in fact, its use will be to compare the puzzle data to ordinary typing of code to see which one generates more useful dialogue for identifying understanding.

Table 10: Summarised Personal Reflexivity: what potential researcher biases could be present? These datapoints originate from short, written pieces before the beginning of each study.

Methodological reflexivity is primarily used a reminder of the focus of the research and any aspects associated to it – such as hypotheses, procedures and general philosophy – alongside any notable abductions and interpretations of individual participants. For example, perhaps a participant has deviated from the norm, and this is worthy of note in methodological reflexivity. The way methodological reflexivity is recorded is through memos, or methodological field notes that take place during the investigation itself. The researcher asks themselves what procedures are working well, for example, or notes anomalies and what those anomalies suggest about the effectiveness of the procedures, and whether the research question benefits from the selected research approach and philosophy. During these studies, memos were assigned to each participant's puzzle attempt which, on reflection, led to alterations of suggestions for future work. Methodological reflexivity is important as transparency as a researcher is key, and that the researcher needs to be accountable for explaining their actions and reactions to various methodological procedures. Table 11 summarises the methodology reflexivity that took place across the three studies – the participants the memos were associated to have been removed from the table (see Table 11).

Study/Studies	Methodology Reflexivity	Methodology Adjustment
Pilot	Movements are difficult to categorise; the difference between swap and add is very difficult to discern with paper-based Code Puzzles. It is difficult to tell when a piece has been picked up and put down – does sliding count? What about touching but never moving? It is difficult to tell, in terms of co-ordinates, where the piece is being placed or whether that data even affects the ability to discern understanding.	Movement data for future studies will be purely the order pieces were placed in rather than time stamps which became problematic as it was difficult to keep track of due to the differences in movement. The chronological ordering of pieces and number of times a piece moved was deemed useful data, so this affects how movement datapoints are collated.
Pilot	As an observer the researcher is consciously worried about their impact on the NPs that they are observing. What if the observer's wording of questions influences the participants' responses?	While not entirely avoidable, a structured observation is more suitable to try design neutral phrases that have as little impact as possible on the observed.
Pilot	The researcher feels as if the research philosophy is incorrect for what they are trying to discover, it seems as if past research used post-positivism but they feel interpretivism and qualitative data yields more accurate results as the movement themselves without context seems meaningless.	Then the researcher will use the pilot study result as a guidance for changing the research philosophy – this might even be why previous research has failed to identify the understanding of NPs as they are treating them as a statistic rather than as an individual human being. Interpretivism seems more appropriate and what future studies will use instead.
Pilot	The researcher is having trouble distinguishing rhetorical and aimed questions when participants are using the think aloud protocol.	Then the researcher will create a procedure for asking questions – raise a yellow card if the participant wants to talk to the researcher, otherwise, the researcher does not need to respond.
Pilot	The researcher is having trouble calculating the exact time stamp that the participant has finished their solution; most participants tested their code in their head before submitting. When are they 'finished'?	Then the researcher will create a procedure for submitting the solution – raise a red card if the participant has completed the solution.
Pilot	The researcher is finding that participants are spending over the anticipated 40 minutes for a session.	Then the researcher will amend the protocol to factor in the more realistic time frame.
Pilot	The researcher is cautious that participants are just using the pieces remaining at the end to 'fill in the blanks'	While this is a realistic method that participants may use to complete the puzzle, we could incorporate red herrings.
Pilot	The researcher found participants interacting with puzzles in an unexpected way; grouping pieces together based on perceived similarity.	The researcher should repeat the experiment without drawing focus to the workspace; if it is a natural way that some participants interact with

	There is not enough time for a dedicated study on the workspace exclusively, what should the researcher do?	the phenomenon and reveals a different perspective on the phenomenon, we can evaluate whether this is worthy of note.
Secondary	The new philosophy and procedures worked well, although movement data was still an issue to record.	Then the researcher will just record the order of pieces put down and if any pieces are held for longer than 4 seconds, the time stamp.
Secondary	The minority used a workspace; does this mean it isn't valid?	Some still did group pieces, especially in the secondary puzzle. It is still a valid approach, and the researcher has obtained more data on the type of discussion generated which consistently differs from the usual discussion of the expected interactions with the puzzle.
Tertiary	This environment isn't suitable as the pieces are unmovable and static. Participants are having difficulty operating it, typing on it and indenting like they normally would do easily in the pilot and secondary studies.	Then in future work the environment should just remove the puzzle pieces and only display the task description.
Tertiary	Movement data is impossible. Participants are also naturally muting their microphones when typing and not speaking as clearly as in pilot and secondary studies.	This was the best that could be done during a pandemic; the researcher cannot record movement data so let us just view what participants type and choose to say to the observer and see if that is as effective as the previous studies.

Table 11: Summarised Methodological Reflexivity: what potential changes are needed in order to improve the methodology? These datapoints originate from short pieces made from the memos of each participant observation.

The final form of reflexivity covered in this research was theoretical reflexivity; this is where a researcher needs to reflect upon their own assumptions and prejudices about interpreting the data so that all assumptions are accounted for so that the data analysis process is consistently applied with acknowledgements of potential limitations of the way the data has been analysed. Table 12 produced for the three studies associated to the assumptions made by the researcher during analysis. As these assumptions were consistent through the three studies, the studies column has been omitted (see Table 12).

Theoretical Reflexivity	Affected Process/ Processes
The researcher has assumed that when a piece is touched that the participant was indicating that their focus was on that piece rather than the participant intending to move that piece.	During the experiment, data analysis, theory construction.
The researcher has assumed that when a participant asks the observer about the meaning behind a piece that the participant's understanding of at least one of the programming concepts associated to the piece is minimal.	During the experiment, data analysis, theory construction.
The researcher has assumed that participants who do not vocalise their reasoning for their movements after two prompts to do so are not naturally inclined to think their reasoning is worthy of note.	During the experiment, data collection.
The researcher has assumed that a participant who picks up a piece for longer than or equal to 5 seconds is having some difficulty interpreting the meaning behind a piece.	During the experiment, data analysis, theory construction.
The researcher has assumed that participants had their own motivations for attending the experiments and that the majority wanted to gain feedback in order to improve their learning and/or help with their revision as their examinations were drawing nearer at the time of each experiment. The researcher has also assumed this has no real effect on the participants' behaviours.	Follow up meeting; tone of the observer and the amount of time talking to the participant after the experiment was over was due to this assumption.
The researcher has assumed that unless the participant asks a specific question associated to the task, that any issues the participants have are not due to the task itself but more of their interpretation of what the task means or what they think is relevant to solving the task.	During the experiment, data analysis, theory construction

Table 12: Theoretical Reflexivity – the summarised assumptions: what has the researcher assumed to be true when analysing the data or conducting the study? This is based on transcriber notes of the anonymised transcripts.

4.10 Chapter 4 Summary

This chapter provides justification for the chosen research philosophy, approach, techniques, methodology, data analysis, coding, general protocol, protocol changes and ethics processes that the three studies went through. It also provides the bracketing that was completed for each study and discusses the importance of using bracketing. Due to these protocols, a total of 21 participants (5 in the pilot study, 13 in the secondary study, and 3 in the tertiary study) volunteered near the end of their second semester.

Chapter 5. Pilot Study: Identifying Understanding

This study assessed the feasibility of using novice programmer (NP) interactions with Code Puzzles to categorise their understanding of programming in Java.

This study aimed to describe and classify the observable characteristics of NP understanding through their interactions with Code Puzzles and determine whether classified movements of Code Puzzle pieces could be directly correlated to an NP's intention. This study collected quantitative data relating to time and movement of Code Puzzles (e.g., what Code Puzzle piece was moved where) and qualitative data relating to the NP's reasoning for their movement of the piece and their confidence in the produced solution.

While the study's design aimed to mimic previous works – e.g., that of Ihantola and Karavirta (2011) and Helminen et al. (2012) – to form a baseline reading for future work, there were differences. For example, no 2D Parson's Problems software was available to the investigator, nor were the study's materials integrated as part of a module for the undergraduate degree on the grounds of ethical constraints. The study's data is primary data, i.e., collected by the investigator, and obtained from the movements and dialogue used in video recordings of NPs interacting with two types of paper-based Code Puzzles alongside two questionnaires taken after each Code Puzzle had been completed.

5.1 Hypotheses

ID	Hypothesis
H-1	NPs of a similar level of understanding will share similar characteristics in their interactions with a particular code puzzle.
H-2	NP interactions can be classified and categorised
H-3	Classified NP interactions can be mapped to a level of understanding
H-4	NPs will make moves that correlate to swap, remove, and add with no other possibilities of movement.
H-5	There are a finite number of ways in which a code puzzle can be experienced and understood.
H-6	The reasoning behind a classification of NP interactions will be the same among all participants of that category of interaction.

Table 13: Pilot Study Hypotheses

5.2 Methodology

5 CS undergraduate students at the end of their first year at Aston University volunteered to take part in the pilot study in 2017, with the criteria that they must have completed a Java Foundations module to participate in the study to establish a baseline ability among participants.

5.2.1 Advertising and Recruitment Process

Students were recruited through a Blackboard announcement from a shared second term module that all CS and combined honours CS students had access to, and posters were displayed around the university's main building in the summer term. No financial incentives were offered; however, the study was promoted as a revision opportunity to students.

5.2.2 Procedure and Data Collection

The pilot study consisted of 40-to-60-minute observation sessions where participants were asked to create working Java class in two, separate paper-based Code Puzzles in the presence of an observer. Participants were asked to complete two Code Puzzles – the first Code Puzzle (CP1) presented 23 pieces in the style of a paper-based 2D Parson's Problem (where each piece correlated to one line of code) and related to creating a Potato Shop Java class, whereas the second Code Puzzle (CP2) presented 78 pieces in the style of a difficult 2D Parson's Problem (where each piece correlated to one word or piece of punctuation relating to the code) and related to creating a Potato Java class (see Appendix, sections 11.1.1, 11.1.2, and 11.1.3).

CP1 was always completed before CP2, as randomising the order was not possible with the sample size. For both puzzles, participants were asked to create a "working" class using the puzzle pieces that fit the requirements of the task description while adopting a think-aloud protocol. After each Code Puzzle, participants completed a post-puzzle questionnaire which asked them to rate their confidence on how well the solution would work on a 5-point Likert scale and rate how difficult they found the puzzle on a 7-point Likert scale. P1, P2 and P3 experienced feedback after each puzzle, but P4 and P5 only experienced feedback at the end; this was determined to be a change of protocol after the observer could influence the way in which P1, P2 and P3 answered post-puzzle questionnaires.

For P3, CP1's video recording was not included due to technical corruption however the audio file remains intact, therefore, the following datapoints are used in the analysis: 4 CP1 and 5 CP2 movement datasets; 5 CP1 and 5 CP2 think-aloud datasets, and 5 CP1 and 5 CP2 post-puzzle questionnaire answers.

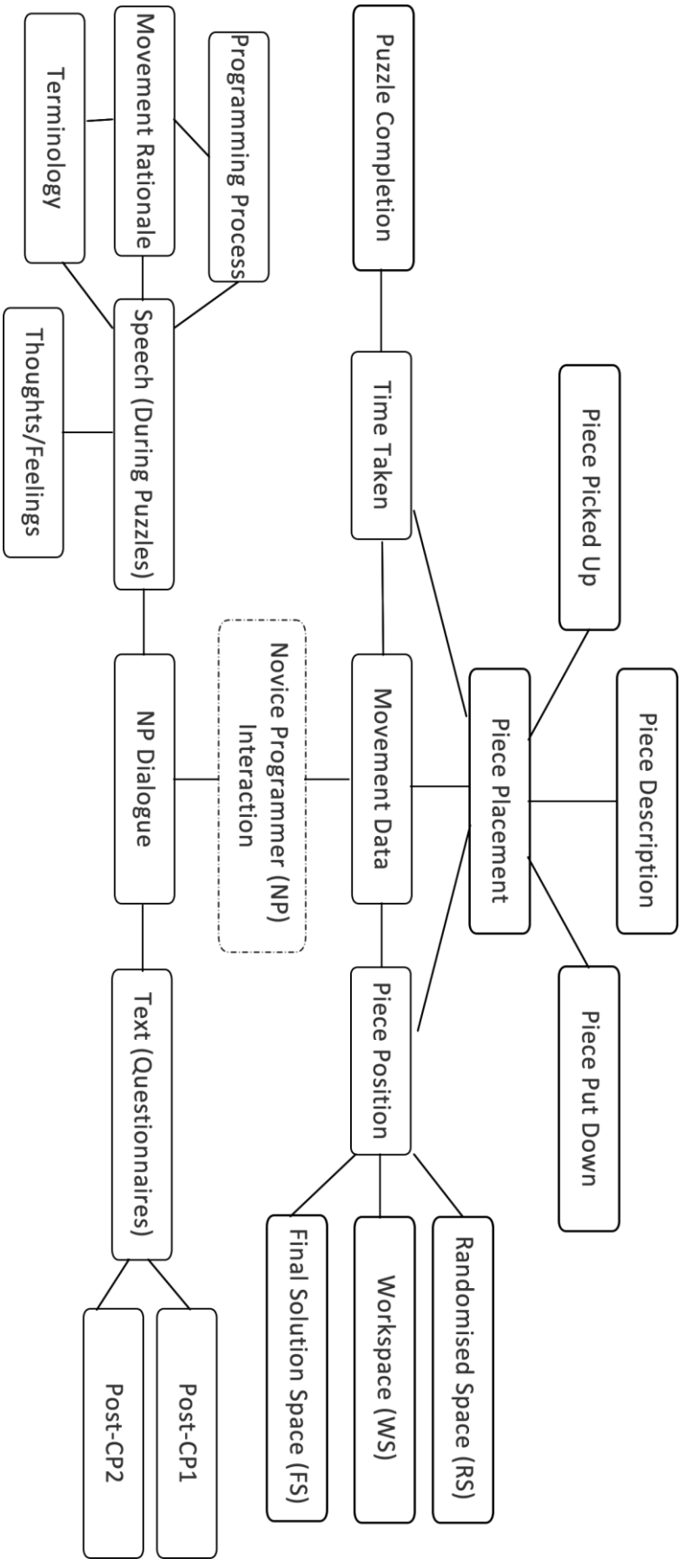


Figure 14: Overview of the type of Pilot Study data collected – referred to as Novice Programmer (NP) interactions.

5.2.3 Data Analysis

The pilot study chose to analyse the: types of movements, number of movements, number of mistakes, types of movements, order of movements and the post-puzzle questionnaires to gauge how difficult participants found each puzzle.

Figure 15: Overview of the way movement data was analysed, and what questions the analysis needed to address.

There was a methodological issue in how to calculate the time taken between moves to further narrow down the pieces that the participants were spending the most time on. Unlike cursor paths used in previous works, the hand movements were tricky – sometimes participants slid the pieces across the desk to place them, others picked them up and held them before placing them. It was therefore determined that the time taken between pieces – known as 'time intervals' – would be calculated from the moment the piece had started to move towards the final solution space rather than when the participant placed their hand on a piece.

$$Ti = Td - Tu$$

Equation 1: The Time Interval calculation used to calculate how long a participant was spending on a single piece. Key: *Tu* = Time piece is picked up, *Td* = Time piece is placed on the table, *Ti* = time interval.

The types of movements needed to be classified in order to clarify what types of movements were present – in contrast to previous works, it became apparent that the flexibility of using paper-based pieces meant that additional movements began to arise (see Table 14).

Type of Movement	**Description**	**Seen in previous research?**
Add	Where the participant picks up a piece from the RS or WS and places the piece in the FS.	Yes
Swap	Where the participant picks up two pieces that they already placed in the FS and switches their positions – this is why the number of swaps occurred should be divided by two as both pieces in the movement data have 'swap' label and the swap itself involves two pieces.	Yes
Remove	Where the participant picks up a piece that they already placed in the FS and places it back in the RS or WS.	Yes
Decide	Where the participant picks up a piece from RS and places it in the WS to focus on where to put the piece in the FS.	No
Back	Where the participant picks up a piece from RS, hovers for at least two seconds, before placing it back in the RS.	No
Group	Where the participant picks up a piece from RS and places it next to another piece in the WS to organise before placing them in the FS.	No
Unclassifiable	Where the participant performs actions that cannot be analysed or are unclear movements.	Yes

Table 14: Classification of movements seen in the pilot study

The classification of 'unclassifiable' was generally assigned to closing brackets – as, unless the brackets or punctuation are given context, it is unclear what participants truly intend the piece to do or what they think of the piece.

The technical difficulty was also how to create a movement log that was readable by a third-party researcher as the research data needed to be sanitised to fit the requirements of the ethical agreement forged between researcher and participants. After several designs, it was determined that there needed to be a grid placed on the participant's recordings of the final solution space that

allowed the researcher reading the file to understand the placement (see Figure 16). This grid was not visible to the participants during the study itself, and the grid was used purely as part of the data analysis after the study had concluded; participants were presented with a blank space to place their final solution on.

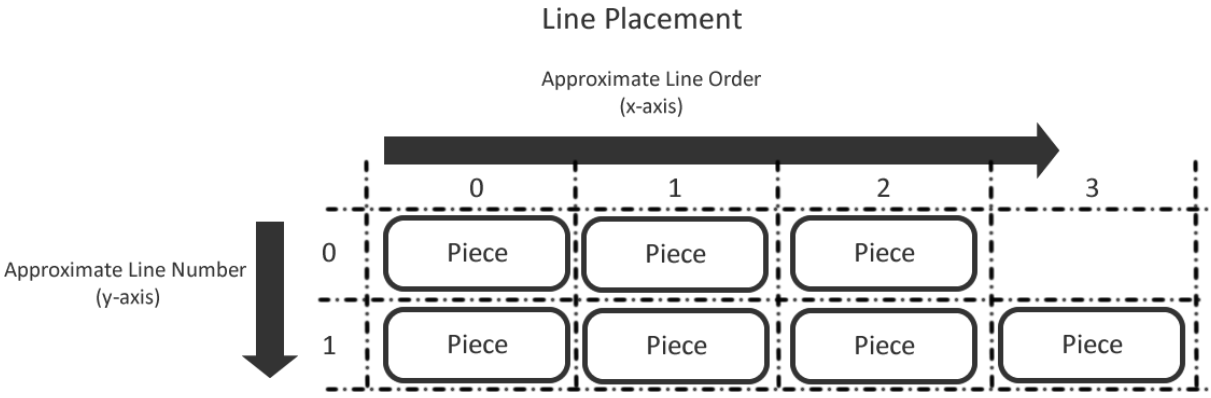


Figure 16: How line placement was determined in the movement logs.

Another technical difficulty encountered that was not highlighted in previous studies was the fact that participants often did overlap or place pieces ambiguously before rectifying them (see Figure 17).

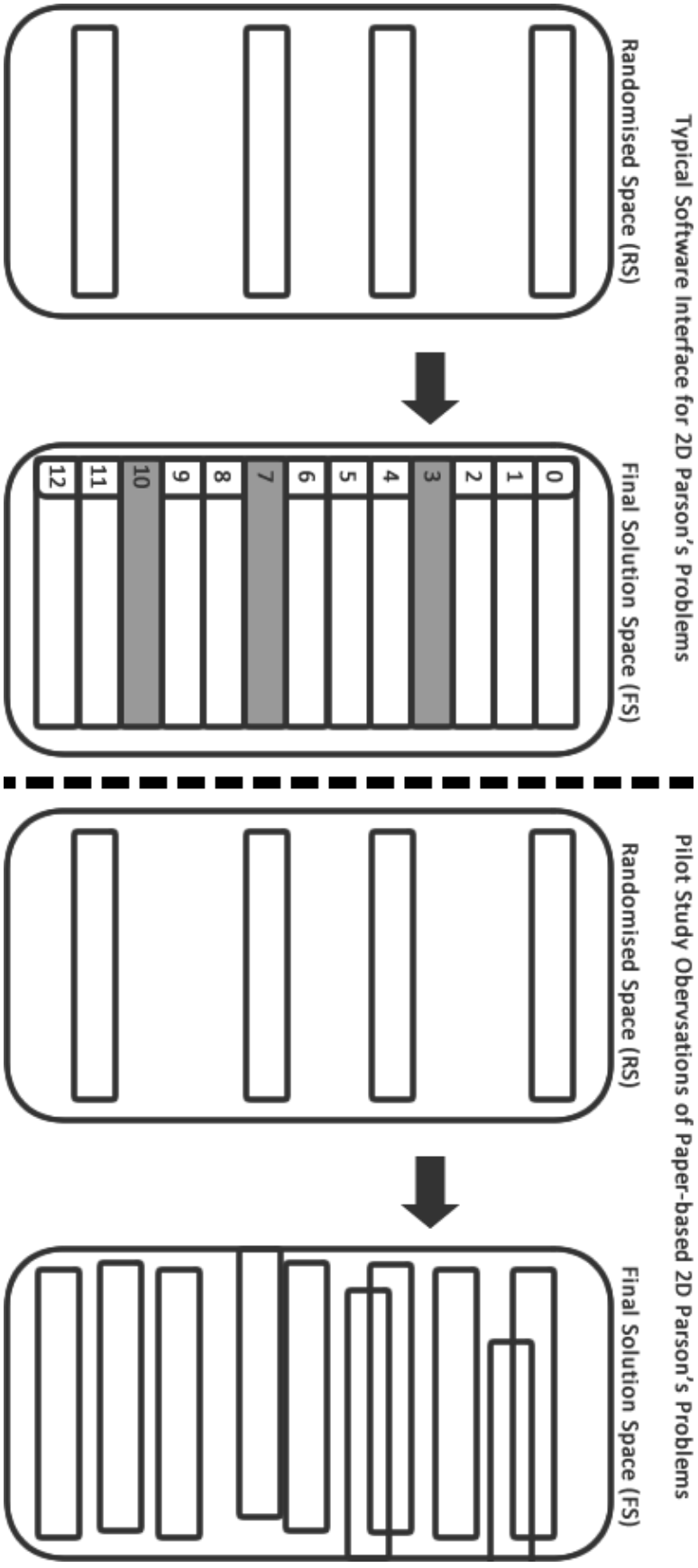


Figure 17: Diagram of an observed paper-based Code Puzzle experiment issue regarding the judgement of 'correct' or 'incorrect' piece placement and explanation for 'approximate line order'.

The placement of the pieces was important to establish the order of the class and the class' structure for further analysis, so it was determined that the intended positioning of the pieces was the final state of that piece for that movement rather than the shifting movements between the pieces where the participants hands were in the way in the recording. This structured process meant that pieces that were classified as 'missing' were pieces that were expected to be in that position when a full method was closed off for CP1 (see Figure 18), and a full line of code was completed for CP2 (see Figure 19).

Figure 18: How missing pieces for CP1 were determined during the analysis of movement logs.

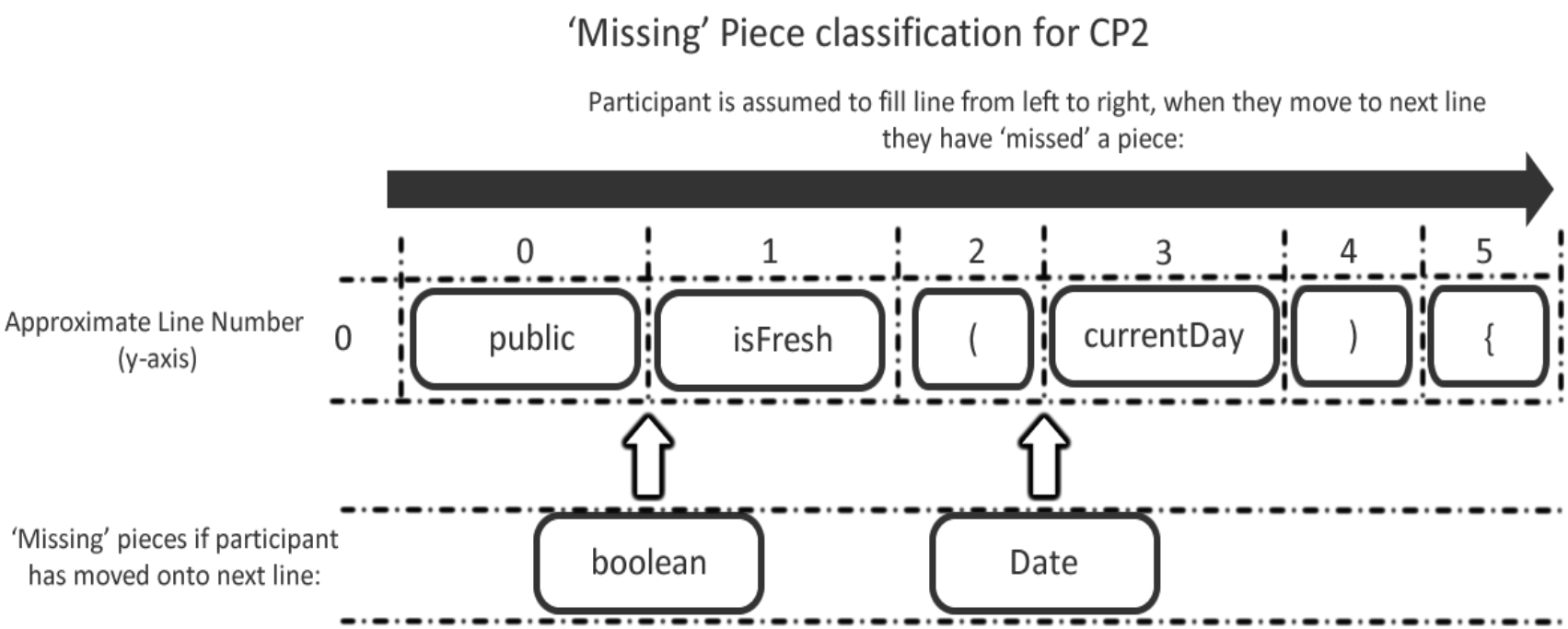


Figure 19: How missing pieces for CP2 were determined during the analysis of movement logs.

Other mistakes were if a piece was placed in an area of code where the code would run into bugs, or, for CP2, the piece was in a place that meant the class would not compile.

5.3 Results

5.3.1 Time Observations

60% of participants were undisturbed by interruptions, however one participant experienced 1 minute and 57 seconds of interruptions (P2) which likely further contributed to the frustration documented in their tone during CP2. Despite this, they produced a flawless solution for both tasks. All participants exceeded the expected time frame for the pilot study sessions, and therefore the sessions were expanded to 60 minutes in future studies.

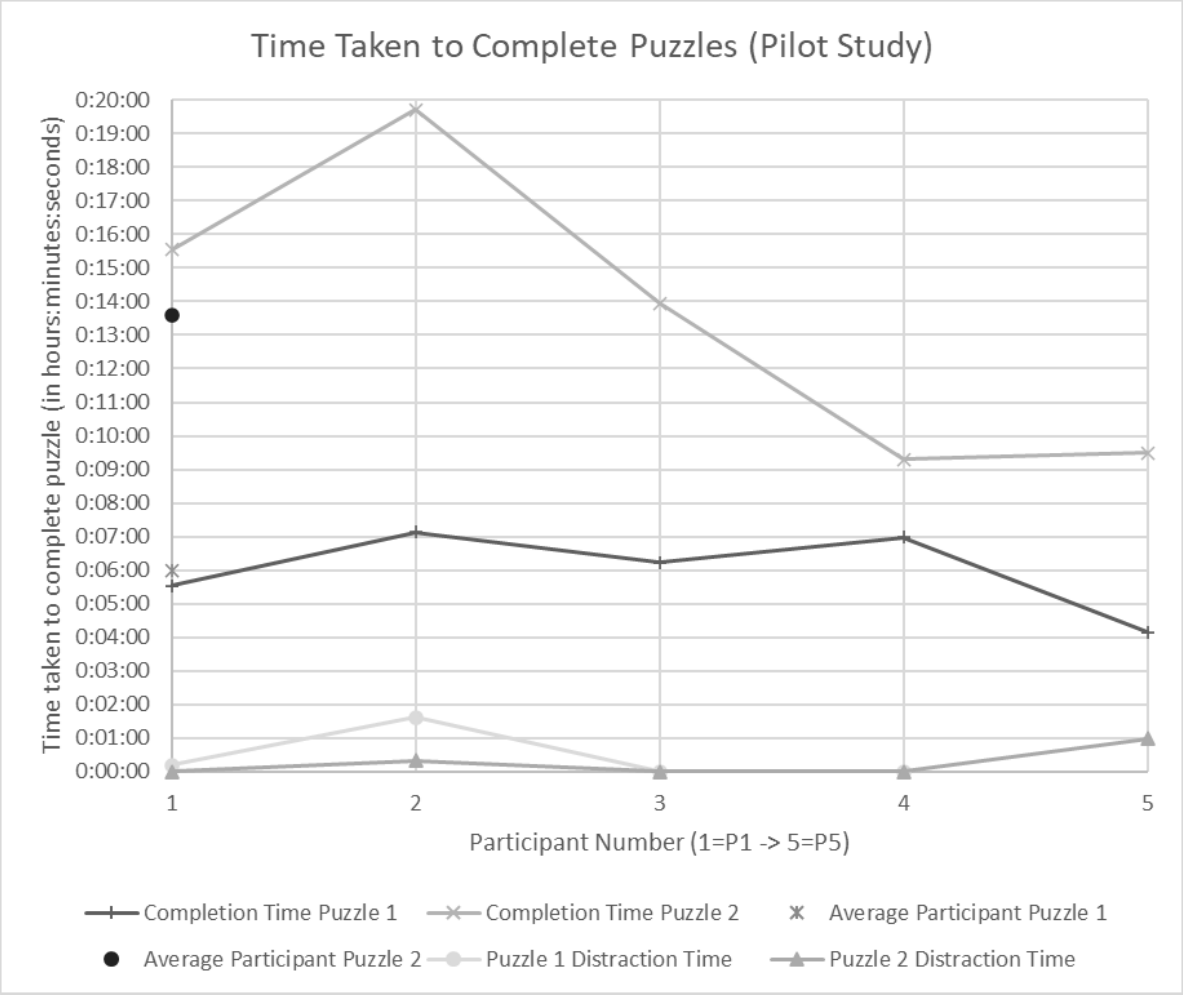


Figure 20: Line chart for time taken to complete the puzzle by each participant (CP1: range = 04:09-07:08, M = 06:00, SD = 01:13|| CP2: range=09:18-19:43, M = 13:36, SD = 04:23). Distraction time was labelled as moderate or severe interruptions (CP1: range=0:00-1:37, M = 0:27, SD = 0:42|| CP2: range=0:00-0:59, M = 0:16, SD = 0:26).

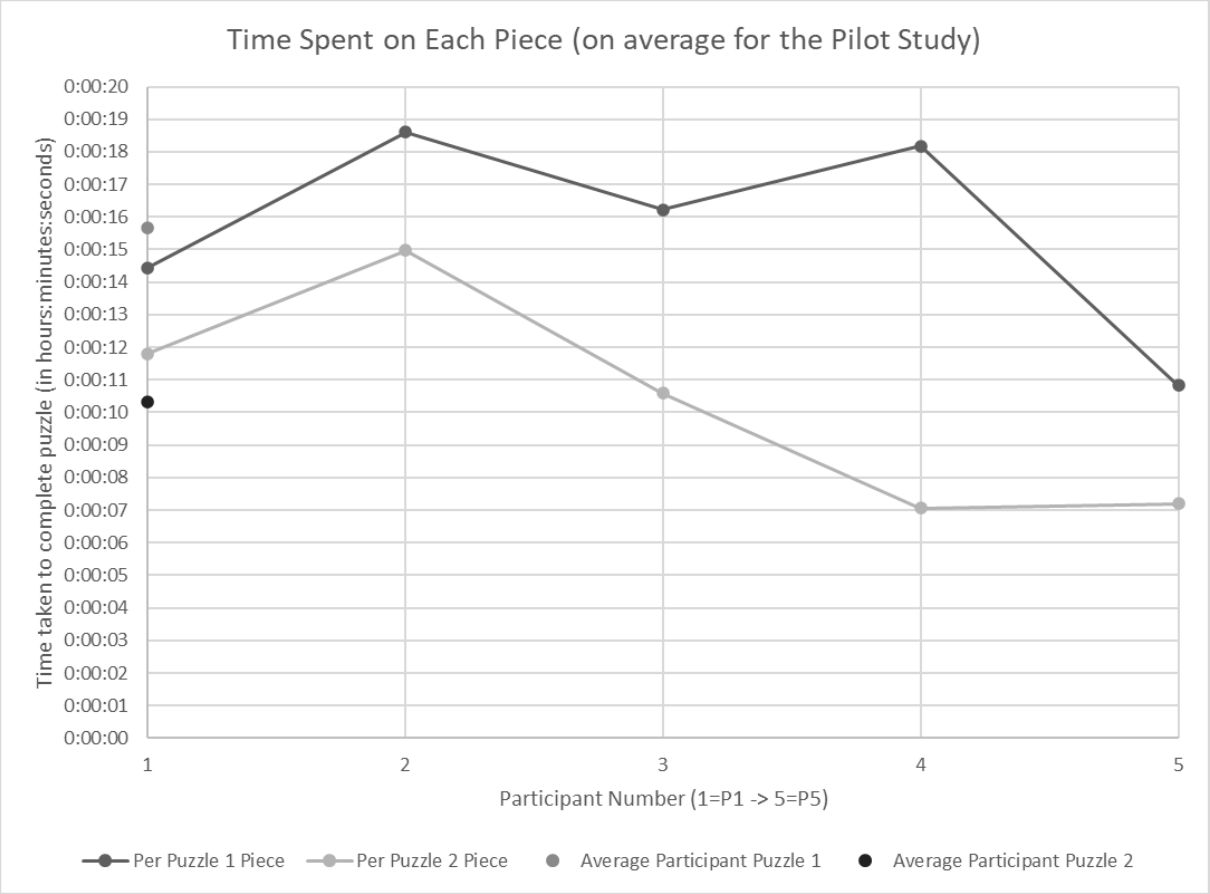


Figure 21: Scatter graph for the average time spent on each piece per puzzle (bottom chart) (CP1: range = 00:11-00:19, M = 00:16, SD = 00:03|| CP2: range=00:07-00:15, M = 00:10, SD = 00:03) for the pilot study.

All participants completed CP1 quicker than CP2, however, the time spent on each piece (on average) is lower for CP2 than CP1.

5.3.1.1 Code Puzzle 1 Time Intervals

Time intervals were calculated by Equation 1, and indicated the time spent between puzzle piece placements.

P1 demonstrated a strange grouping mechanism that later was classified as a workspace even in CP1 where few pieces are displayed. The participant spent the first half of their movement log grouping these pieces, before placing them in the correct order on the final solution. When asked to explain their movements, they suggested that they needed to determine the reasoning behind why the piece exists and to group it with similar pieces so that they can interpret the pieces more easily than if they were randomised. The dialogue changed subtly with the way the participant grouped, rather than the usual process-orientated movements seen by other participants the explanations of the grouping mechanism shed a light onto how the participant diagnoses the meaning behind a piece and how

they relate that meaning to other pieces that they perceived as similar. While this was useful to determine their understanding, the use of 2D Parson's problems meant that the participant's construction of the piece in the context of the task could be identified. Therefore, both aspects seemed useful – although the workspace was a completely unexpected outcome of this research (see Figure 22 and Figure 23).

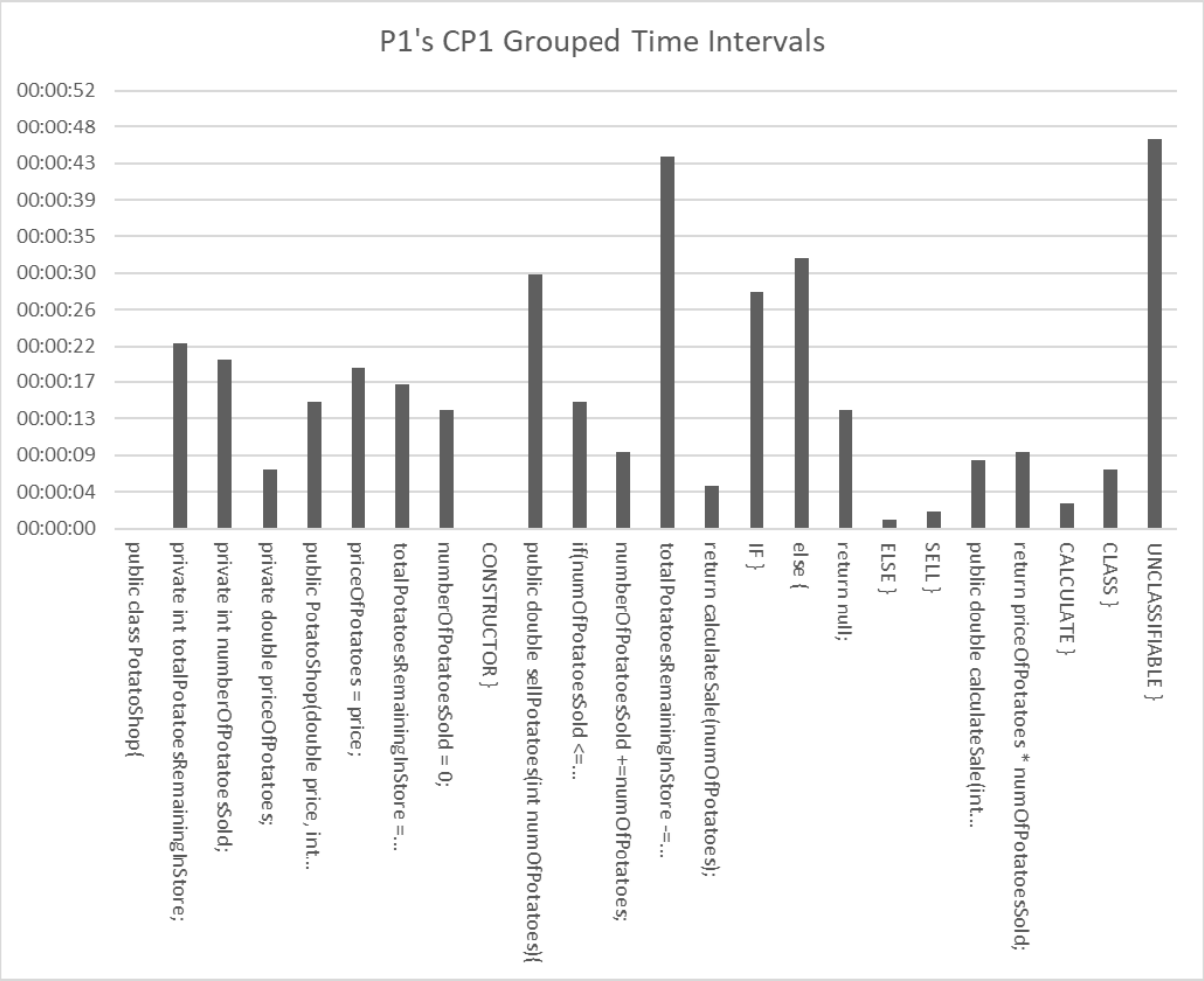


Figure 22: A bar chart of P1's Grouped Time Intervals – i.e., the sum of the time taken to place each piece - for CP1

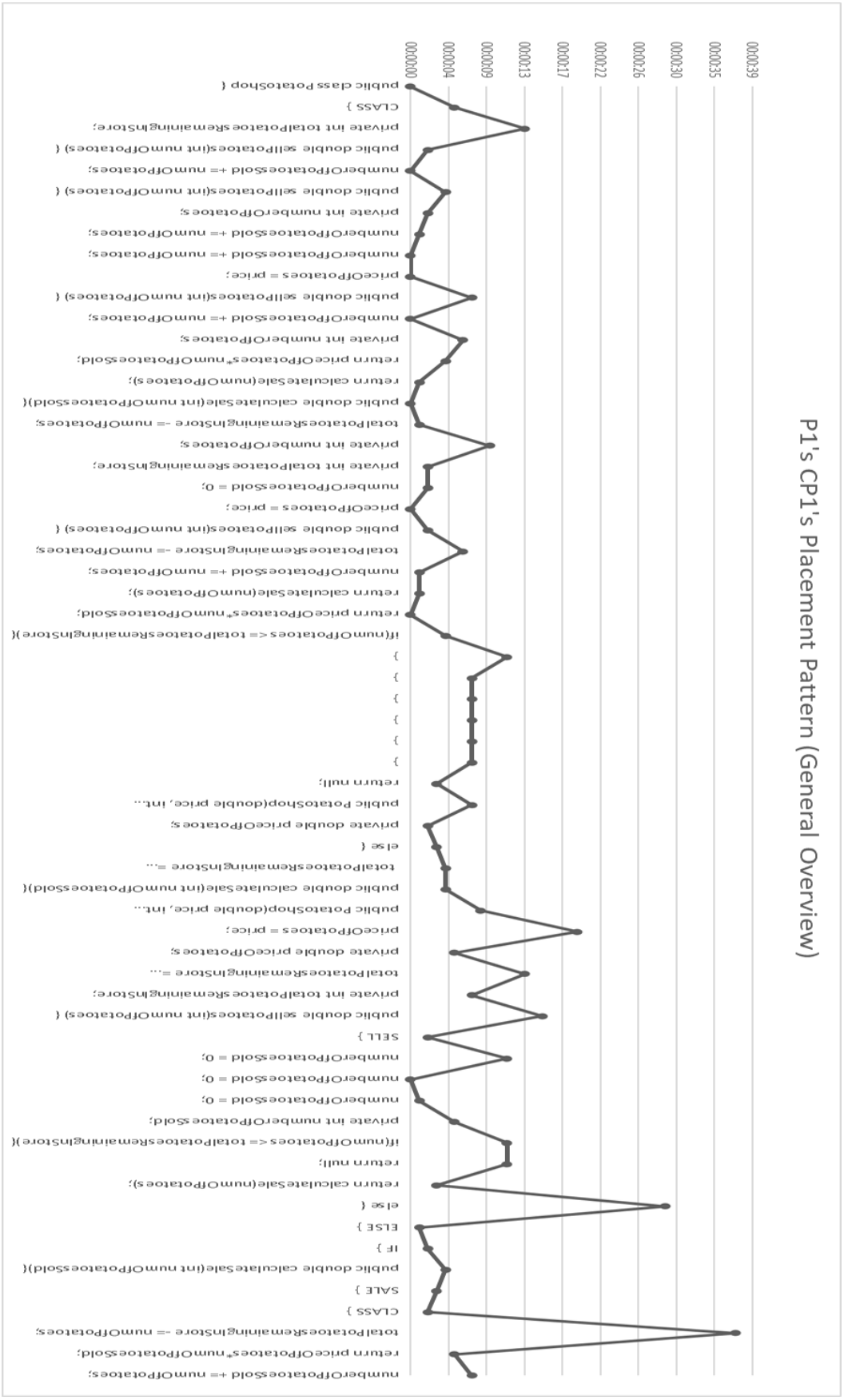


Figure 23: A line graph presenting a general overview of P1's time placement pattern – i.e., the time for each individual movement – for CP1.

P2 spent the most time on the sellPotatoes method and also in the field initialisation aspects of the class, they did not use any grouping movements, but did use a remove and a new movement – 'decide' – which was also unexpected. Decide movements occur when the participant is holding the piece for a prolonged period of time, and, in some cases, where the participant places the piece in front of them as the one to concentrate on before placing it in the final solution space (see Figure 24 and Figure 25).

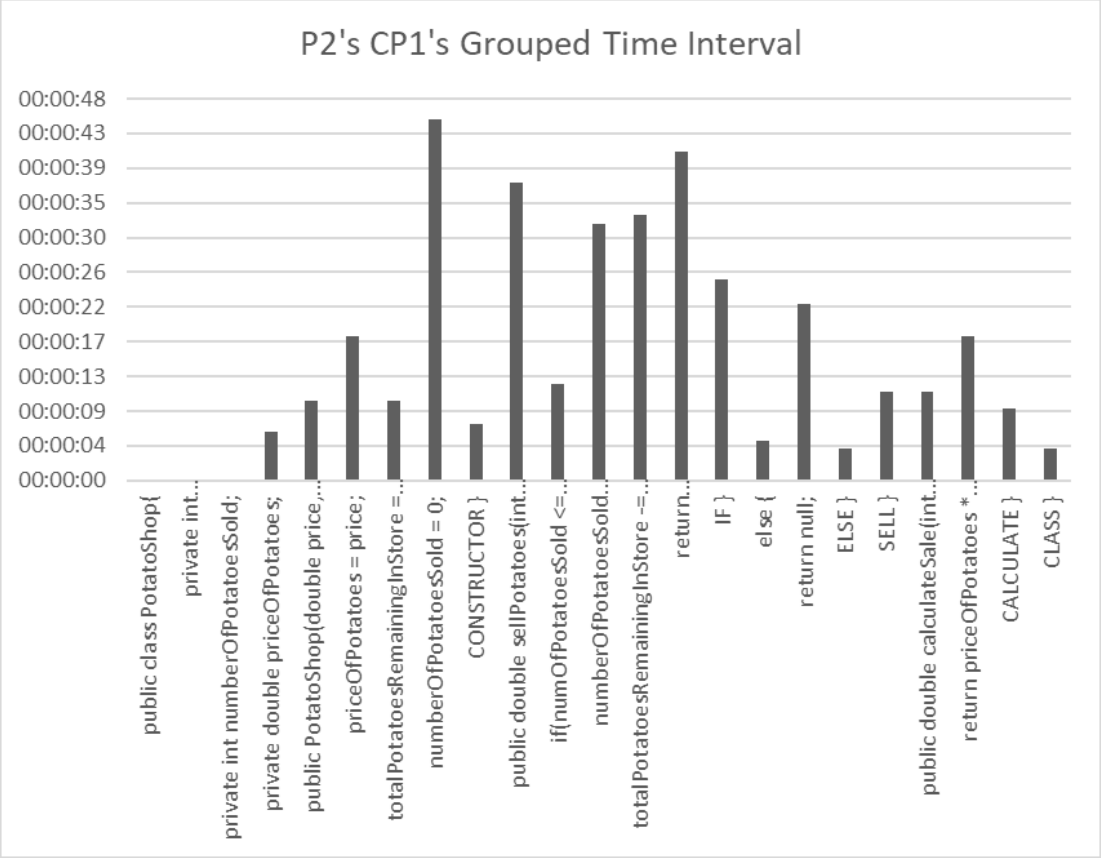


Figure 24: A bar chart of P2's Grouped Time Intervals

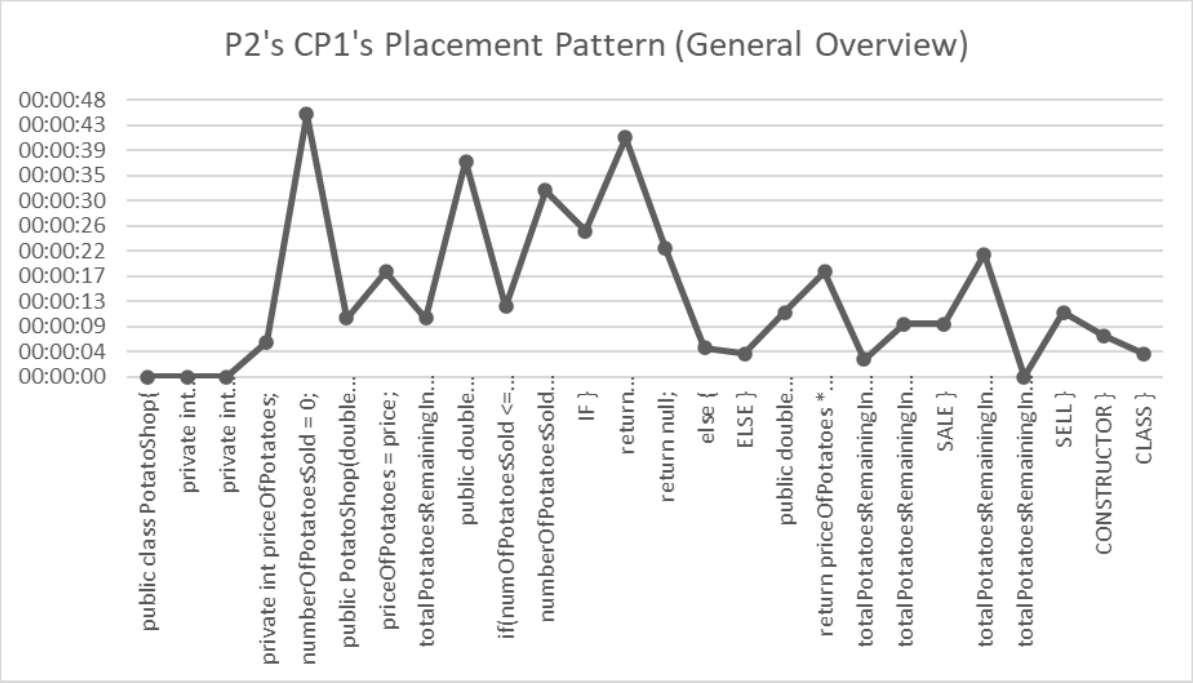


Figure 25: A line graph presenting a general overview of P2's time placement pattern for CP1

P3's movements were unfortunately lost due to video corruption, although the audio file did remain intact, their movements were quite different from those recorded here based on the audio file. The audio implicated that P3 would not have successfully finished the puzzle without the explicit guidance of their mental processes through the puzzle. This, in effect, did suggest that the think aloud protocol could be used in isolation with the 2D Parson's problems acting as a medium in which to inspire discussion with the observer. The participant struggled to finish the task as they failed to understand the logic behind some of the prewritten pieces.

P4 swapped 'numOfPotatoesSold += numOfPotatoes;' with 'return calculateSale(numOfPotatoes);' after realising that Java return statements are the last line executed in a method as they originally had placed the incremental counter after the return statement which would not be reachable. The statement returns for calculateSale and sellPotatoes were confused by P4 initially, but this could be explained by calculateSale only having the one return statement and the only clue that it does not belong in sellPotatoes is the name of the variable name matching the parameter for calculateSale but not sellPotatoes. A critique of this is that the variable names of numOfPotatoes and numOfPotatoesSold are very similar but the idea behind this is to see whether the NPs could notice this issue in their code. P2 and P4 struggled with the incremental counter and distinguishing which return statement belonged to what function, which may implicate that more thought needs to be put into the reasoning behind the return statements and incremental counter in comparison to the other pieces available to the participants (see Figure 26 and Figure 27).

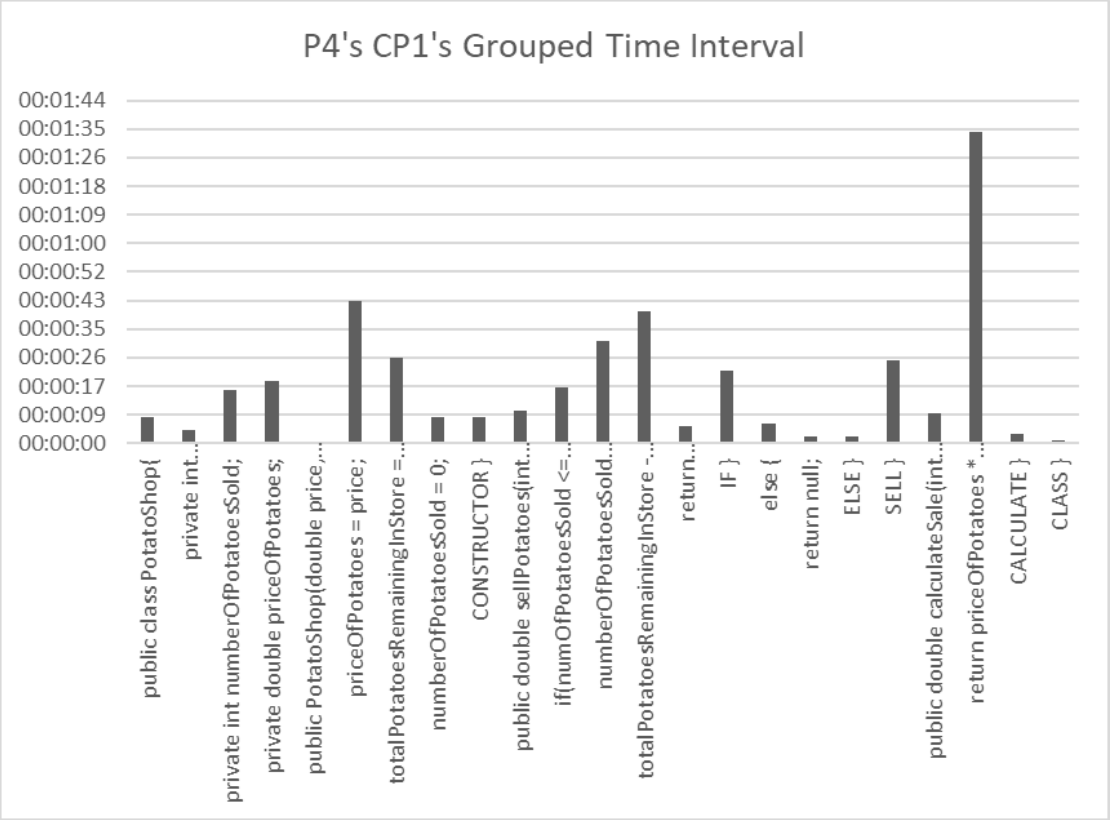


Figure 26: A bar chart of P4's Grouped Time Intervals for CP1

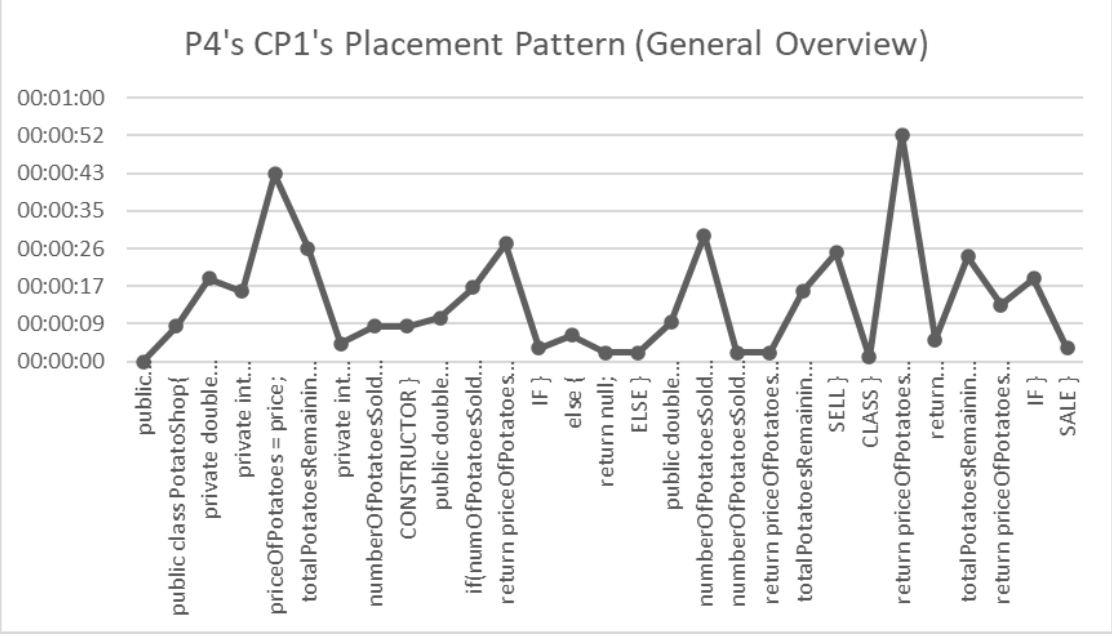


Figure 27: A line graph presenting a general overview of P4's time placement pattern for CP1.

P5 did not notably show any points of interest during their movements, but their pattern placement shows that the spent they had very little difference between peaks implying that did not find the puzzle very difficult to complete as they did not spend a long time considering certain pieces in

comparison to the other participants. P5 did become confused between returning the calculation of the sale in comparison to returning the method calculate sale (see Figure 28 and Figure 29).

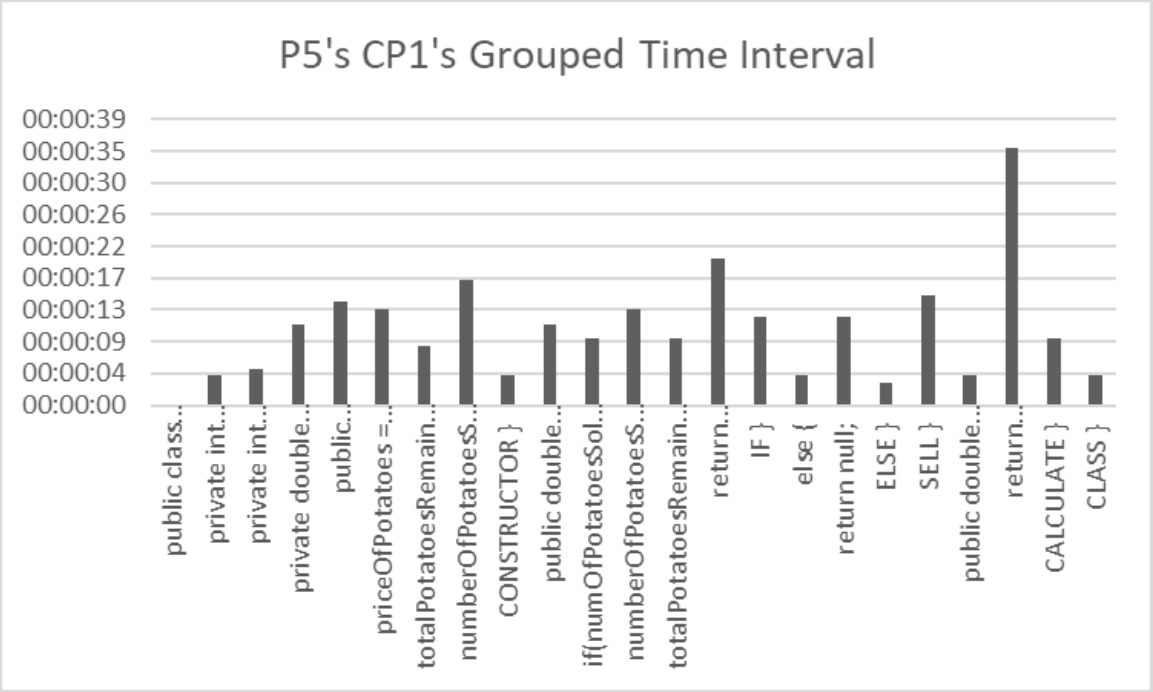


Figure 28: A bar chart of P5's Grouped Time Intervals for CP1.

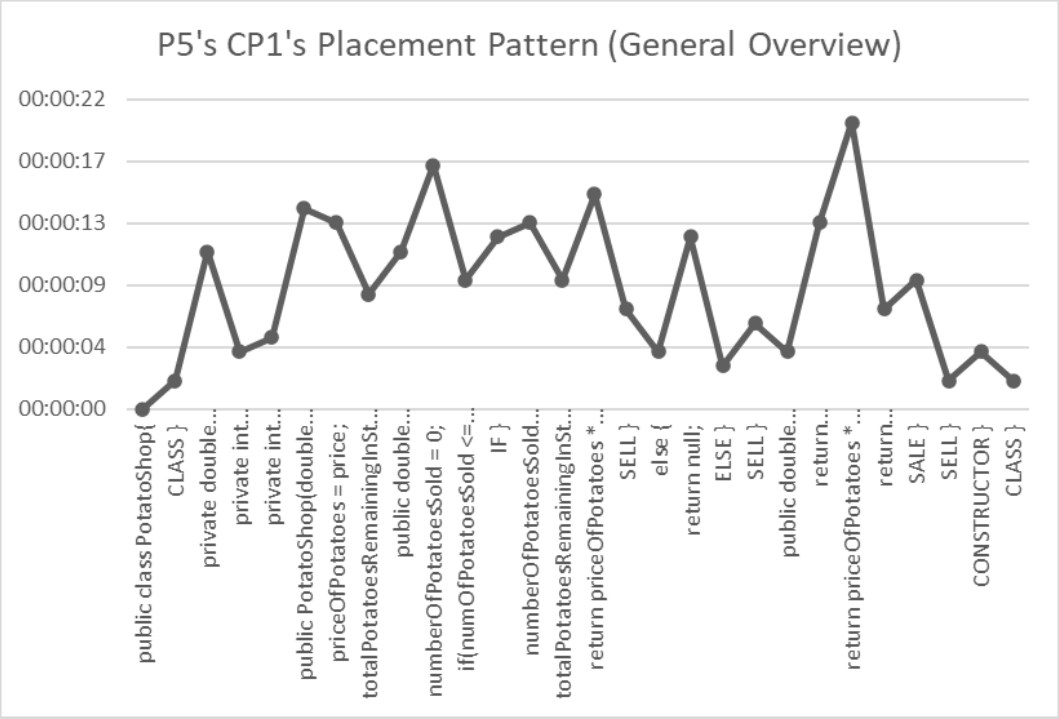


Figure 29: A line graph presenting a general overview of P5's time placement pattern for CP1.

The average time indicates that the pieces participants tended to struggle with were the latter parts of the class for CP1 – this is potentially due to confusing calculate sale method's return statement with the sell potatoes method's return statement. Participants barely spent any time on the earlier parts of the class associated to field declarations and the constructor (see Figure 30).

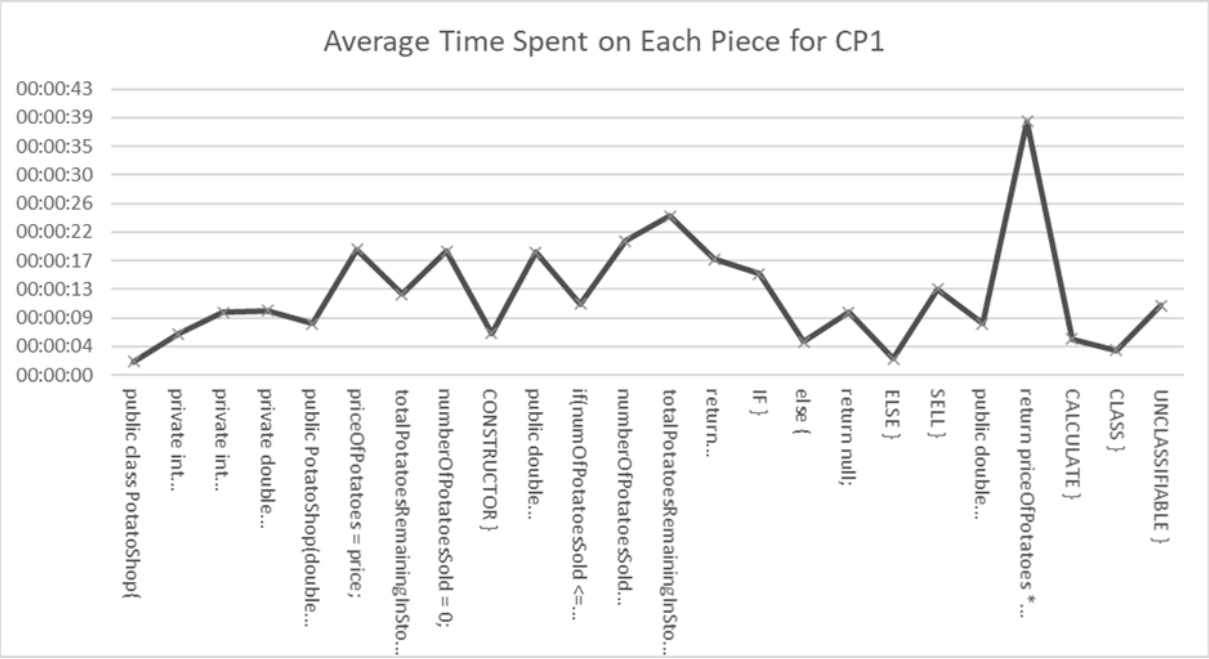


Figure 30: A line chart which demonstrates the average time each participant spent on each part of CP1.

5.3.1.2 Code Puzzle 2 Time Intervals

P1's CP2 placement pattern was greater in length than other participants' patterns in the pilot study (see Figure 34), and thus deserves closer inspection as to how and why this participant differed from others. P1 grouped their pieces together initially, and according to the placement pattern and audio transcript, spent little time on each piece or explaining the meaning behind each piece. Once the solution creation for CP2 began, the 'if' piece had a considerable pause prior to placement where, according to the audio log, the participant was carefully considering the outcome of the if condition and order of consequent pieces. From these datapoints, it is suggested that the number of characters on a code puzzle piece does not necessarily dictate how long they will spend on the piece, and that it is important to consider the context that the piece is being placed in and whether this contributes to the mental load as better indicators for time taken to place the piece. As all participants' spikes relate to the condition of expiry date being after the current date, it is also recognisable that the number of decisions and branching that the code in the puzzle has contributes to the overall difficulty and time required to complete the puzzle. Straightforward context behind pieces, such as

the declaration of the class and fields, took less time on average to place than those with contexts such as decisions implying that context complexity contributes to the level of mental load. No participants recognised that the 'if' was not required, and that they could return just the Boolean generated from Java's Date's after method.

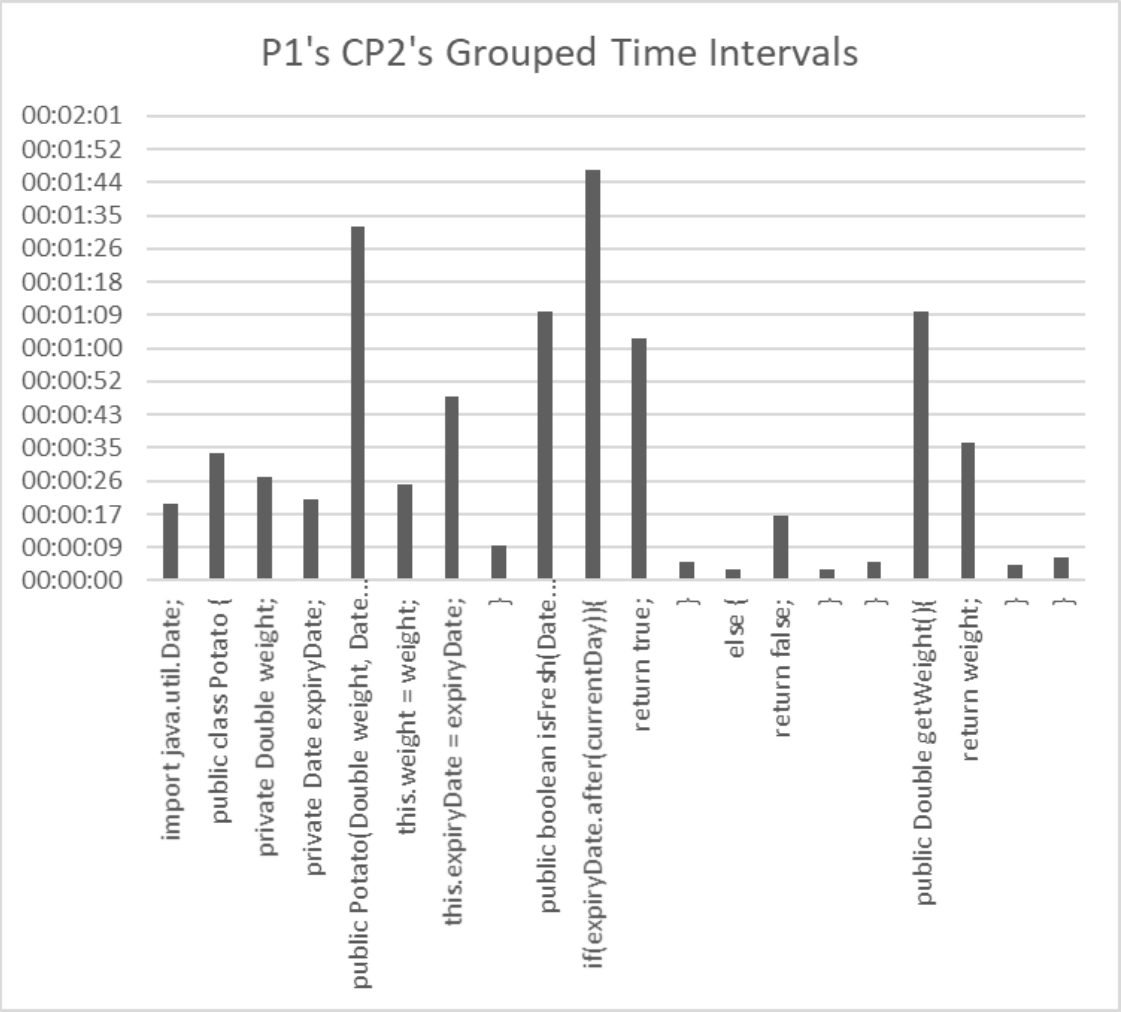


Figure 31: A bar chart of P1's Grouped Time Intervals

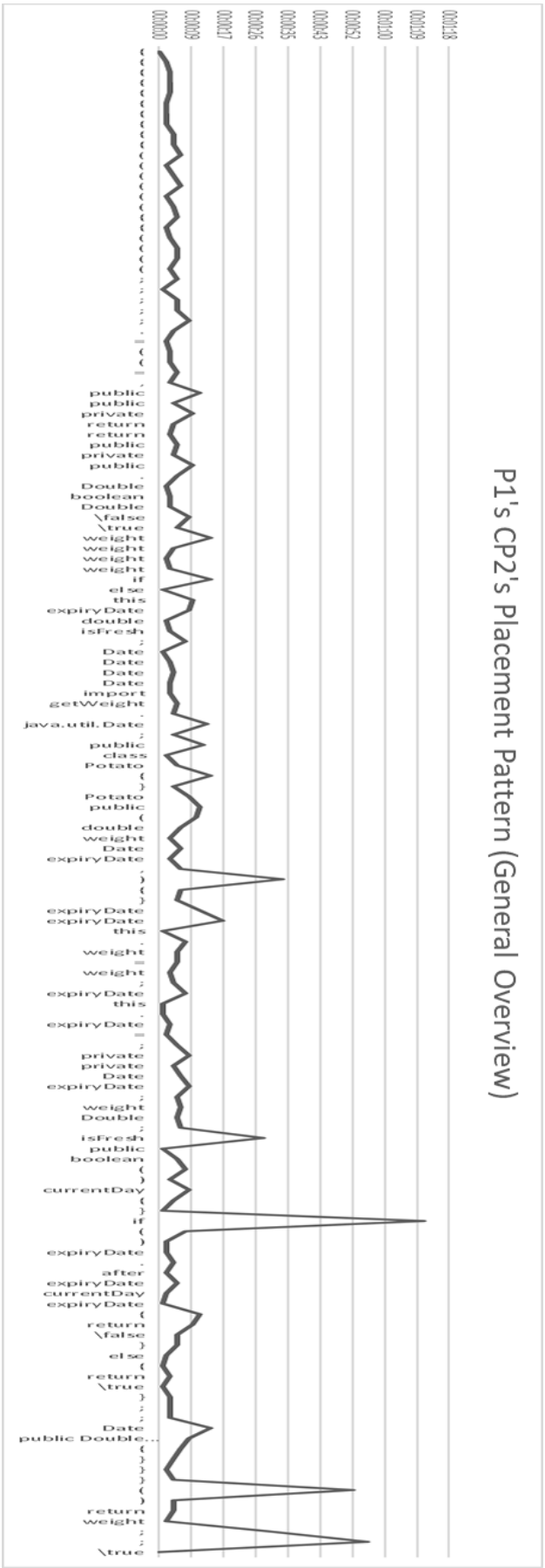


Figure 32: A line graph presenting a general overview of P1's time placement pattern for CP2

.

P2 similarly struggled with the condition of isFresh, but also spent a while detecting the missing semi-colon in a field initialisation line of code. The missing semi-colon does not indicate a lack of understanding of the participant, however, and was likely a genuine error in which they had forgot to include it until they checked the constructor's initialised fields – this can be deduced because the participant used the semi-colon correctly in all other instances of the class, including the other field initialisation, which indicates the difference between not knowing about a concept and making a mistake. It is important for any algorithm to differentiate between the two – as a repeated error generated consistently throughout the class is more likely to show a poor understanding of where a piece should be placed in comparison to a one-off error. This suggests that the code puzzles need to include multiple instances of aspects in order to truly test the participants' understanding.

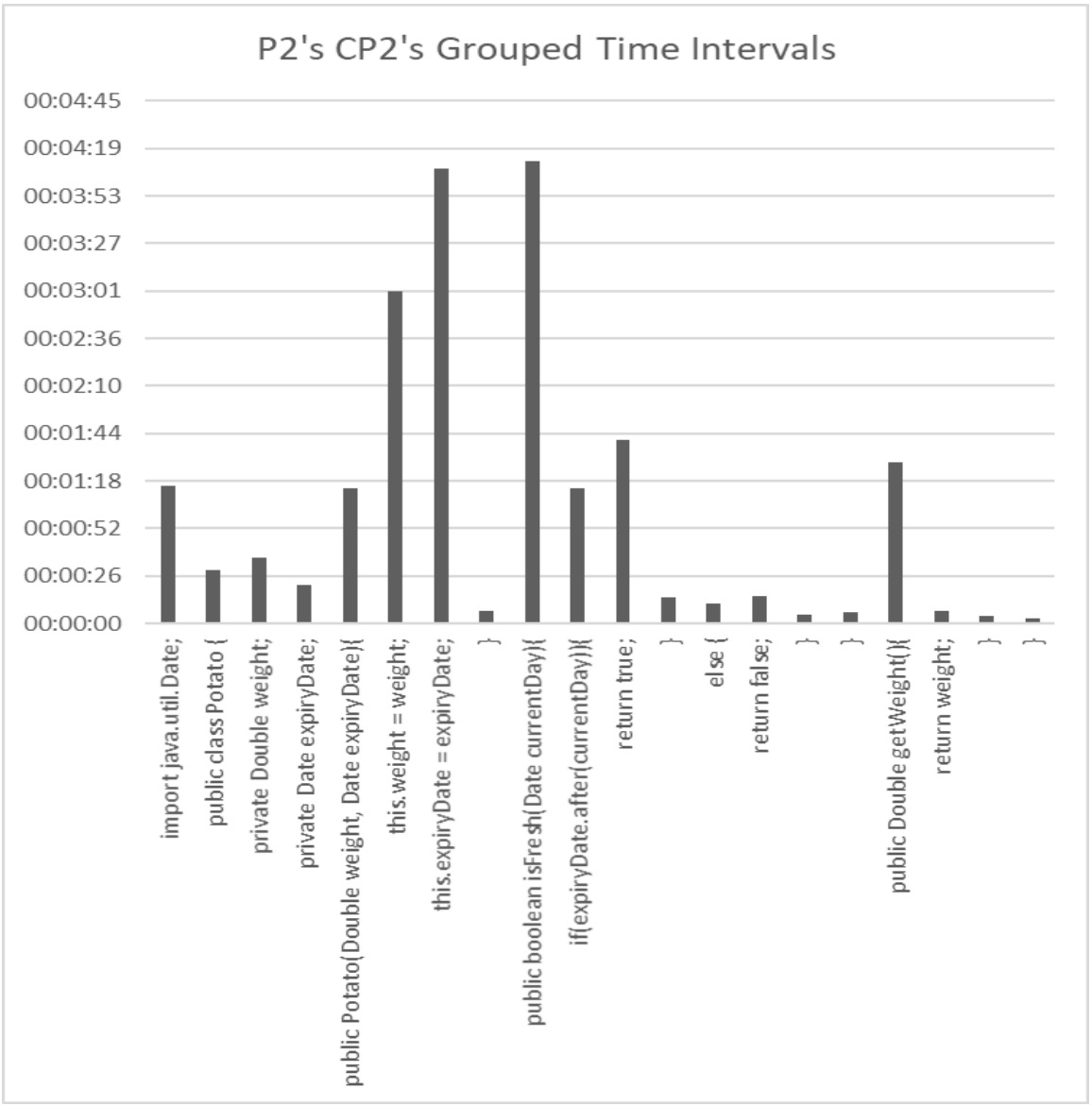


Figure 33: A bar chart of P2's Grouped Time Intervals

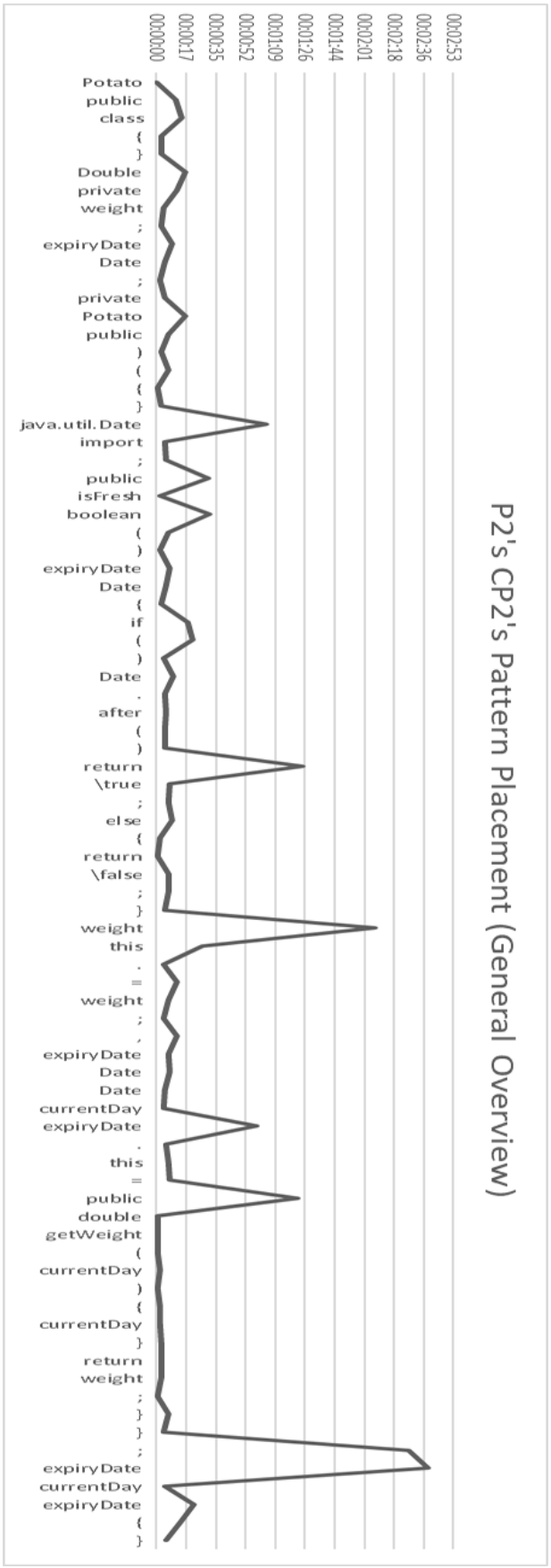


Figure 34: A line graph presenting a general overview of P2's time placement pattern for CP2

P3 showed the most errors in their solution when compared to others and decided to create a custom piece despite the pilot study not specifying that this was possible to do. They chose to create current day as a class, rather than as a parameter for the isFresh method and their pattern shows a series of quick movements in comparison to others who had spikes in their movement patterns. The necessity of the custom piece illustrates that the participant did not understand the aspect of parameters and did seem more used to creating fields for the class.

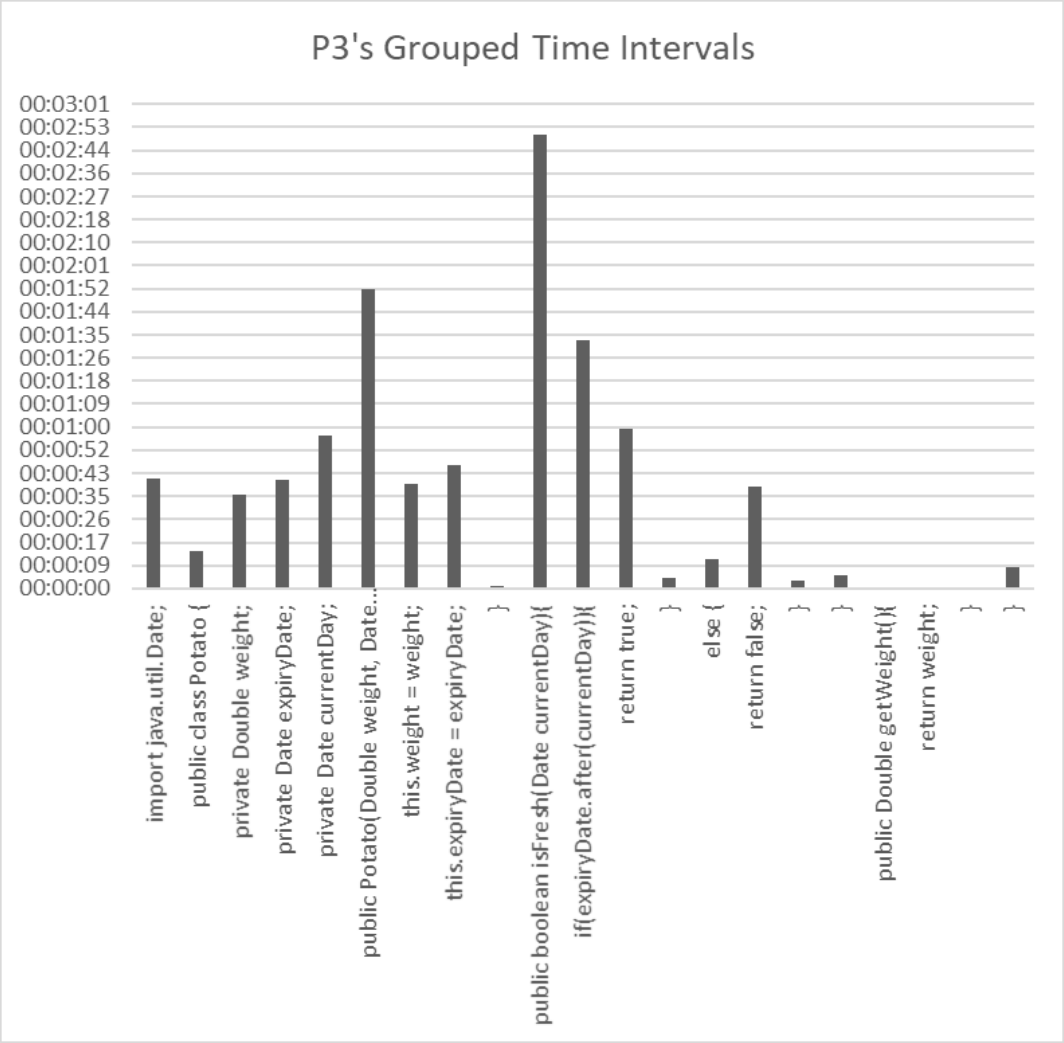


Figure 35: A bar chart of P3's Grouped Time Intervals for CP2

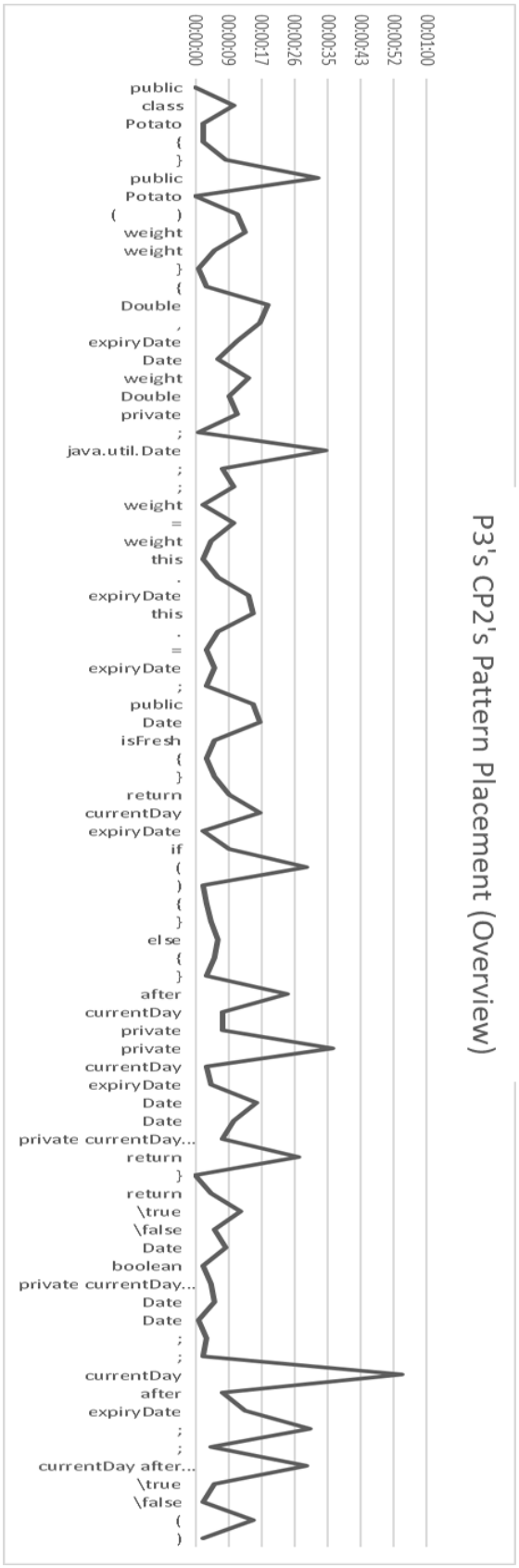


Figure 36: A line graph presenting a general overview of P3's time placement pattern for CP2 and a table documenting the top five pieces that had the longest time intervals.

P4 did not tend to struggle with CP2 and did not have any overly long peaks in their movement pattern, this again suggests that P4 found the puzzle somewhat easy to complete as the aspect they tended to focus the most on was whether the condition would return true or false.

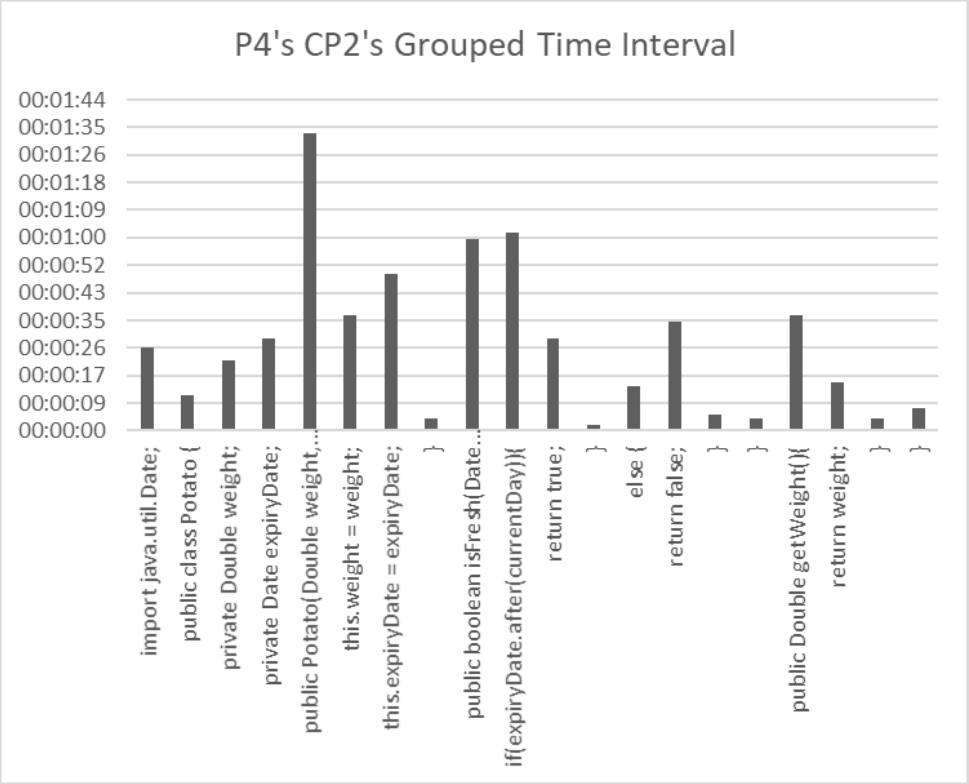


Figure 37: A bar chart of P4's Grouped Time Intervals for CP2

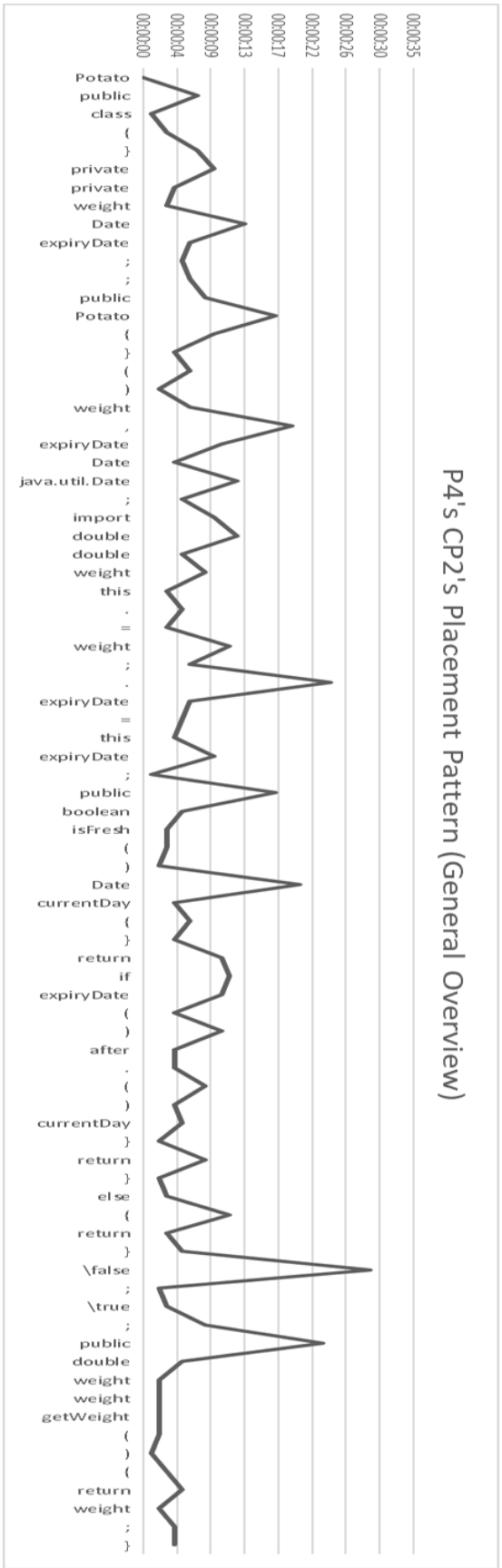


Figure 38: A line graph presenting a general overview of P4's time placement pattern for CP2

P5 tended to spend the most time on the pieces associated to the isFresh method's condition but did not show any other points of interest in their movement pattern.

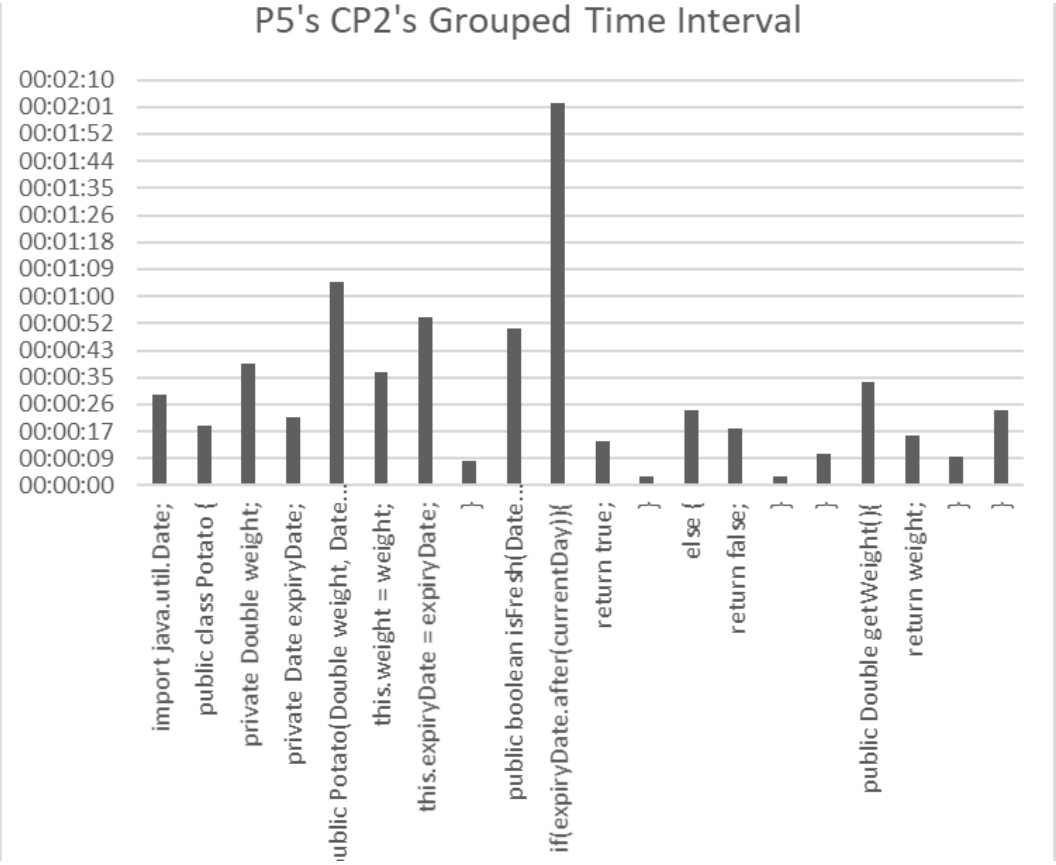


Figure 39: A bar chart of P5's Grouped Time Intervals for CP2

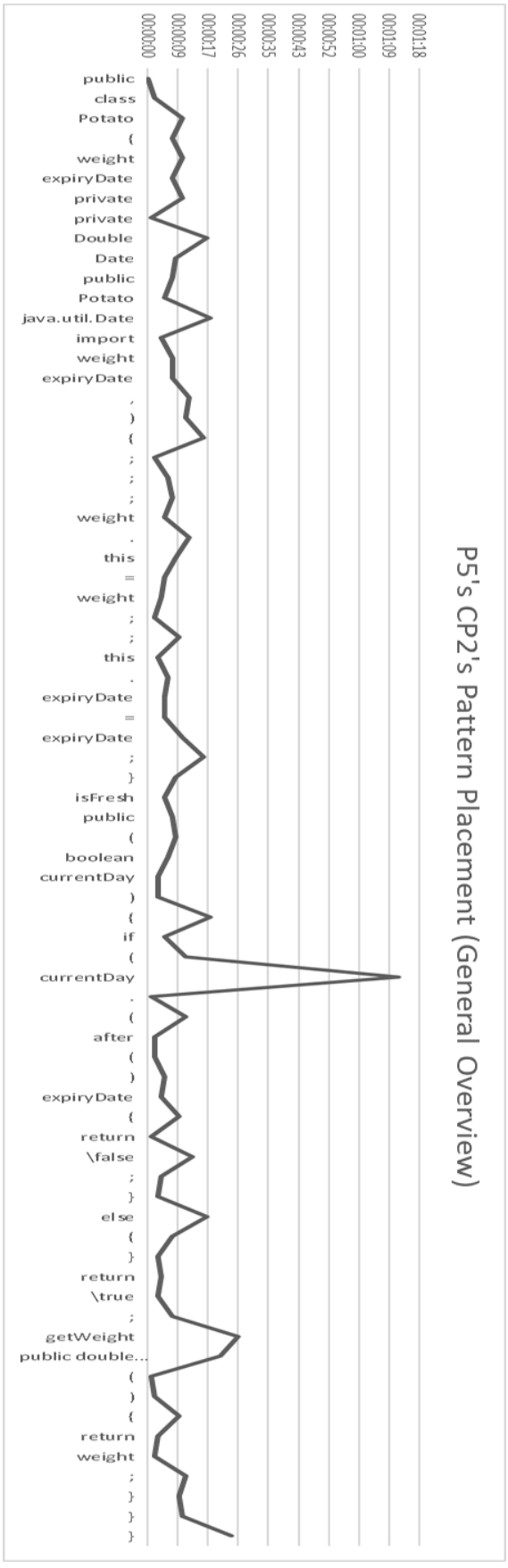


Figure 40: A line graph presenting a general overview of P5's time placement pattern for CP2

In comparison to CP1, it is clear that there is a shift in concentration from the return statements towards the conditional statement. This is not surprising, considering the conditional statement requires thought if used – the returns also were considered carefully but the return of true did not require as much thought as it was typically the else return, while the if statement's return of false required more thought as it was the initial Boolean return.

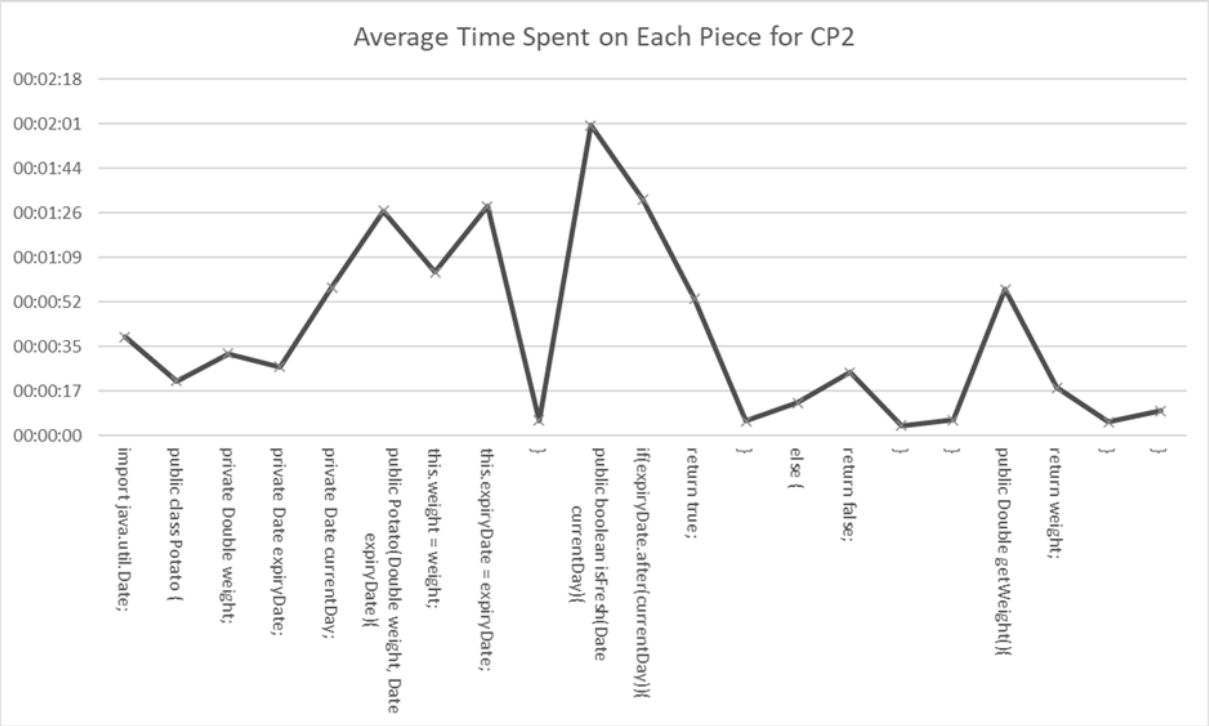


Figure 41: A line chart which demonstrates the average time each participant spent on each part of CP2.

To analyse whether there is a template for the placement pattern, the chronological pattern of participants was calculated using their chronological ordering of movements and averaged based on chronological step to produce Figure 42. While there is a spike at the end, this is likely artificial as the average of the steps were taken for at least two participants and by the beginning of the spike two participants' movement logs had ended. Overall, there does not appear to be any logical pattern – and the beginning of the movement time appeared to be like the end movement time showing that pieces that required the most thought were not always left until the end.

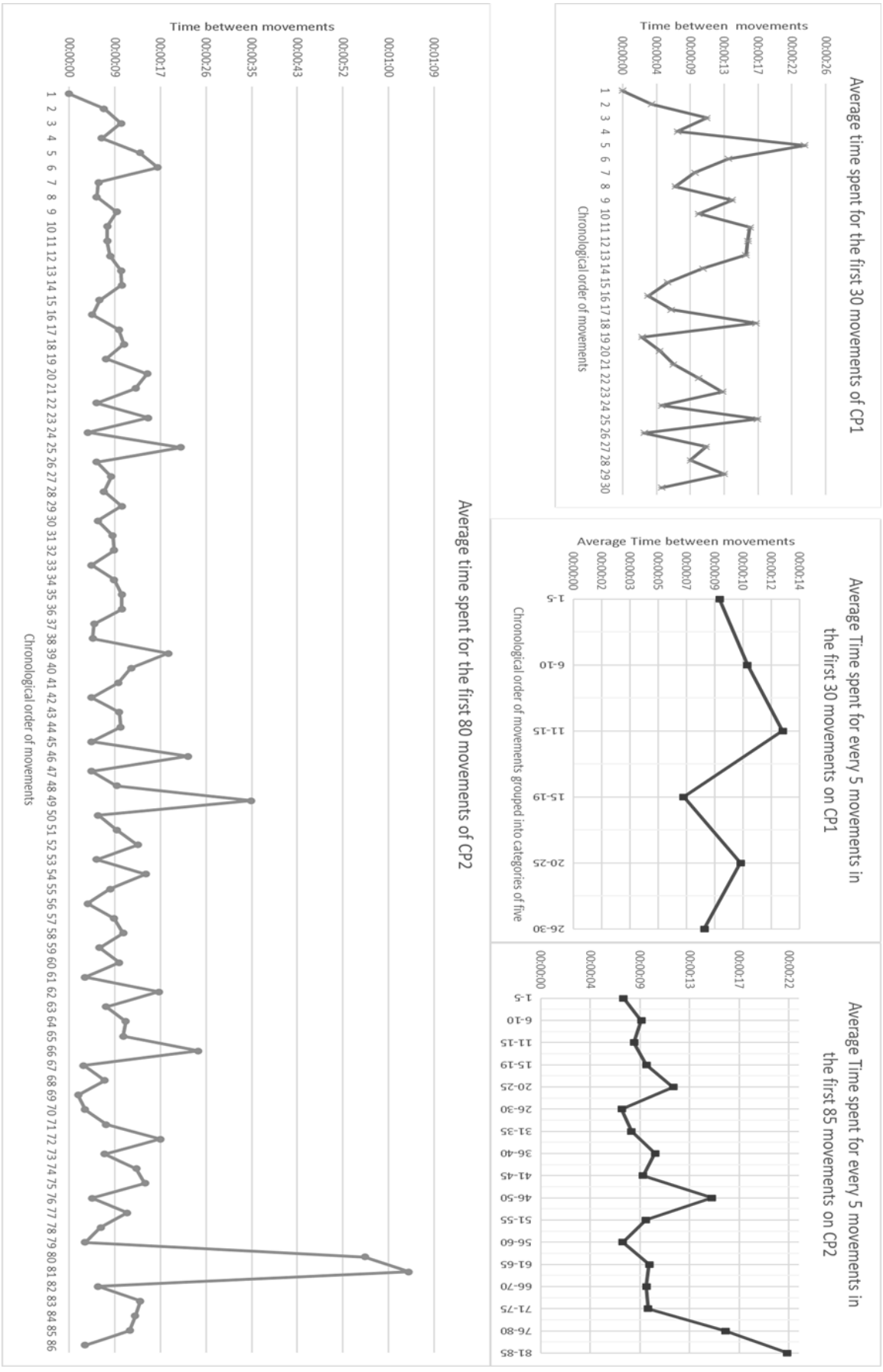


Figure 42: Series of line charts showing the average time taken chronologically for CP1 and CP2.

5.3.2 Movement Observations

5.3.2.1 Frequency of Movements

All participants made less movements for CP1 in comparison to CP2 (see Figure 43).

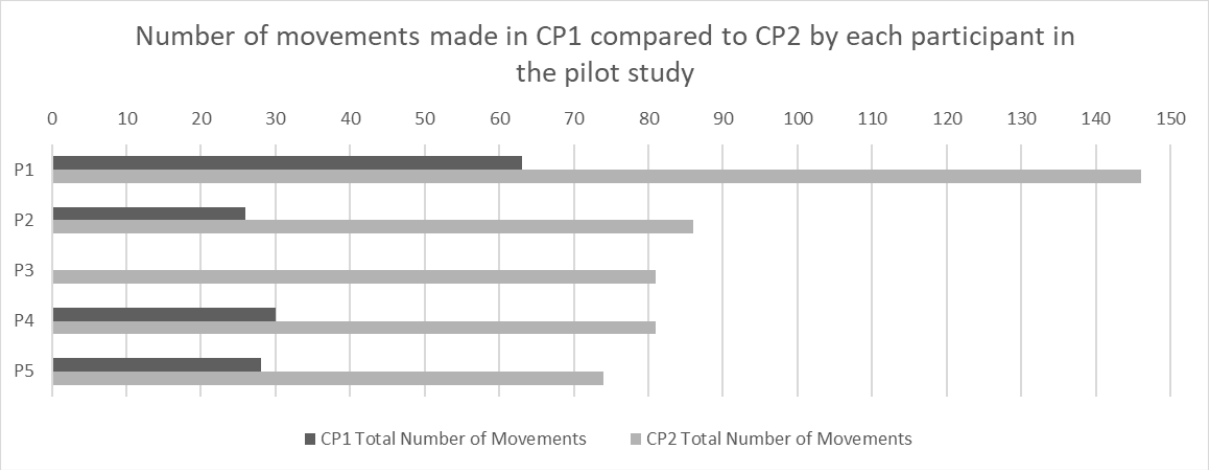


Figure 43: Clustered bar chart for the number of movements made by participants for Code Puzzle 1 (CP1) and Code Puzzle 2 (CP2).

The number of movements indicated how many participants struggled with each part of the puzzle – for CP1, it is expected that the participants should only need one add movement to complete the final solution for each piece, except for 'unclassifiable' pieces, however this was rarely seen as the case for P1 due to their grouping mechanism which involved most of the pieces. The number of times a piece was moved did indicate that there was an issue understanding the piece if more than P1 was performing excessive movements. The piece 'totalPotatoesRemainingInStore -= numOfPotatoes;' caused the most confusion and many participants were trying to determine the intended logic behind the piece and why it should or shouldn't be included in the puzzle. In retrospect, this is likely because the participants were confused by the names – while the researcher did wish to be as specific with the names as possible to avoid confusion, and the intended reasoning behind the piece was to decrease the number of potatoes in store upon a successful sale, participants did seem to not understand this intuitively.

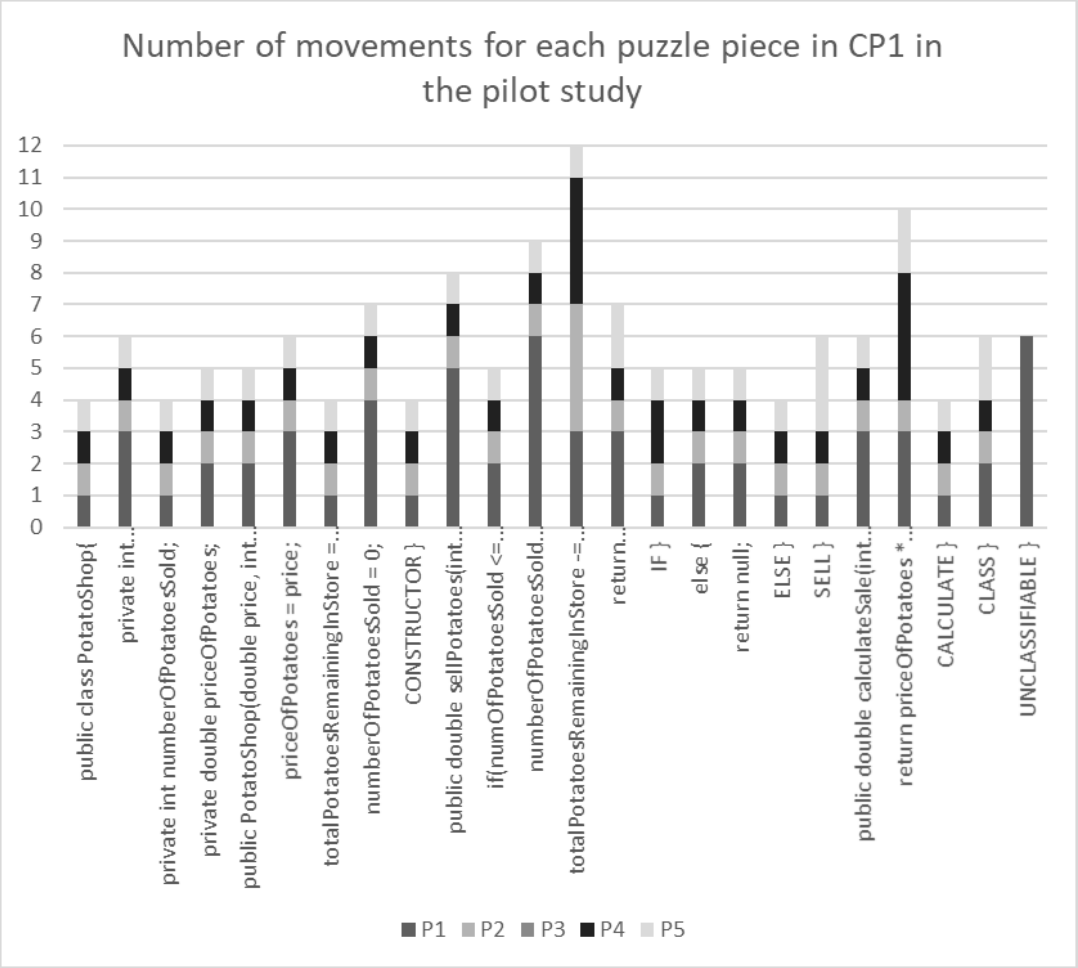


Figure 44: Stacked bar chart illustrating the number of movements made per puzzle piece per participant for CP1.

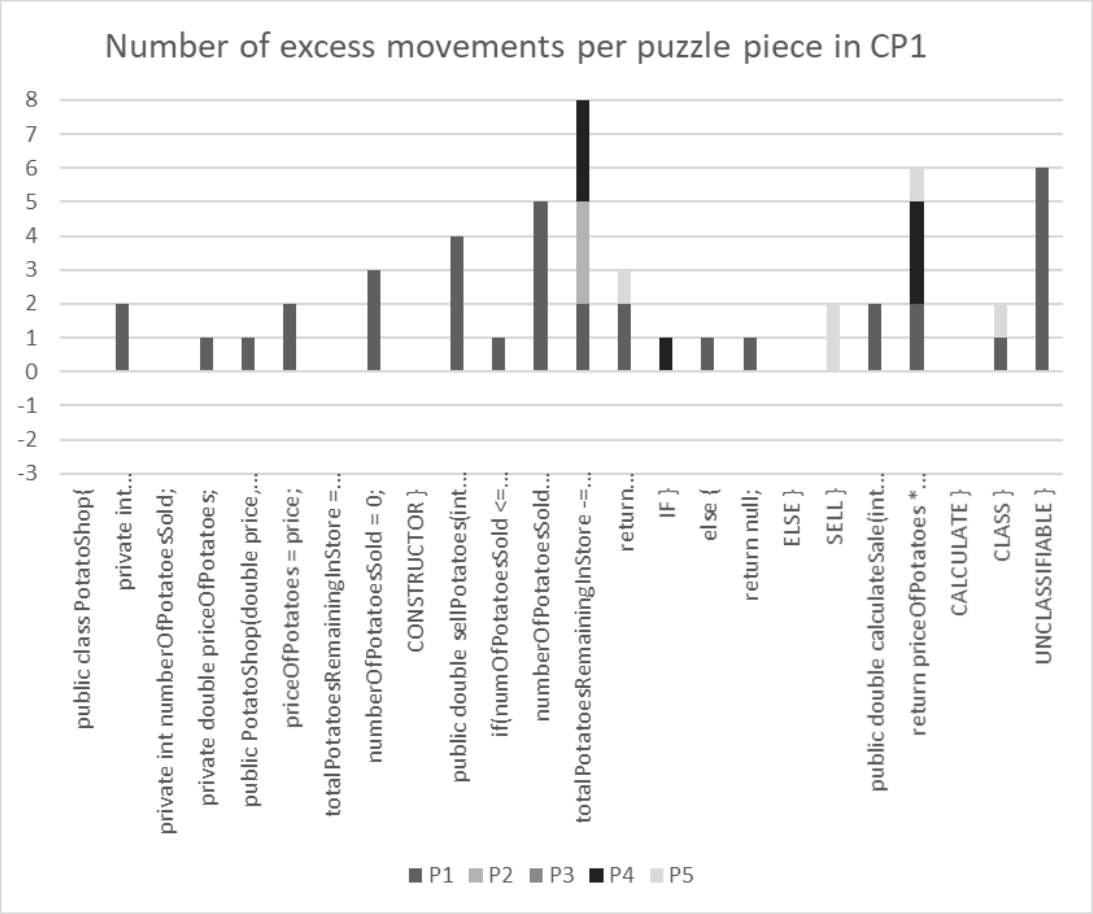


Figure 45: Stacked bar chart that illustrates the number of 'excess' movements –movements that exceed the anticipated number required to construct a line of – recorded for CP1.

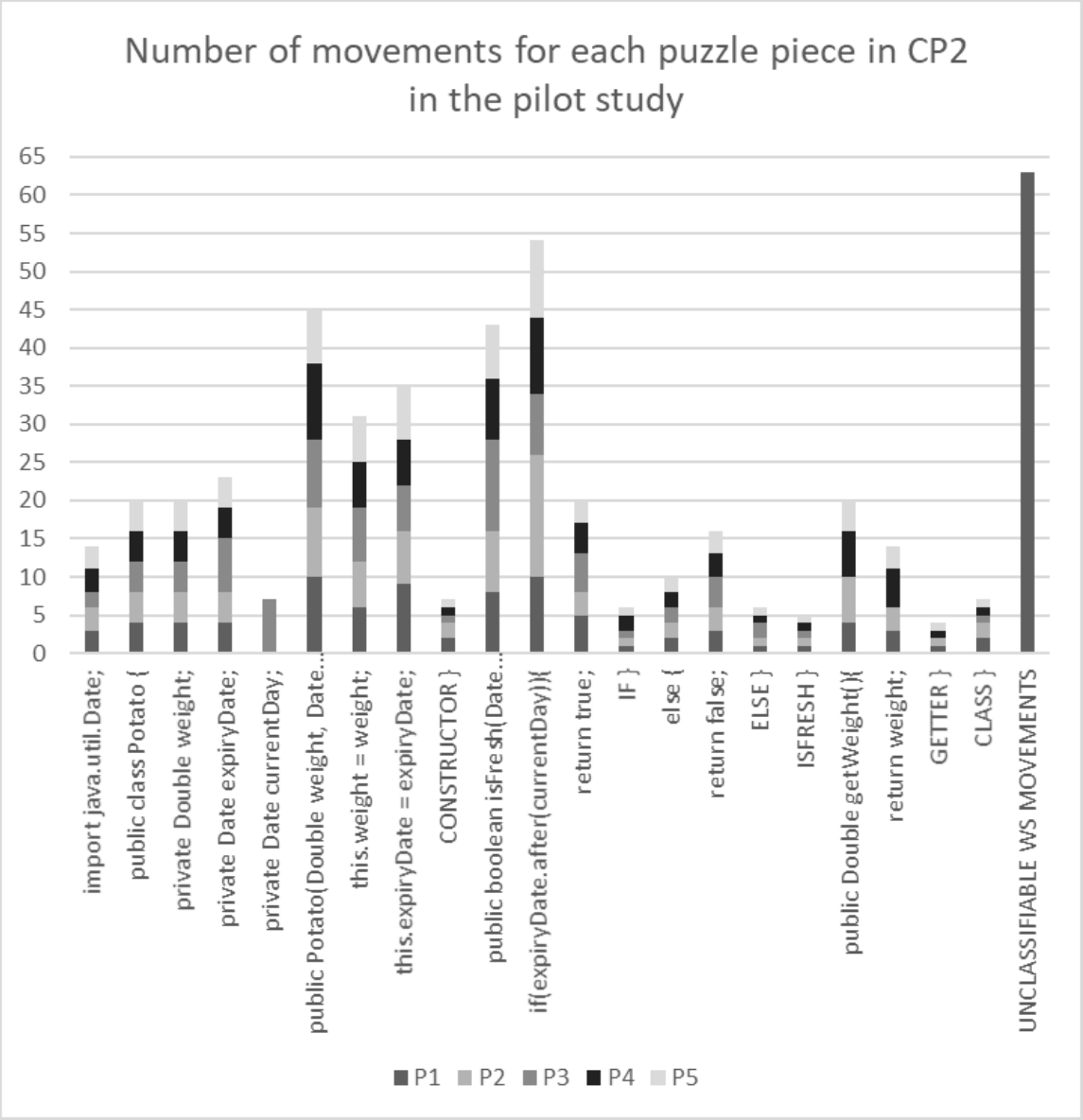


Figure 46: Stacked bar chart illustrating the number of movements made to create each line of code per participant for CP2.

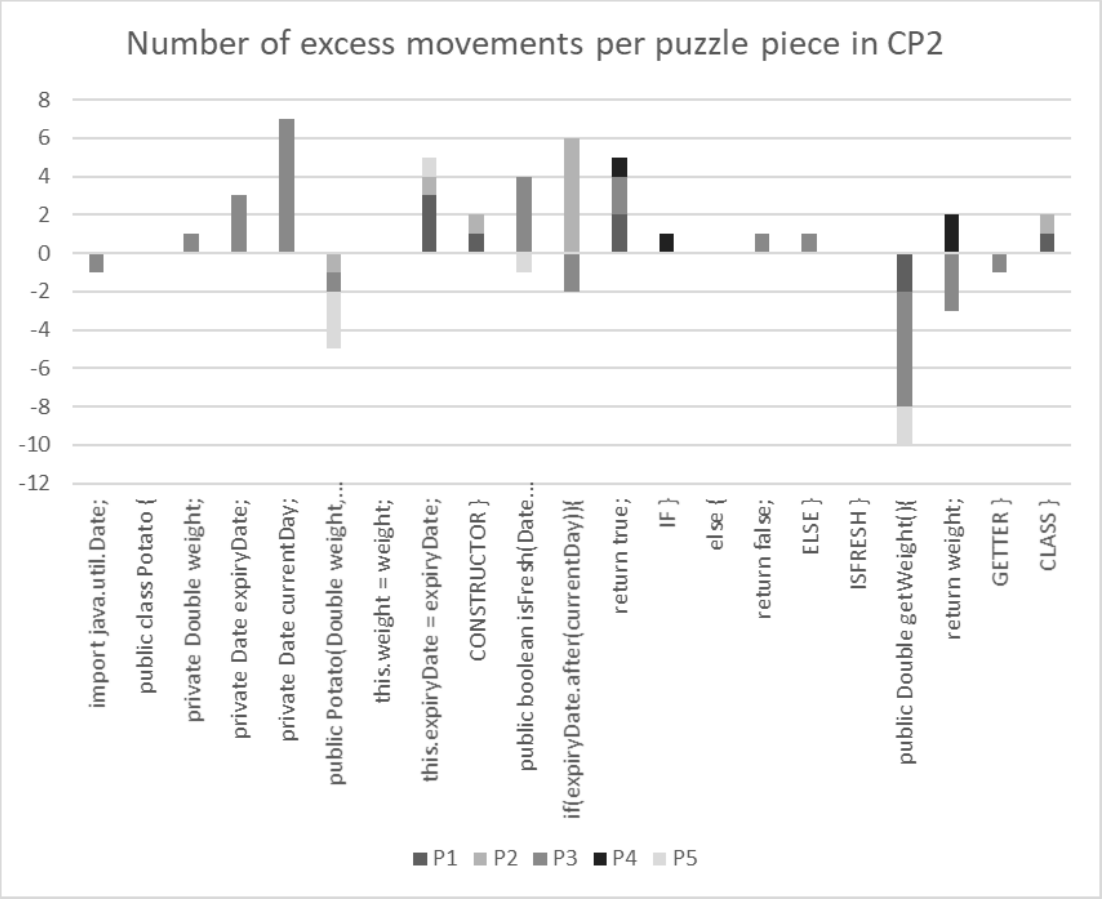


Figure 47: Stacked bar chart that illustrates the number of 'excess' movements –movements that exceed the anticipated number required to construct a line of – recorded for CP1. P1's 63 unclassifiable movements are omitted.

In Figure 45 and Figure 47, excess movements were calculated based on the assumption that one add move is required per piece in CP1, and the number of individual pieces required to create a line of code is required for each line of CP2. A positive number indicates that piece took more movements than anticipated, whereas the negative indicates that the piece took fewer movements than required to complete. Fewer movements did not indicate that the piece was missing – in actuality, participants began to group the pieces and add them at the same time to the final solution space for the getter method in CP2 and for parts of the constructor as well. Grouping, in this instance, indicated that the participant was very confident with the ability to construct the line and that it required little thought based on the audio transcripts related to the participants' feelings associated to the method construction. Grouping was not seen in previous studies and was therefore not anticipated. Previous studies focused on the number of movements performed on pieces, however, this research noted that even participants who had a negative number of excess movements could achieve a correct final solution and that the number of movements or excess movements was not indicative of whether they are struggling to complete the puzzle. Yet these

findings suggest that the number of certain types of movements (such as removing pieces from the final solution space) may be a better indicator – therefore it is not the quantity of movements observed that we should study, but the number of types of movements observed and the reasoning behind them.

5.3.2.2 Order of Movements

The order of movements suggested that the participants tend to work from top of the class to the bottom of the class, with a few deviations. Yet, their dialogue and reasoning implicated that there were subtle differences in approaches that did resonate with the movement order. While participants did tend to forget the import at the start of the class, the constructor or the class definition were always the starting point for participants engaging with the puzzles that were not using the workspace. P4's approach was particularly interesting – as they did not establish the data type of the weight variable until they had investigated whether weight should be an integer or a double. The confusion as to whether a variable should be of a certain type suggests that the way the line order is constructed does indicate the type of reasoning that the participants use to create the class – and that a missing piece on a line may not immediately indicate that the participant does not understand the concept. It was assumed that participants would finish one line of code before proceeding onto the next, but participants tended to complete a section of the class before moving onto the next thus there is often a switch between completing field declarations rather than completing a full line. The lack of completion of full lines suggests that participants do struggle to find pieces when there are a lot in the randomised solution space and that participants place key words that look relevant to the section they are working on before moving onto the next. Participants exhibited signs from the audio transcripts that they were using the task itself to scaffold the class, to the degree that participants did tend to follow the task description and read it aloud at the beginning of each section.

Code Puzzle 1 (CP1) P1's Approach

```
1. | public class PotatoShop { |
3.19.44. | private int totalPotatoesRemainingInStore; |
36.42. | private double priceOfPotatoes; |
7.13.18.50. | private int numberOfPotatoesSold; |
35.40. | public PotatoShop(Double price, int totalPotatoesInStore){ |
10.21.41. | priceOfPotatoes = price; |
20.47.48.49. | numberOfPotatoesSold = 0; |
38.43. | totalPotatoesRemainingInStore = totalPotatoesInStore; |
46. | } |
6.11.22.45. | public double sellPotatoes(int numOfPotatoes){ |
27.51. | if(numOfPotatoes <= totalPotatoesRemainingInStore){ |
5.8.9.12.24.62. | numberOfPotatoesSold += numOfPotatoes; |
17.23.60. | totalPotatoesRemainingInStore -= numOfPotatoes; |
15.25.53. | return calculateSale(numOfPotatoes); |
63. | } |
37.54. | else { |
34.52. | return null; |
55. | } |
56. | } |
4.16.39.57. | public double calculateSale(int numOfPotatoesSold){ |
14.26.61. | return priceOfPotatoes*numOfPotatoesSold; |
58. | } |
2.59. | } |
28.29.30.31.32.33.|(Unknown context) } |
```

Code Puzzle 2 (CP2) P1's Approach

```
63.|import| 66.|java.util.Date| 67.|;|
68. |public| 69.|class| 70.|Potato| 71.|{|
98.|private| 104.|Double| 103.|weight| 105.|;|
99.|private| 100.|Date| 101.|expiryDate| 102.|;|
74.|public| 73.|Potato| 75.|(| 76.|Double| 77.|weight| 80.|,| 78.|Date| 79.|expiryDate| 81.|)| 82.|{|
86.|this| 87.|.| 88.|weight| 89.|=| 90.|weight| 91.|;|
93.|this| 94.|.| 84.92.|expiryDate| 96.|=| 85.|expiryDate| 97.|;|
83.139.|}|
107.|public| 108.|boolean| 106.|isFresh| 109.|(| 134.|Date| 111.|currentDay| 110.|)| 112.|{|
114.|if| 115.|(| 117.120.122.|expiryDate| 118.|.| 119.|after| -.|(| 121.|currentDay| -.|)| 116.|)| 123.|{|
129.|return| 130.147.|true| 132.146.|;|
126.|}|
127.|else| 128.|{|
124.|return| 125.|false| 133.|;|
140.|}|
113.|}|
135.|public| 135.|Double| 135.|getWeight| 141.|(| 142.|)| 136.|{|
143.|return| 144.|weight| 145|;|
137.|}|
72.138.|}|

1.2.3.4.5.6.7.8.9.10.17.18.|(Unknown context) } |
11.12.13.14.15.16.19.20.21.22.30.31.|(Unknown context) ( |
23.24.25.26.27.58. |(Unknown context) ; |
28.42.65. |(Unknown context) . |
29.32. |(Unknown context) = |
33. |(Unknown context) , |
34.35.39.41.|(Unknown context) public |
36.40.|(Unknown context) private |
37.38.|(Unknown context) return |
43.45.56.|(Unknown context) Double |
44.|(Unknown context) boolean |
46.|(Unknown context) false |
47.|(Unknown context) true |
48.49.50.51.|(Unknown context) weight |
52.|(Unknown context) if |
53.|(Unknown context) else |
54.|(Unknown context) this |
55.|(Unknown context) expiryDate |
57.|(Unknown context) isFresh |
59.60.61.62.|(Unknown context) Date |
64.|(Unknown context) getWeight |
```

Figure 48: P1's movement order for each puzzle (each piece is represented in this format: [order number]|piece|).

Code Puzzle 1 (CP1) P2's Approach

```
1. | public class PotatoShop { |
2. | private int totalPotatoesRemainingInStore; |
4. | private Double priceOfPotatoes; |
3. | private int numberOfPotatoesSold; |
6. | public PotatoShop(Double price, int totalPotatoesInStore){ |
7. | priceOfPotatoes = price; |
5. | numberOfPotatoesSold = 0; |
8. | totalPotatoesRemainingInStore = totalPotatoesInStore; |
25. | } |
9. | public Double sellPotatoes(int numOfPotatoes){ |
10. | if(numOfPotatoes <= totalPotatoesRemainingInStore){ |
11. | numberOfPotatoesSold += numOfPotatoes; |
19.20.22.23 | totalPotatoesRemainingInStore -= numOfPotatoes; |
13. | return calculateSale(numOfPotatoes); |
12. | } |
15. | else { |
14. | return null; |
16. | } |
21. | } |
17. | public Double calculateSale(int numOfPotatoesSold){ |
18. | return priceOfPotatoes*numOfPotatoesSold; |
24. | } |
26. | } |
```

Code Puzzle 2 (CP2) P2's Approach

```
21.|import| 20.|java.util.Date| 22.|;|
2. |public| 3.|class| 1.|Potato| 4.|{|
7.|private| 6.|Double| 8.|weight| 9.|;|
13.|private| 11.|Date| 10.|expiryDate| 12.|;|
15.|public| 14.|Potato| 17.|(| 50.|Double| 51.|weight| 57.|,| 60.|Date| 83.|expiryDate| 16.|)| 18.|{|
52.|this| 53.|.| 49.|weight| 54.|=| 55.|weight| 56.|;|
64.|this| 63.|.| 58.|expiryDate| 65.|=| 62.|expiryDate| 80.|;|
19.78.|}|
23.|public| 25.|boolean| 24.|isFresh| 26.|(| 29.34.59.|Date| 70.73.82.|currentDay| 27.|)| 30.|{|
31.|if| 32.|(| 28.81.83.|expiryDate| 35.|.| 36.|after| 37.|(| 59.61.|currentDay| 33.|)| 38.39.|)| 84.|{|
40.|return| 41.|true| 42.|;|
85.|}|
43.|else| 44.|{|
45.|return| 46.|false| 47.|;|
48.|}|
86.|}|
66.|public| 67.|Double| 68.|getWeight| 69.|(| 71.|)| 72.|{|
75.|return| 76.|weight| 77.|;|
74.|}|
5.79.|}|
```

Figure 49: P2's movement order for each puzzle (each piece is represented in this format: [order number]|piece|).

Figure 50: P3's order of movements – the number to the left of each |piece| is associated to the numerical step in their movement log. Underlined without a number means missing piece and underlined with numbers means custom piece or custom placement of piece.

Code Puzzle 1 (CP1) P4's Approach

```
2. | public class PotatoShop { |
7. | private int totalPotatoesRemainingInStore; |
3. | private double priceOfPotatoes; |
4. | private int numberOfPotatoesSold; |
1. | public PotatoShop(double price, int totalPotatoesInStore){ |
5. | priceOfPotatoes = price; |
8. | numberOfPotatoesSold = 0; |
6. | totalPotatoesRemainingInStore = totalPotatoesInStore; |
9. | } |
10. | public double sellPotatoes(int numOfPotatoes){ |
11. | if(numOfPotatoes <= totalPotatoesRemainingInStore){ |
22. | numberOfPotatoesSold += numOfPotatoes; |
18.19.21.27. | totalPotatoesRemainingInStore -= numOfPotatoes; |
26. | return calculateSale(numOfPotatoes); |
13. | } |
14. | else { |
15. | return null; |
16.29. | } |
23. | } |
17. | public double calculateSale(int numOfPotatoesSold){ |
12.20.25.28. | return priceOfPotatoes*numOfPotatoesSold; |
30. | } |
24. | } |
```

Code Puzzle 2 (CP2) P4's Approach

```
25.|import| 23.|java.util.Date| 24.|;|
2. |public| 3.|class| 1.|Potato| 4.|{|
6. |private| 27.|Double| 8.|weight| 11.|;|
7. |private| 9.|Date| 10.|expiryDate| 12.|;|
13.|public| 14.|Potato| 17.|(| 26.|Double| 19.|weight| 20.|,| 22.|Date| 21.|expiryDate| 18.|)| 15.|{|
29.|this| 30.|.| 28.|weight| 31.|=| 32.|weight| 33.|;|
37.|this| 34.|.| 35.|expiryDate| 36.|=| 38.|expiryDate| 39.|;|
16.|}|
40.|public| 41.|boolean| 42.|isFresh| 43.|(| 45.|Date| 46.|currentDay| 44.|)| 47.|{|
50.|if| 56.|(| 51.|expiryDate| 55.|.| 54.|after| 52.|(| 58.|currentDay| 53.|)| 57.|)| -.|{|
49.|return| 66.|true| 67.|;|
59.|}|
60.|else| 61.|{|
62.|return| 64.|false| 65.|;|
63.|}|
48.|}|
68.|public| 69.|Double| 70.71.|weight| 72.|getWeight| 73.|(| 74.|)| 75.|{|
76.|return| 77.|weight| 78.|;|
79.|}|
5.|}|
```

Figure 51: P4's order of movements – the number to the left of each |piece| is associated to the numerical step in their movement log. Underlined without a number means missing piece and underlined with numbers means custom placement of piece.

Code Puzzle 1 (CP1) P5's Approach

```
1. | public class PotatoShop { |
4. | private int totalPotatoesRemainingInStore; |
3. | private double priceOfPotatoes; |
5. | private int numberOfPotatoesSold; |
6. | public PotatoShop(Double price, int totalPotatoesInStore){ |
7. | priceOfPotatoes = price; |
10. | numberOfPotatoesSold = 0; |
8. | totalPotatoesRemainingInStore = totalPotatoesInStore; |
27. | } |
9. | public double sellPotatoes(int numOfPotatoes){ |
11. | if(numOfPotatoes <= totalPotatoesRemainingInStore){ |
12. |_{_|
13. | numberOfPotatoesSold += numOfPotatoes; |
14. | totalPotatoesRemainingInStore -= numOfPotatoes; |
22.24. | return calculateSale(numOfPotatoes); |
16. | } |
17. | else { |
18. | return null; |
19. | } |
20.26. | } |
21. | public Double calculateSale(int numOfPotatoesSold){ |
15.23. | return priceOfPotatoes*numOfPotatoesSold; |
25. | } |
2,28._| }_|
```

Code Puzzle 2 (CP2) P5's Approach

```
14.|import| 13.|java.util.Date| 22.|;|
1. |public| 2.|class| 3.|Potato| 4.|{|
7.|private| 9.|Double| 5.|weight| -.|;|
8.|private| 10.|Date| 6.|expiryDate| -.|;|
74._|}|
11.|public| 12.|Potato| -.|(| -.|Double| 15.|weight| 17.|,| -.|Date| 16.|expiryDate| 18.|)| 19.|{|
25.|this| 24.|.| 23.|weight| 26.|=| 27.|weight| 28.|;|
30.|this| 31.|.| 32.|expiryDate| 33.|=| 34.|expiryDate| 29.35.|;|
36|}|
38.|public| 40.|boolean| 37.|isFresh| 39.|(| -.|Date| 41.|currentDay| 42.|)| 43.|{|
44.|if| 45.|(| 52.|expiryDate| 47.|.| 49.|after| 50.|(| 46.|currentDay| 51.|)| 48.|)| 53.|{|
61.|return| 62.|true| 63.|;|
57.|}|
58.|else| 59.|{|
54.|return| 55.|false| 56.|;|
60.|}|
73.|}|
65.|public| 65.|Double| 64.65.|getWeight| 66.|(| 67.|)| 68.|{|
69.|return| 70.|weight| 71.|;|
72.|}|
-.|}|
```

Figure 52: P5's order of movements – the number to the left of each |piece| is associated to the numerical step in their movement log. Underlined without a number means missing piece and underlined with numbers means custom placement of piece.

The order of movements did not show any clear indicators of understanding, but when accompanied with the reasoning behind the movements, it becomes clearer as to what participants name various parts of the class and the reasoning behind missing pieces.

Figure 53 was generated by examining the movement transcripts in line with the audio transcripts for each participant in the pilot study and labelling each piece with an associated context. For CP1, this was relatively easy to achieve as the lines of code had been pre-written with an intended purpose that the participants generally adhered to (e.g., all participants knew 'public class PotatoShop {' meant the class definition and/or establishing the class), however, the braces needed context (e.g., whether a closed brace was used with the intent to close the class or a method, or something incorrect like closing a field). All pieces for CP2 needed to be carefully examined as the intent of the piece was required to understand what the participant wanted to do with it. Each movement was labelled with a contextual purpose, and P1's grouping of pieces was omitted – instead their pattern, for the purpose of the approach flowchart, only started from when they vocally established that they were creating their class, and P3's movements for CP1 were unfortunately lost due to a technical glitch, but all other movements were included. The participants' movements were placed in chronological order next to each other alongside the labels, and the modal average was then taken from the pieces. The participant with the shortest movement pattern would simply not contribute to later parts of the flow chart, and the flow chart would stop generation when only one participant's data was left to be used for further movements. This process is far from perfect as it is not taking into account that some participants may spend longer on a certain phase than other participants, and in retrospect it may have been better to then abstract the pieces further to classify them as a general 'part' of the class, but this procedure did reveal a general process that participants typically followed (see Figure 53).

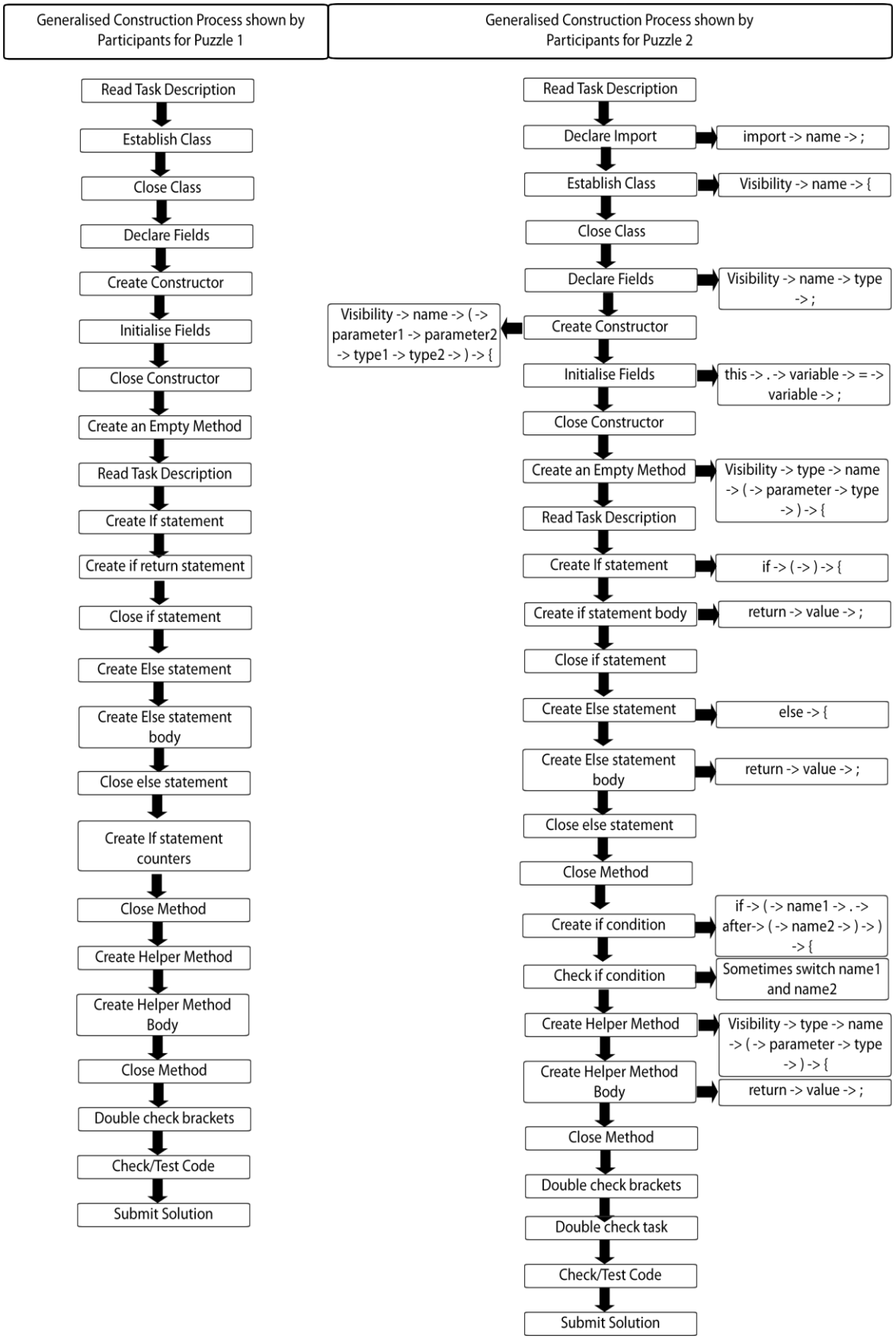


Figure 53: The generalised construction process that participants go through when construction solutions for Puzzles 1 and 2; it should be noted there was very little deviation seen in the approach.

5.3.2.3 Types of Movements Observed

The types of movements observed were very similar between the majority of participants (see Figure 54). P1's movement pattern was a stark contrast to the anticipated movements that were seen to the degree that roughly 50% of their movements were of grouping. While the grouping took place at the beginning of the process, it is still technically a valid movement despite being out of the range of expectations based on the works of previous researchers. While add moves were expected for all participants, as they did need to 'add' to the final solution to complete the puzzle, decide movements were not where the participant placed the piece in front of them and compared it to other pieces in the final solution space.

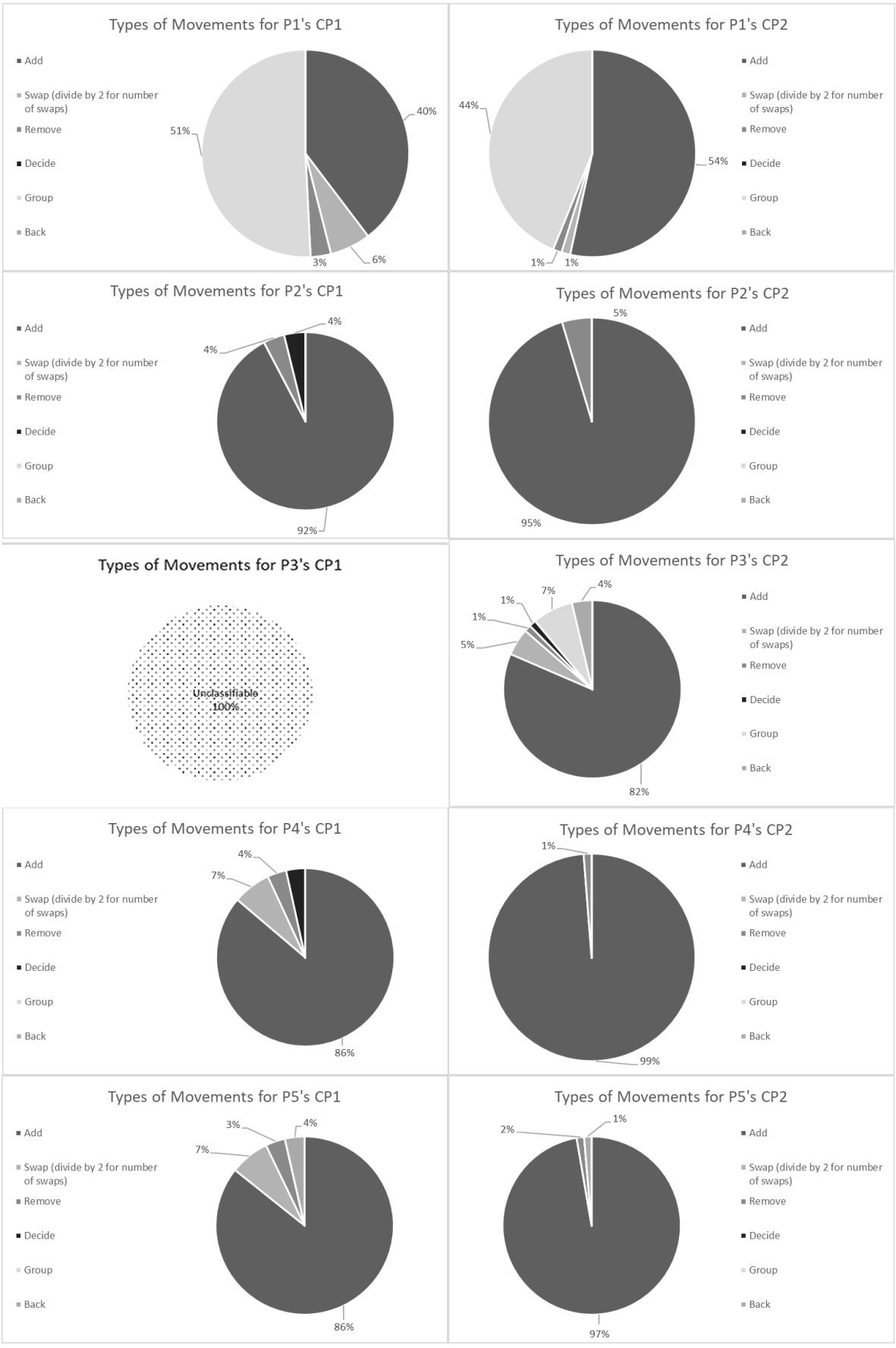


Figure 54: The classification of movement type for each participants' movements in CP1 and CP2.

A series of diagrams (see Figure 55, Figure 56, Figure 57, Figure 58, Figure 59, Figure 60, Figure 60, Figure 61, and Figure 62) have been produced to demonstrate how the new types of movements manifested in the contexts of CP1 and CP2 and also to demonstrate points of interest.

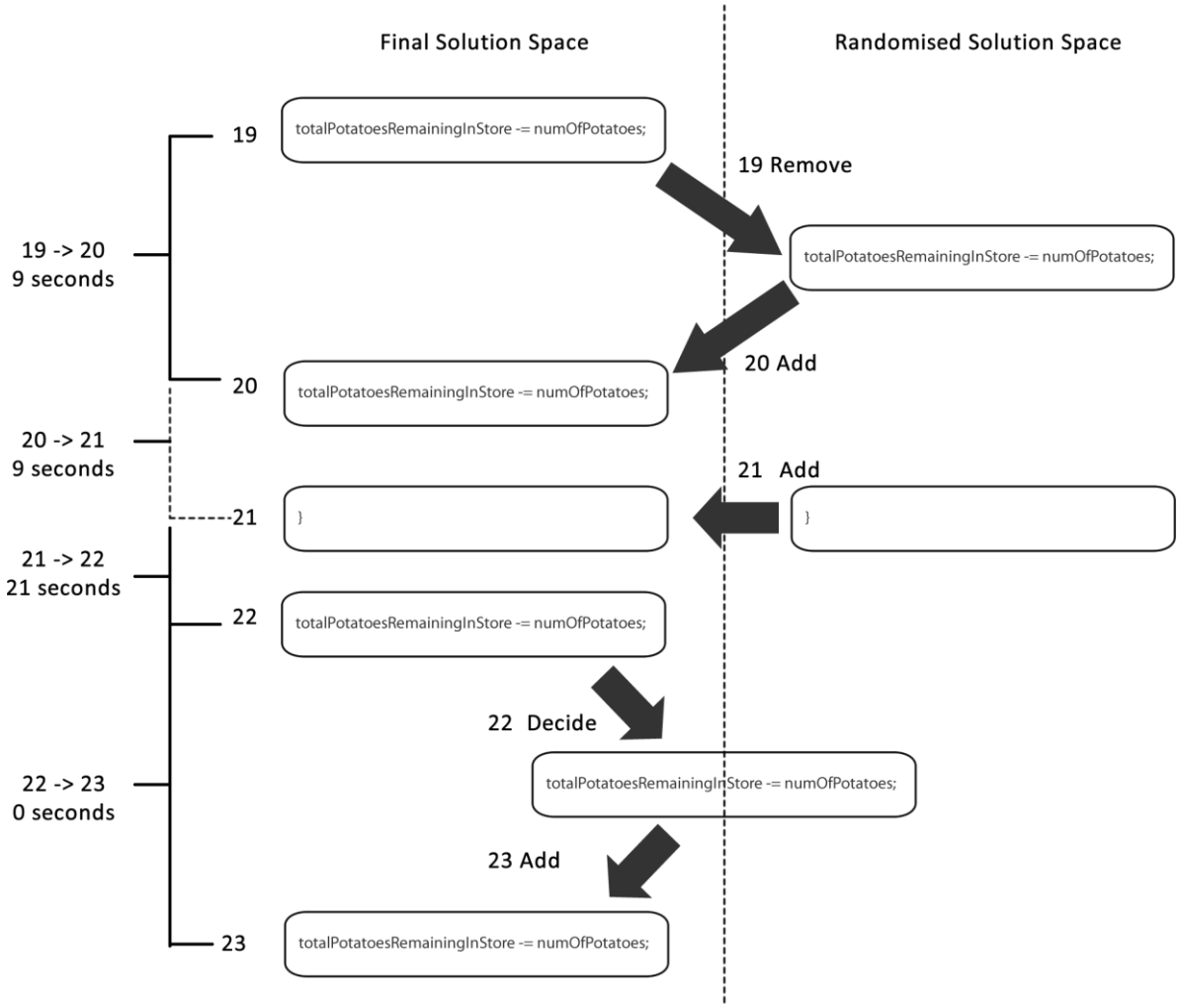


Figure 55: Diagram demonstrating P2's 19 to 23 movements involving the same piece used with different types of movements in close proximity of one another.

In Figure 55, it is apparent that P2's movements show unease at using the totalPotatoesRemainingInStore piece and of some uncertainty as to where it should be placed. Therefore, a series of movements of the same piece – particularly if they are consecutive or almost consecutive – suggests that the NP is struggling to determine the context of the piece and needs to 'decicde' where to place that piece. The exact issue with the piece requires audio transcript reading, though, as the movements could indicate there is a problem with the understanding of the underpinning programming concepts or the participant fails to understand the purpose behind the piece. Figure 56 demonstrates the way P2 decided to move the piece. This thesis argues that 'decide' and 'group' are novel movements for 2D Parson's problems that have manifested due to the amount of freedom of paper-based puzzles, and are related to one another. Deciding and grouping

manifested when the participant needed to determine the contextual relevance of the piece, and both movements are part of the workspace as a result.

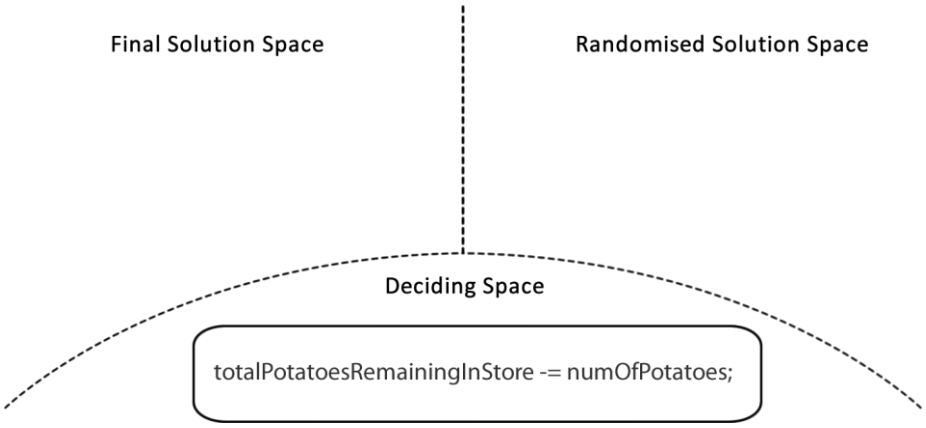


Figure 56: Diagram to represent P2's usage of 'Decide' movement with 'totalPotatoesRemainingInStore -= numOfPotatoes;' piece in CP1 – they held the piece in mid-air for approximately 10 seconds (5:13-5:24 on video footage).

P3 exhibited small amounts of grouping movements, but unlike P1, they often used grouping movements to order pieces that require thought before being placed into the final solution space. The grouping mechanism showed that the participant knew the pieces were related contextually (see Figure 57).

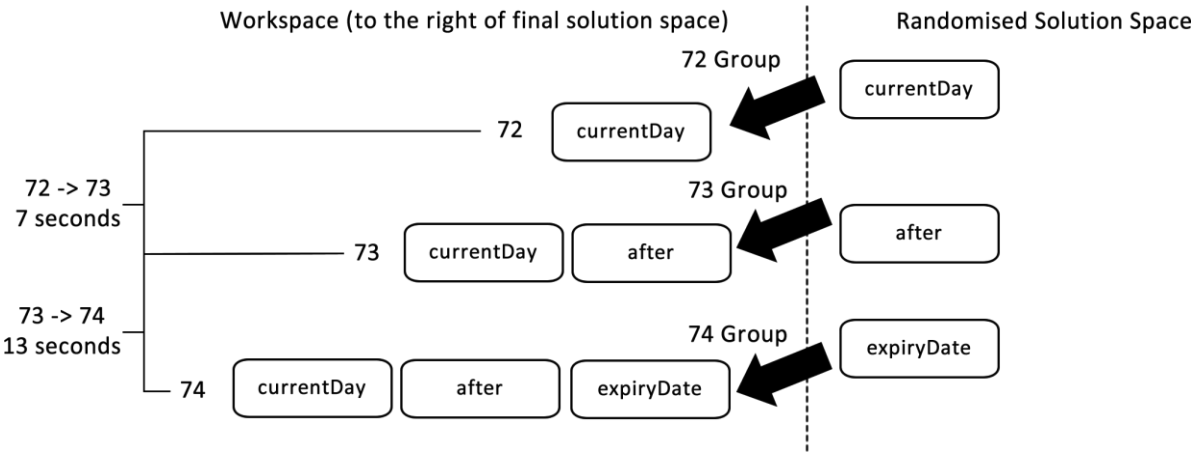


Figure 57: Diagram demonstrating P3's 72 to 74 movements involving grouping the pieces prior to placing them in the final solution.

P3 also demonstrated grouping of variable names related to a condition and kept these pieces to one side while creating the method (see Figure 58). Figure 58 also demonstrates the issues with identifying at what point the participant may be struggling with the class' structure – as shown, P3 forgot the parameter brackets for the isFresh method and had long 'completed' that line, likewise,

they had created a return statement and placed an if in a strange location – seemingly at the end of a close bracket of the method.

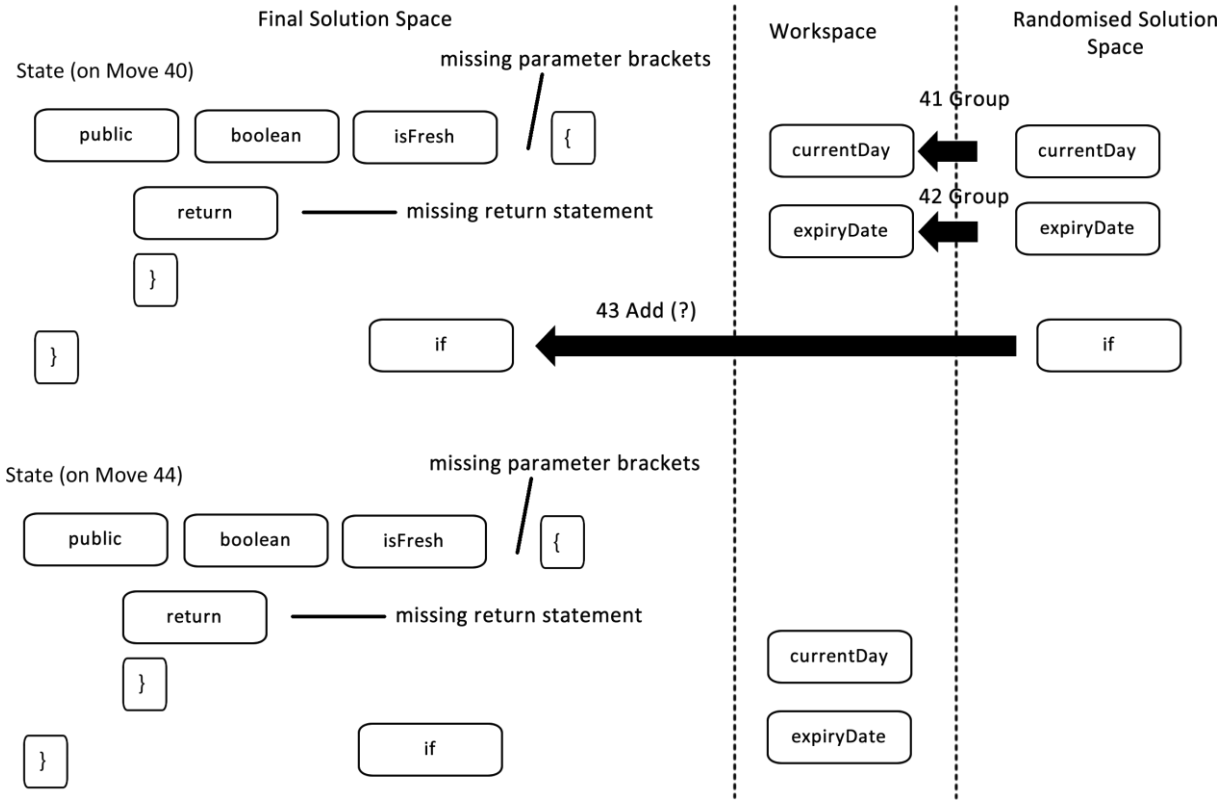


Figure 58: Diagram demonstrating P3's 39 to 44 movements involving grouping of two pieces, and perhaps a linked 'add' piece that was placed out of line but also separated from the group.

Sometimes additions were ambiguous due to the lack of physical barrier between the final solution space and workspace, but the if statement was later added to the correct position and the return was also completed – but at what point does a mistake remain a mistake and when does it transfer to being a sign of an underlying issue with programming? The answer remains undetermined as to the best way to distinguish mistakes from issues in understanding, however, if the mistake is still apparent in the final solution or the mistake had passed the participant checking their work a few times, it became increasingly likely that there was an underling issue present. Therefore, it is not enough to determine understanding based solely on the movements to the final solution space and that considerations about the workspace element and the need for audio transcripts needs to be considered to diagnose the understanding of a CS student.

P3 also demonstrated another movement that is of a similar ilk to 'remove', however, the piece was never placed in the final solution area and instead lingered in the workspace. The 'back' move is where the piece transfers from the workspace back to the randomised solution space when not deemed relevant by the participant. Remove and back movements have different connotations – the

final solution space rarely had pieces moving back to the randomised solution space or workspace and remove did show that the participant was unsure about the piece's position, however in the workspace the only positioning appears to be in groups – and sometimes pieces in the 'decide' category were separated and focused upon by the participant. Thus, the back movement connotation is that the participant has decided the piece is no longer relevant to their focus in the workspace – so their own context, rather than the final solution's context.

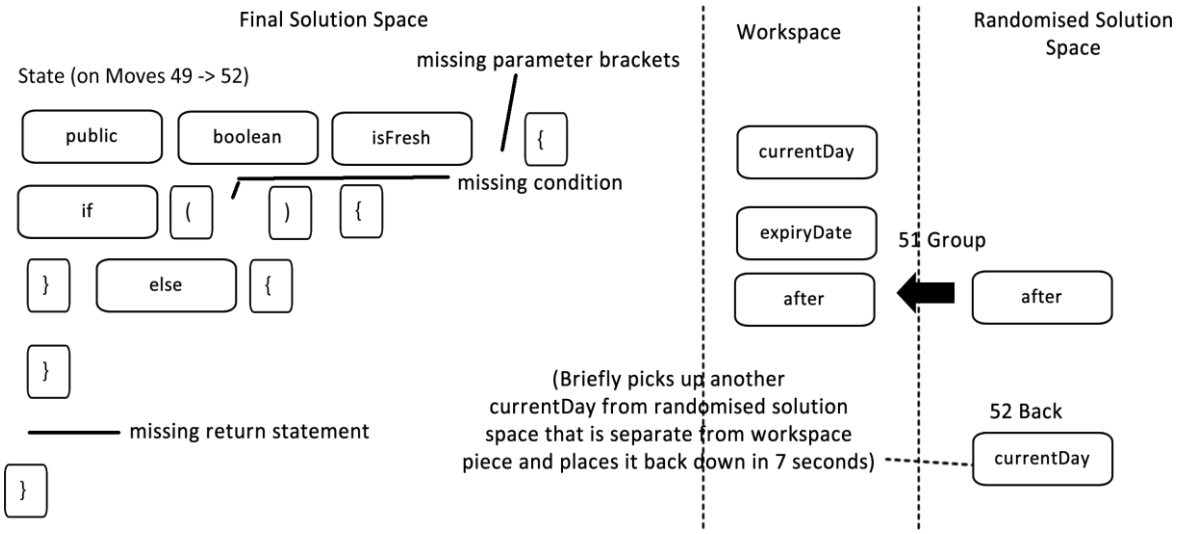


Figure 59: Diagram demonstrating P3's 51 to 52 movements involving adding 'after' to a pre-existing group defined 9 moves prior.

As exhibited in Figure 60, group movements were sometimes seen when removing elements from the final solution space. This indicates that participants did sometimes group based on similar context rather than just on their own interpretation of the pieces.

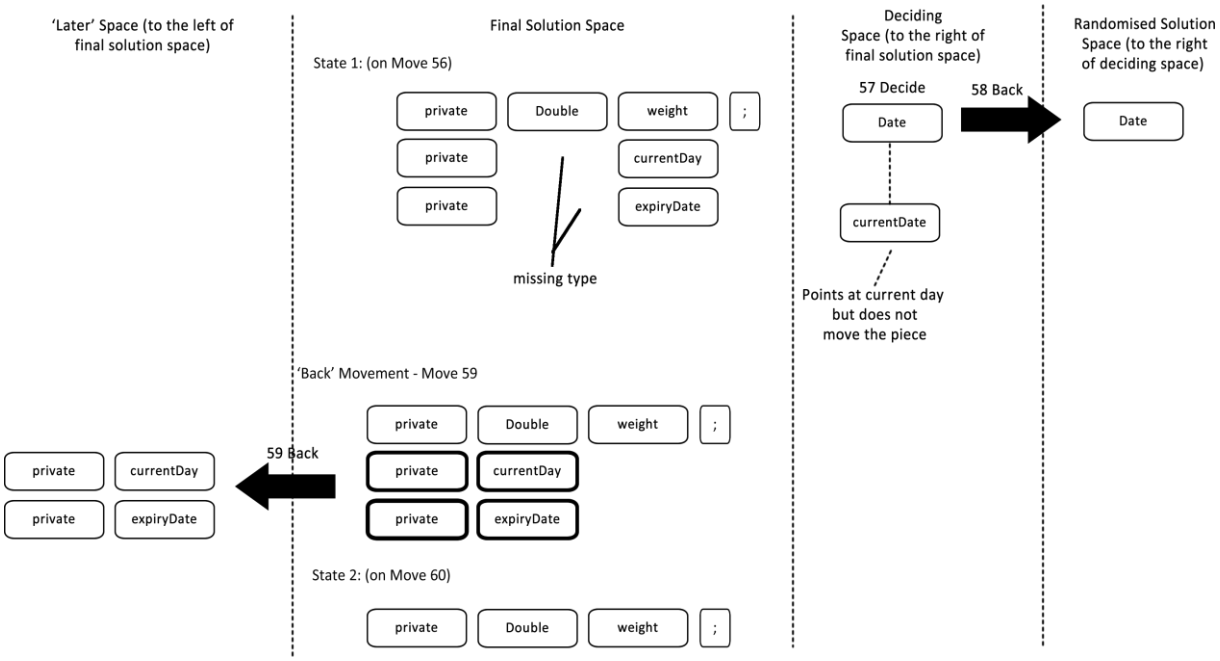


Figure 60: Diagram demonstrating P3's 57 to 59 movements involving decide and back movements.

Figure 61 and Figure 62 demonstrates that swapping movements did occur, when the participant had noticed an incorrect placement, but they were rarer than reported in previous works.

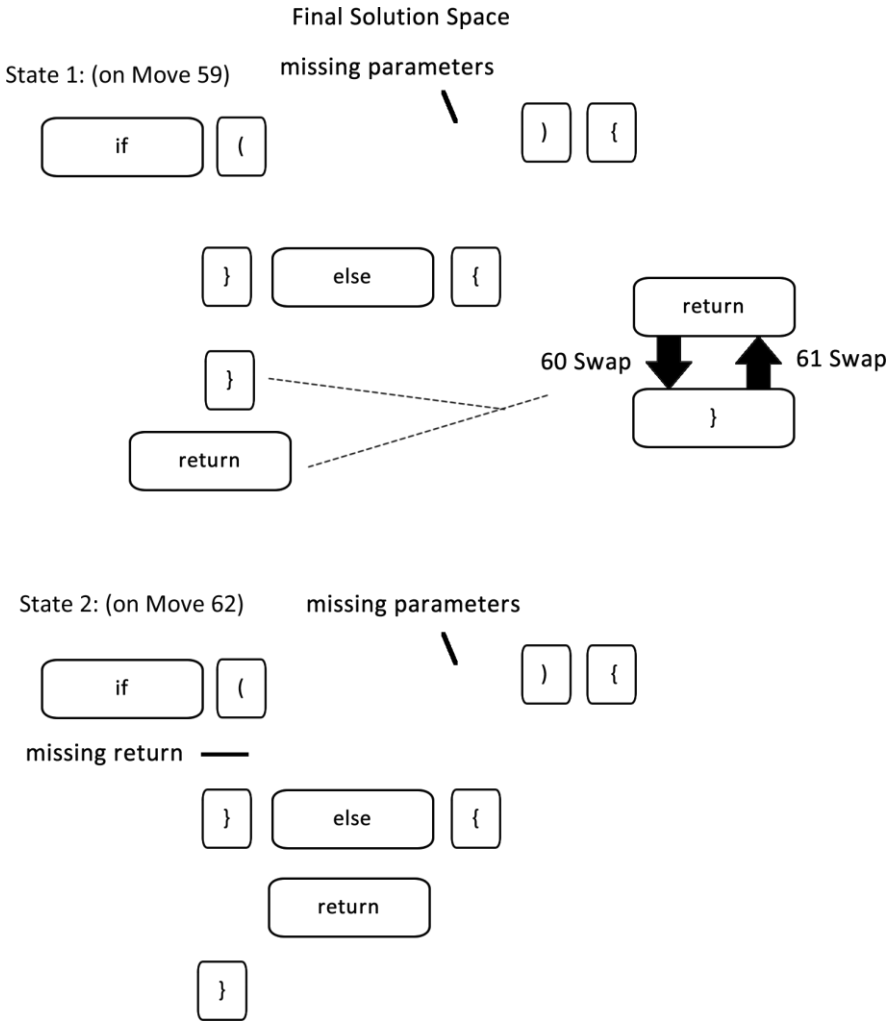


Figure 61: Diagram demonstrating P3's 60 to 61 movements involving swapping 'return' and '}' pieces.

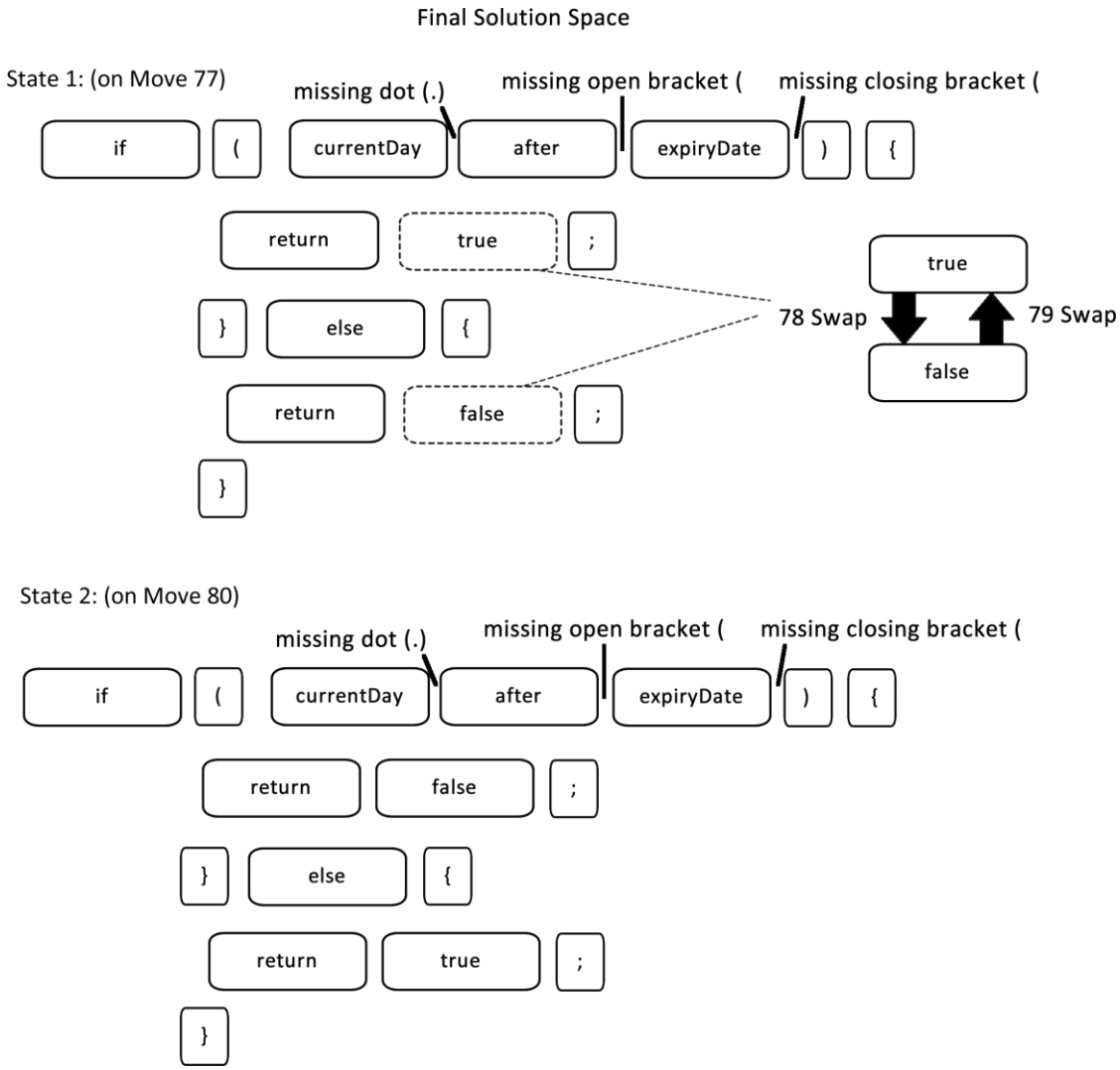


Figure 62: Diagram demonstrating P3's 78 to 79 movements involving swapping 'true' and 'false' pieces.

Ultimately, there were three new types of movements – decide, group and back that were not anticipated to be seen and there were no found records of previous works observing these movements from their participants.

5.3.2.4 Correct versus Incorrect Placements

The debate about what constitutes a correct versus an incorrect placement is debatable – for example, if a participant places a constructor prior to defining field declarations when the task asks for field declarations, is that really an incorrect placement? It could be that the participant is like P4, who needed to deduce what the constructor required and then define the data types for those field declarations prior to starting the rest of the class. While incorrect placements are easy to analyse in the final, submitted solution they are not easy to analyse during the process of creating new movements. The audio transcripts tend to reveal the reasoning behind movements, and the thought

processes accompanying them, but without the transcripts the intention behind the movement is lost. For the sake of discussion, the number of mistakes were quantified using the premise that a coder often completes full lines of code before moving onto the next line for CP2, and for CP1, it is assumed that the participant would complete a full method before moving onto the next part of the code.

The number of mistakes is shown in Figure 63 – a mistake differs from an 'issue' in that the mistake is remedied prior to the final solution submission. Figure 63 indicates that CP2 did encounter more mistakes than CP1 due to this analysis. While one might argue that CP2 should be assessed in the same way as CP1, it was often the case that participants would move between methods and fields and the class structure, and it would have been more difficult to judge what 'mistakes' occurred from incomplete methods due to the amount of moving between aspects that was detected in the approach. While the overall approach for both CP1 and CP2 is from top of the class to the bottom, when individual words are used the puzzles become more difficult to analyse for what is a 'mistake'.

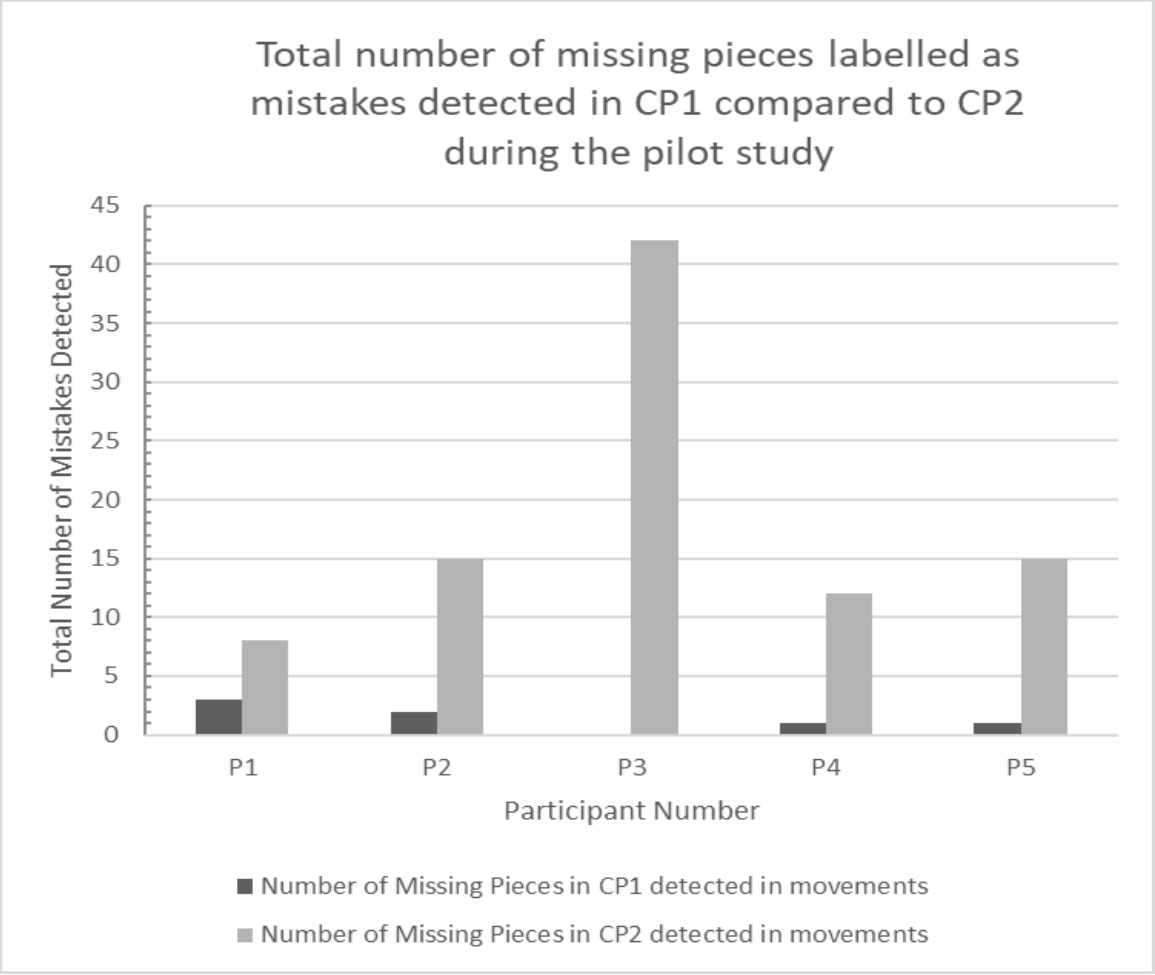


Figure 63: Bar chart illustrating the total number of missing pieces labelled as mistakes made by participants in CP1 compared to CP2.

Figure 64 demonstrates that very few incorrect placements – placements that were placed in the wrong place in the solution, were detected in either puzzle. This shows that 'mistakes' in Parson's 2D puzzles were more likely to be from missing out a piece rather than placing it in an incorrect context.

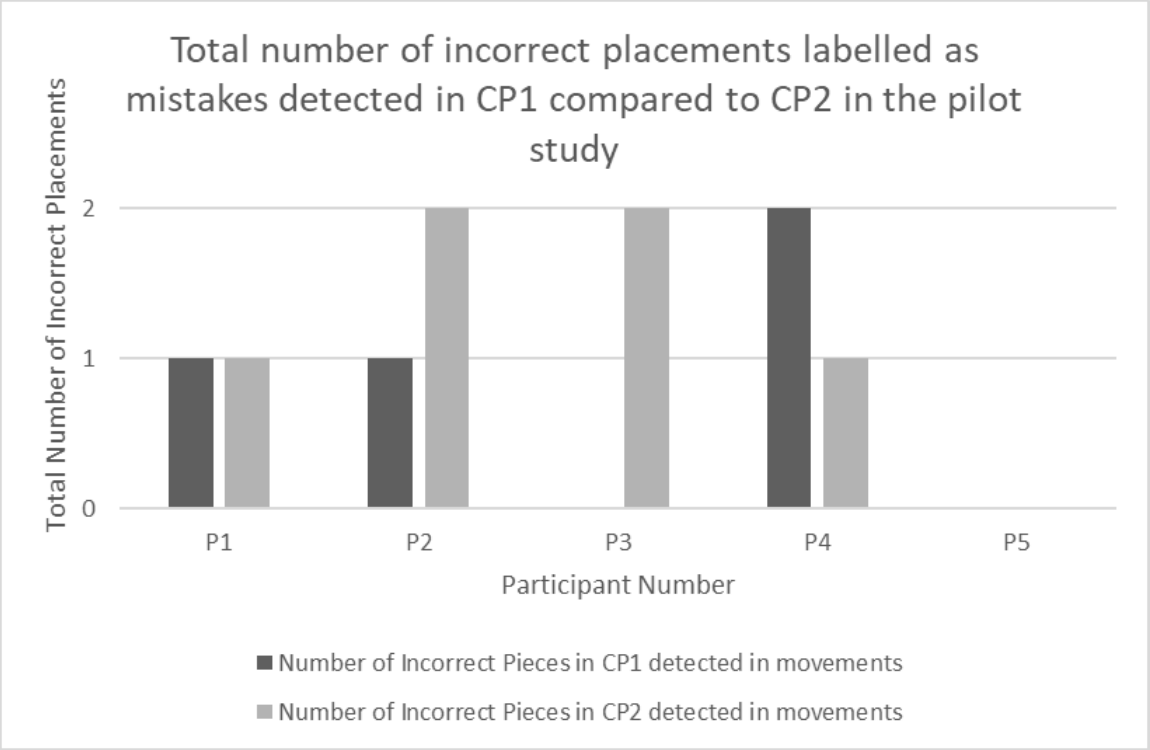


Figure 64: Bar chart illustrating the total number of incorrect placements made by participants in CP1 compared to CP2.

The analysis of mistakes made by participants show that they were all related to the punctuation of the class in CP1 – the closing off of various sections of the class – rather than the pieces associated to the class' logic (see Figure 65).

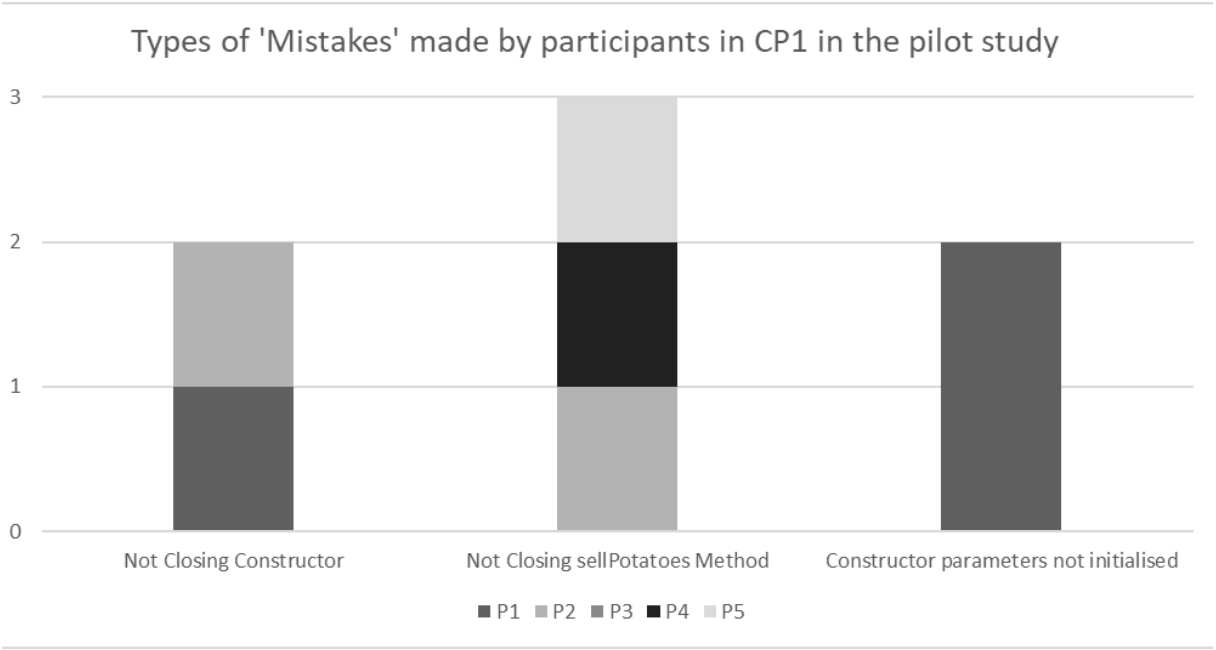


Figure 65: Stacked bar chart illustrating the types of mistakes made by participants in CP1.

The type of incorrect placements corrected are shown in Table 15.

Piece Description	Location	P1	P2	P3	P4	P5
(Field Declaration statements) \n numberOfPotatoesSold = 0;	Used prior to constructor signature	1	1		0	0
{instead of}	Closing else clause	0	0		1	0
return priceOfPotatoes * numberOfPotatoesSold;	Used in sellPotatoes instead of calculateSale	0	0		1	0
Total:		1	1		2	0

Table 15: Number and type of incorrect placements that were corrected prior to the final submission by participants in CP1

CP2 generated far more data on the types of mistakes encountered, with an artificial inflation of the field declaration where participants tended to switch between weight and expiry date without completing the lines fully first (see Figure 66 and Figure 67). The variable names seemed to cause difficulty – expiry date and current day are very similar in nature, and participants felt uneasy about the Date class according to their post-CP2 responses in the questionnaire (see Table 16).

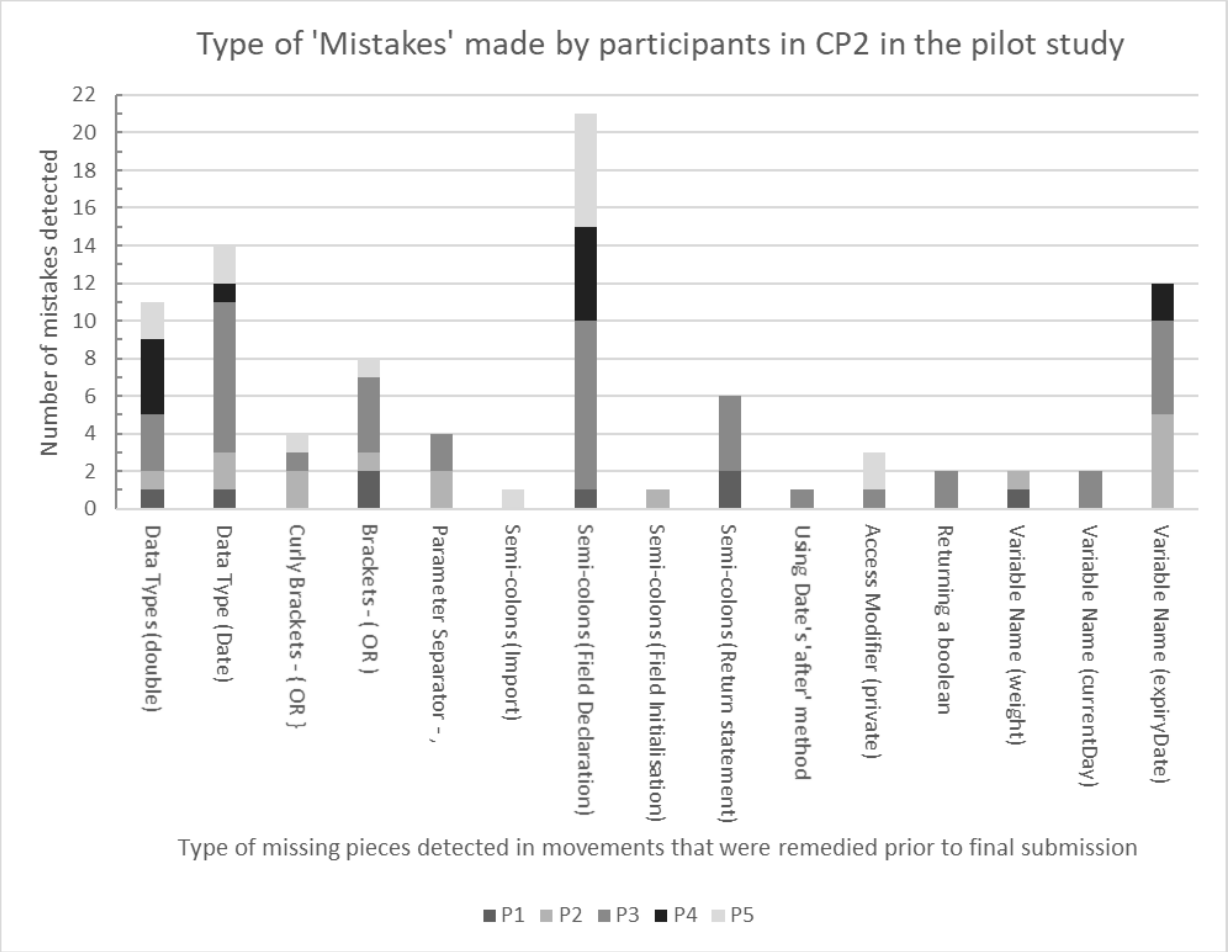


Figure 66: Stacked bar chart illustrating the types of mistakes made by participants in CP2.

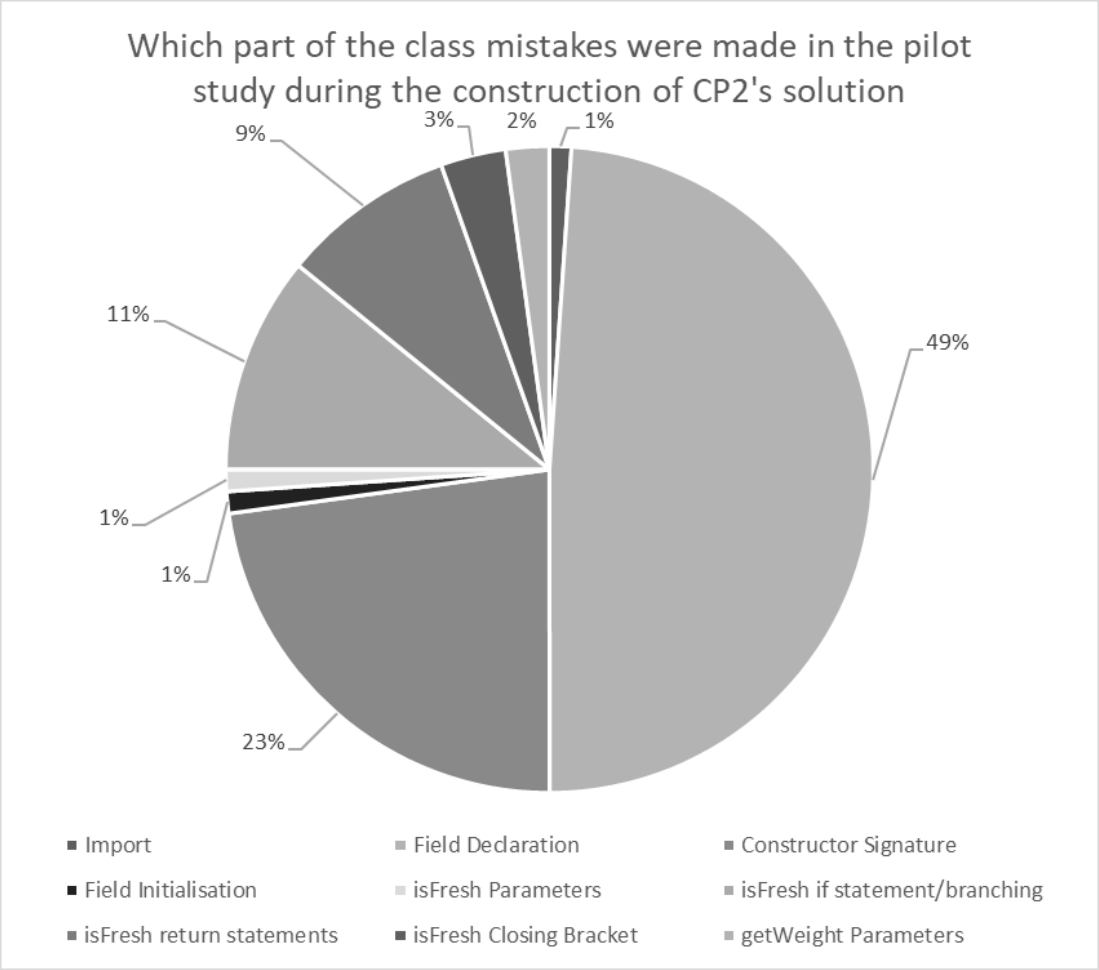


Figure 67: Pie chart that presents the locations the mistakes occurred in for CP2.

Piece Description	Location	P1	P2	P3	P4	P5
Date instead of boolean	isFresh return type	0	0	1	0	0
Return placed outside closing bracket of method	isFresh return	0	0	0	1	0
Method placed outside of class	isFresh after field initialisation	0	1	0	0	0
After called on class not on Date object	isFresh conditional statement	0	1	0	0	0
\true instead of false	isFresh return value logic	0	0	1	0	0
Semicolon in centre of line (e.g., return; true)	isFresh return statement	1	0	0	0	0
Total:		1	2	2	1	0

Table 16: Number and type of incorrect placements that were corrected prior to the final submission by participants in CP2

5.3.2.5 Participants' Approaches and Analysis of the Workspace

While the generalised process, once the participant had moved to the final solution space, did implicate that participants generally work from top to bottom of a class, P1 exhibited an addition

process prior to the established process that requires further investigation. The pilot study expected to see movements from the randomised solution space to the final solution space, and did not anticipate an intermediate area, dubbed the 'workspace', where participants use decide and grouping movements to determine what they think about the piece and how it relates to other pieces in the randomised solution.

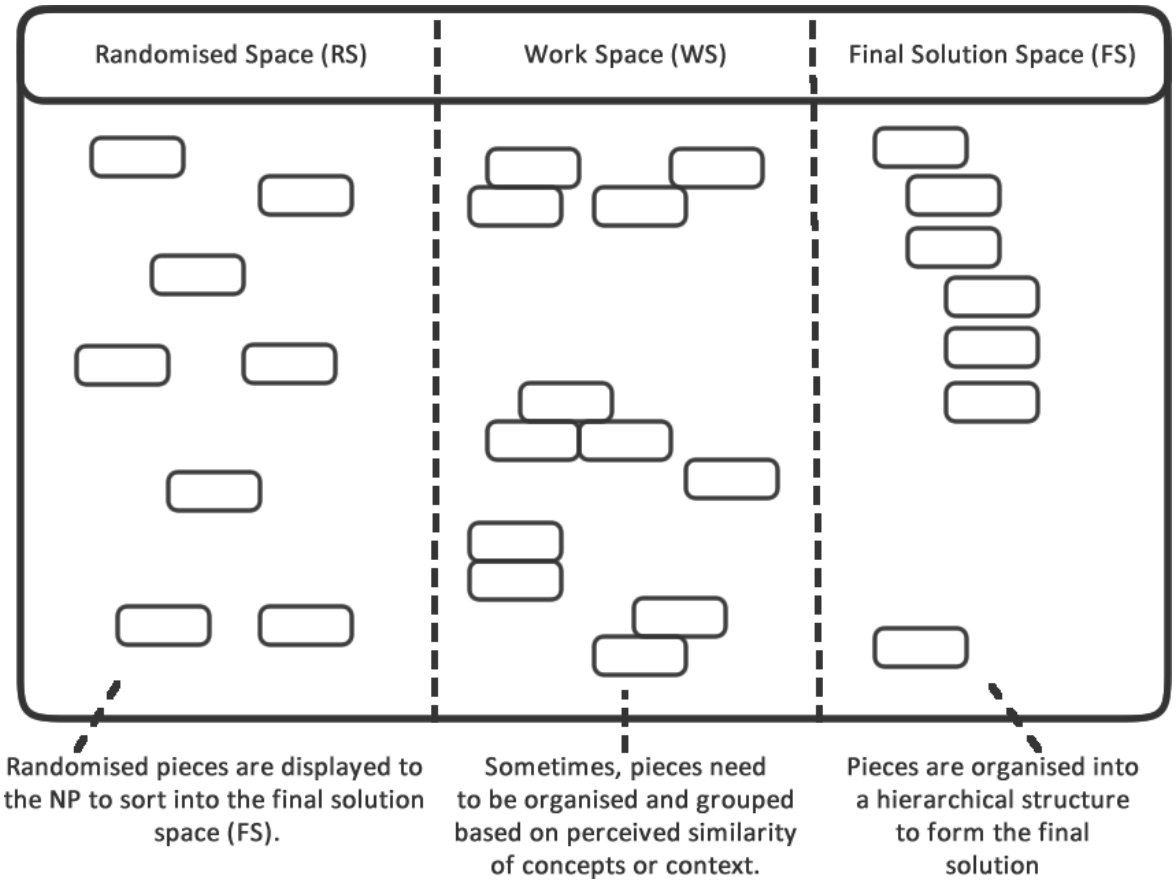


Figure 68: Pilot Study's observations on the interface design of 2D Parson's Problems.

The exact cause of the grouping mechanism was suggested by the audio transcripts of P1 to make the pieces more 'readable' to them, implying that the grouping mechanism, at its core, is about trying to interpret the meanings behind pieces prior to placing them into the final solution space and to make the participant aware of what pieces are available to them. However, the useful finding of this workspace is that the observer could also see more clearly what the participant's thoughts and intentions were about the piece. While it is difficult to get a clear still of the P1's CP1's grouping (due to the size of the pieces and the size of the physical table not allowing for much room to separate the groups), CP2's groups were taken from a still shot of the video recording and analysed to see

whether the groups are distinct enough for a clustering algorithm to anticipate the correct number of clusters based on the participant's intention in the audio file.

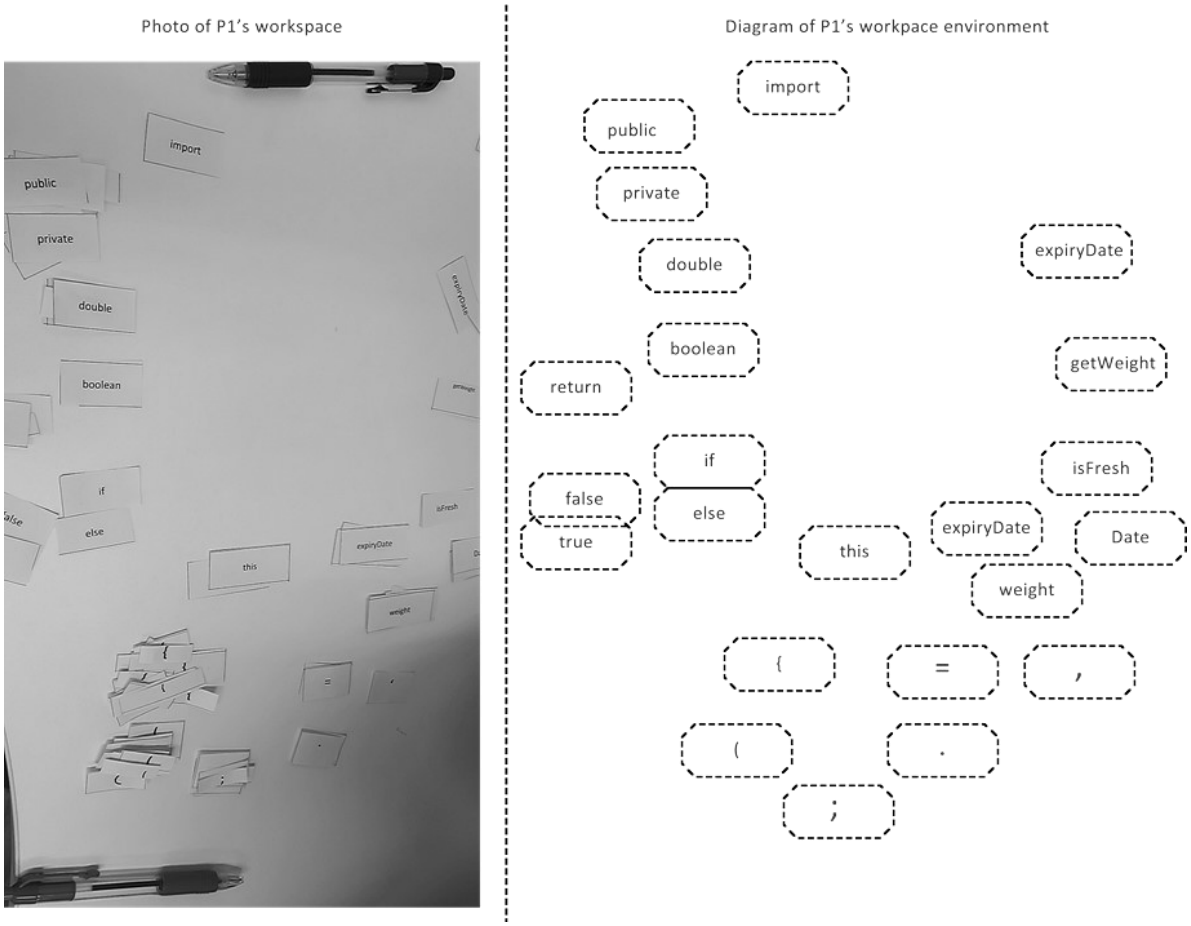


Figure 69: Photo still of P1's workspace for CP2 (left) with a generated diagram of the groupings of the workspace indicated by P1's audio transcript (right)

The still of the photo was used to generate co-ordinates via analysing the photo in Adobe Fireworks and transferring those co-ordinates to a Python program to generate a scatter graph to simulate the co-ordinates of the photo. Figure 69 highlights the intended groups that the participant generated as a consequence of incorporating the workspace, and as established, there were 23 groups with one group over-lapping each other (false, true). However, from looking at the groups as an observer there seems to be a pattern or a selection of groups that could indicate the candidate was also grouping groups together based on similarity – for example, the punctuation could be determined to be clustered together, alongside the variables which have been put next to the 'this' implying that the participant knows that these aspects are related. The conditions have also been placed together, and the access modifiers and return types of the methods have been placed at the top together. The analysis of a workspace, therefore, yields interesting implications for observing the participant's

sphere of learning – from the workspace, we can see how their mind maps the pieces to concepts and how the pieces relate to one another.

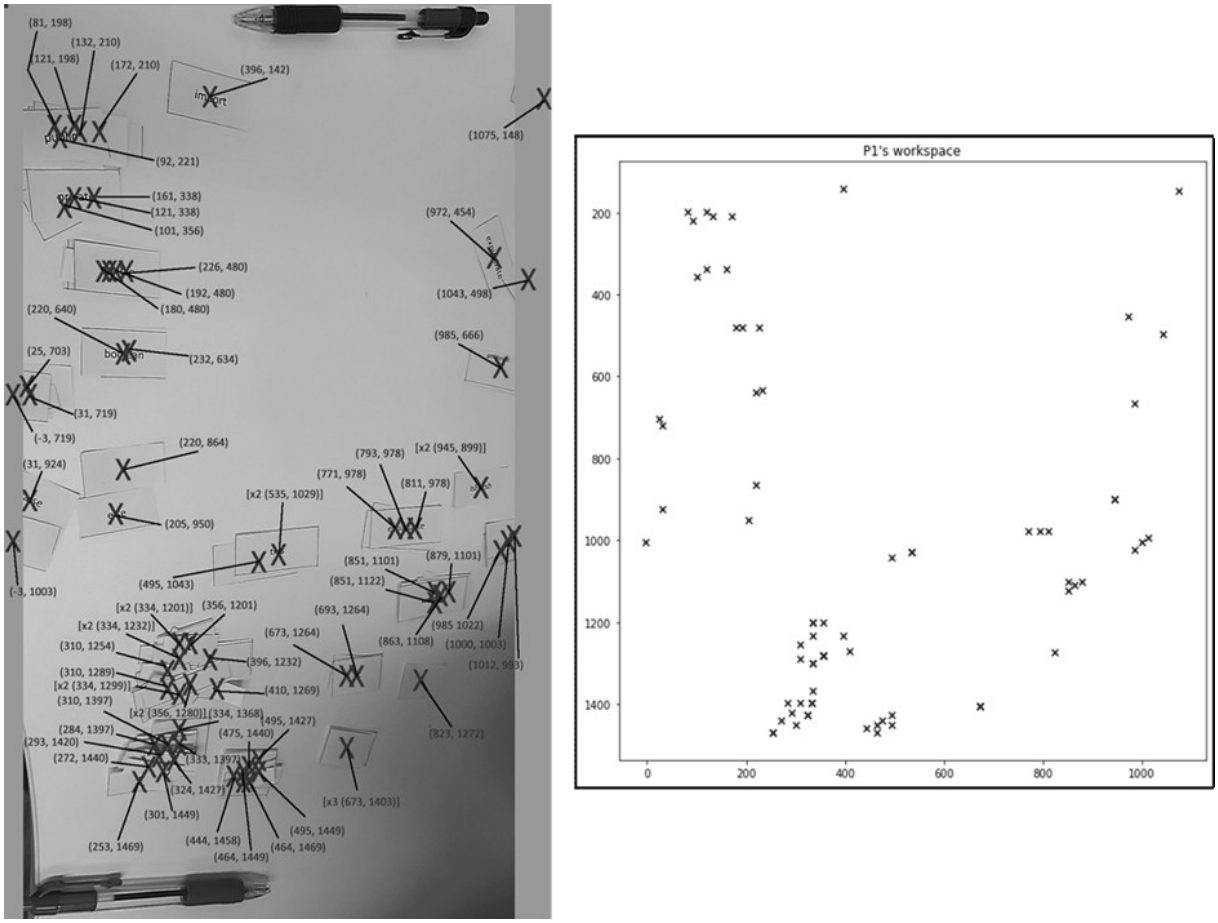


Figure 70: Co-ordinates generated from the photo still of the workspace for P1 assuming top left is (0, 0) (left) and the scatter graph generated from the co-ordinates (right).

There were some technical difficulties with translating the image to a co-ordinates based image, as Adobe Fireworks assumes the (0,0) co-ordinate is the top left rather than the bottom left, therefore the chart had to be inverted but the consequent silhouette diagrams would not invert correctly so there is a small discrepancy in the produced chart images on the right hand side of Figure 74, Figure 75, Figure 76, Figure 77 and Figure 78.

To determine if a machine learning algorithm could correctly identify the number of groups that the participants described in their audio transcripts, K-means and GMM clustering algorithms were used using Python's sklearn package, alongside their silhouette scores.

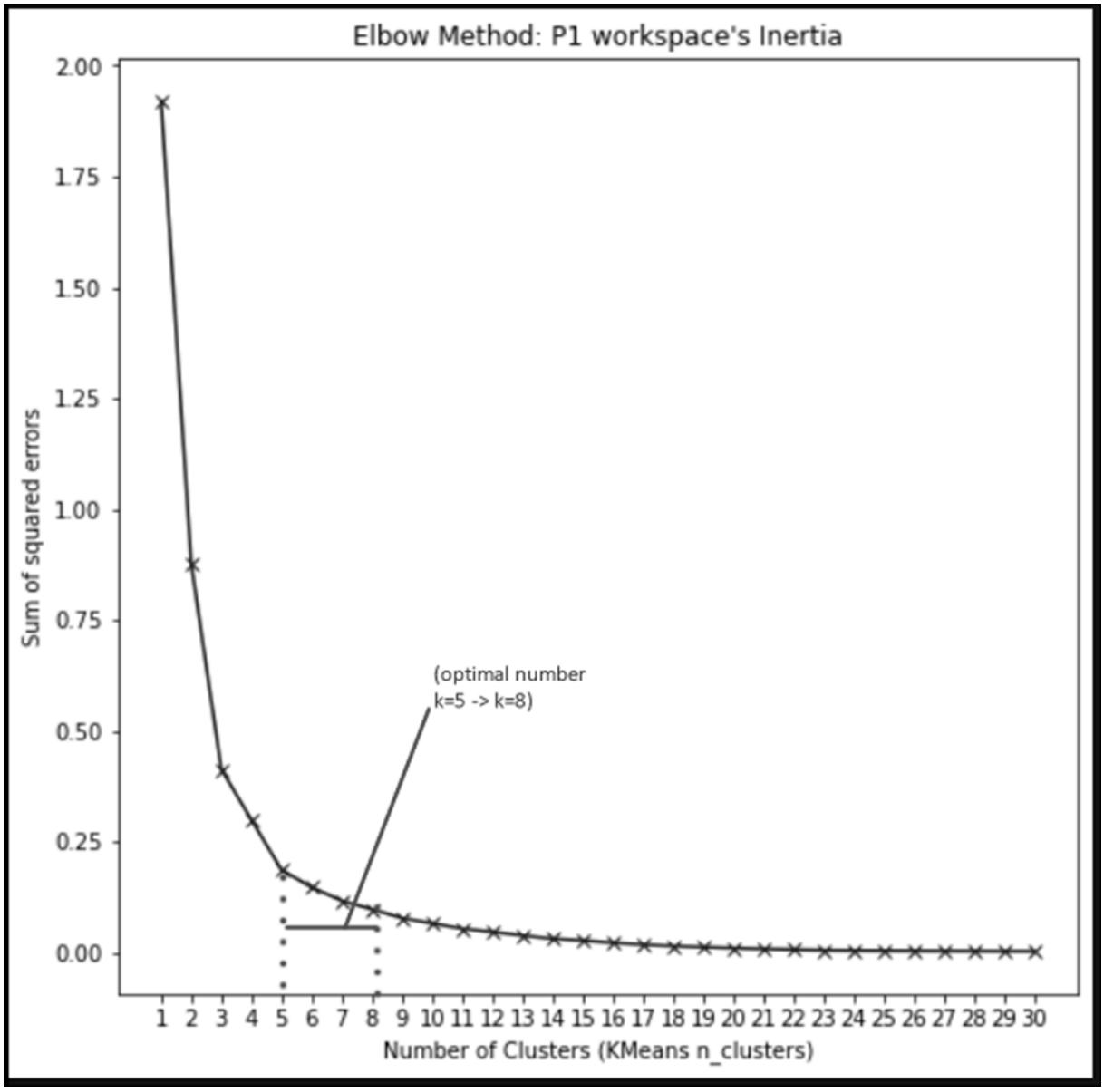


Figure 71: P1's workspace for CP2's inertia using the elbow method.

The elbow method, which is a standard determination algorithm for unsupervised machine learning algorithms, was used to calculate the optimum number. While open to interpretation, Figure 71 suggests that the optimum number is between 5 and 8 when using K-Means with n clusters.

The EM algorithm was used to calculate a log score for an alternate form of clustering, GM, however despite the algorithm running for 30 clusters, it did not seem to produce a low enough log score to suggest that this would be an accurate algorithm (see Figure 72).

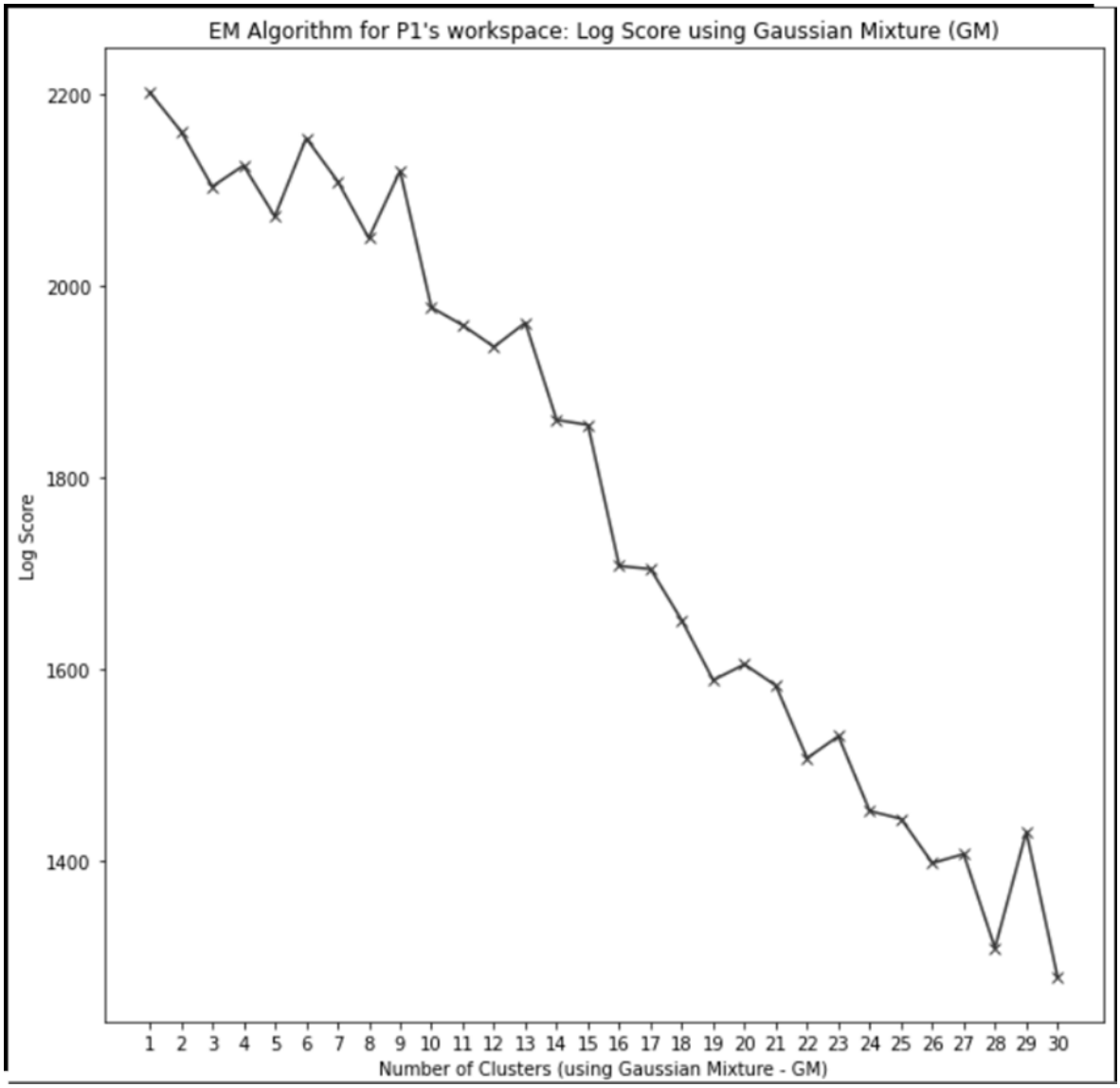


Figure 72: P1's workspace for CP2's optimal number of clusters calculated using Python's EM algorithm using a Gaussian Mixture (GM) to generate a log score.

A silhouette score was produced to calculate the optimum number of clusters using K-means as a result of GM not showing suitability for the diagram. Not surprisingly, the silhouette score shown in Figure 73 was far more accurate than the elbow or EM algorithm as the silhouette score calculates how well each cluster has points residing within that cluster. The accuracy of silhouette score is an indicator that it would be possible, on a user interface, to determine the way in which a person is utilising the workspace to a degree of accuracy although it could be argued that the formation of groups and sub-groups may be difficult to classify automatically unless the participant somehow labels those groups explicitly.

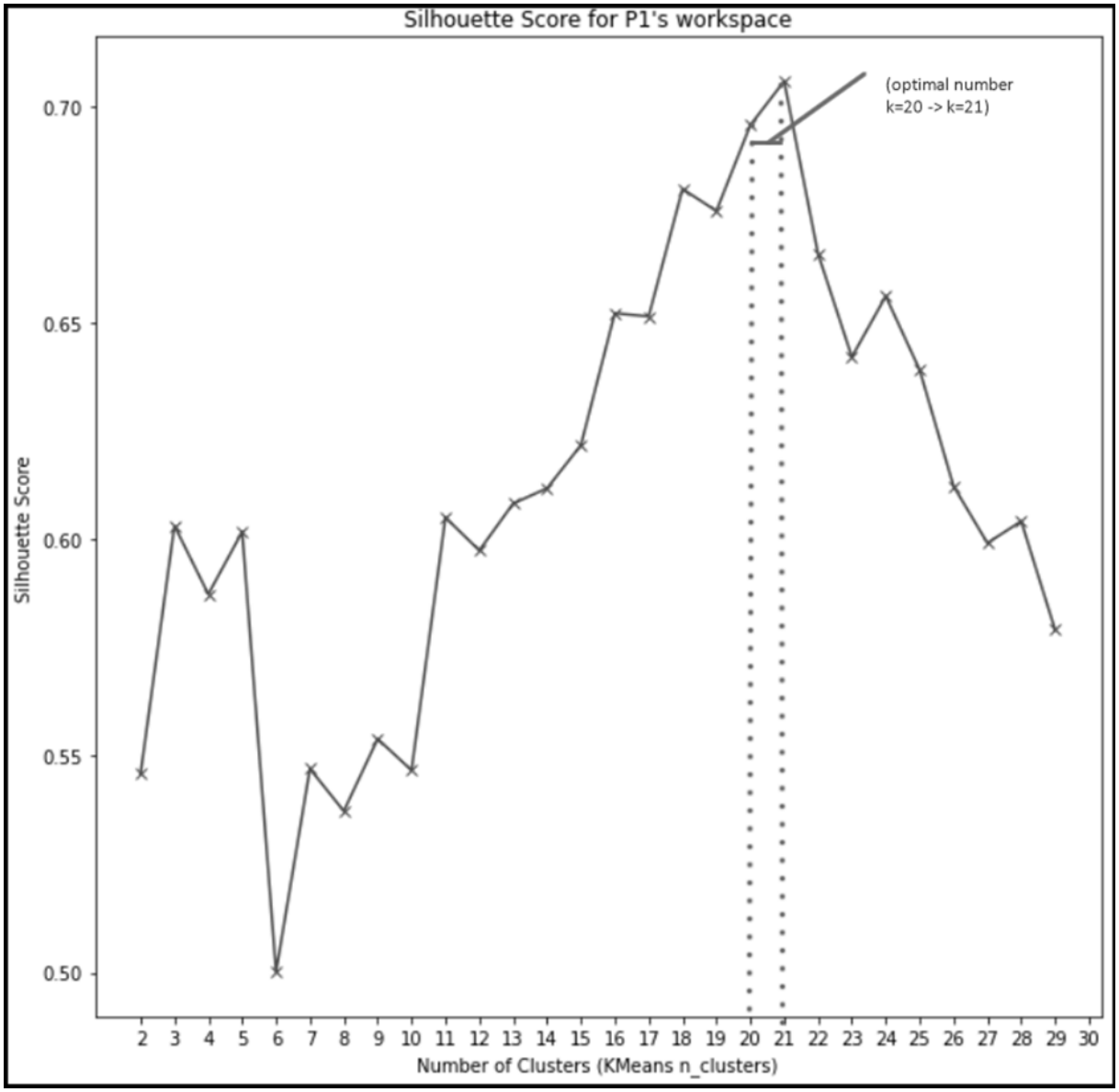


Figure 73: P1's workspace for CP2's optimal number of clusters calculated using K-Means' Silhouette score.

To visualise the elbow method's and silhouette score's suggested clusters, a series of silhouette diagrams (see Figure 74, Figure 75, Figure 76 and Figure 77) were produced – in this case, 'optimum' number is used in the context of trying to find a close match to the participant's identified number of groups.

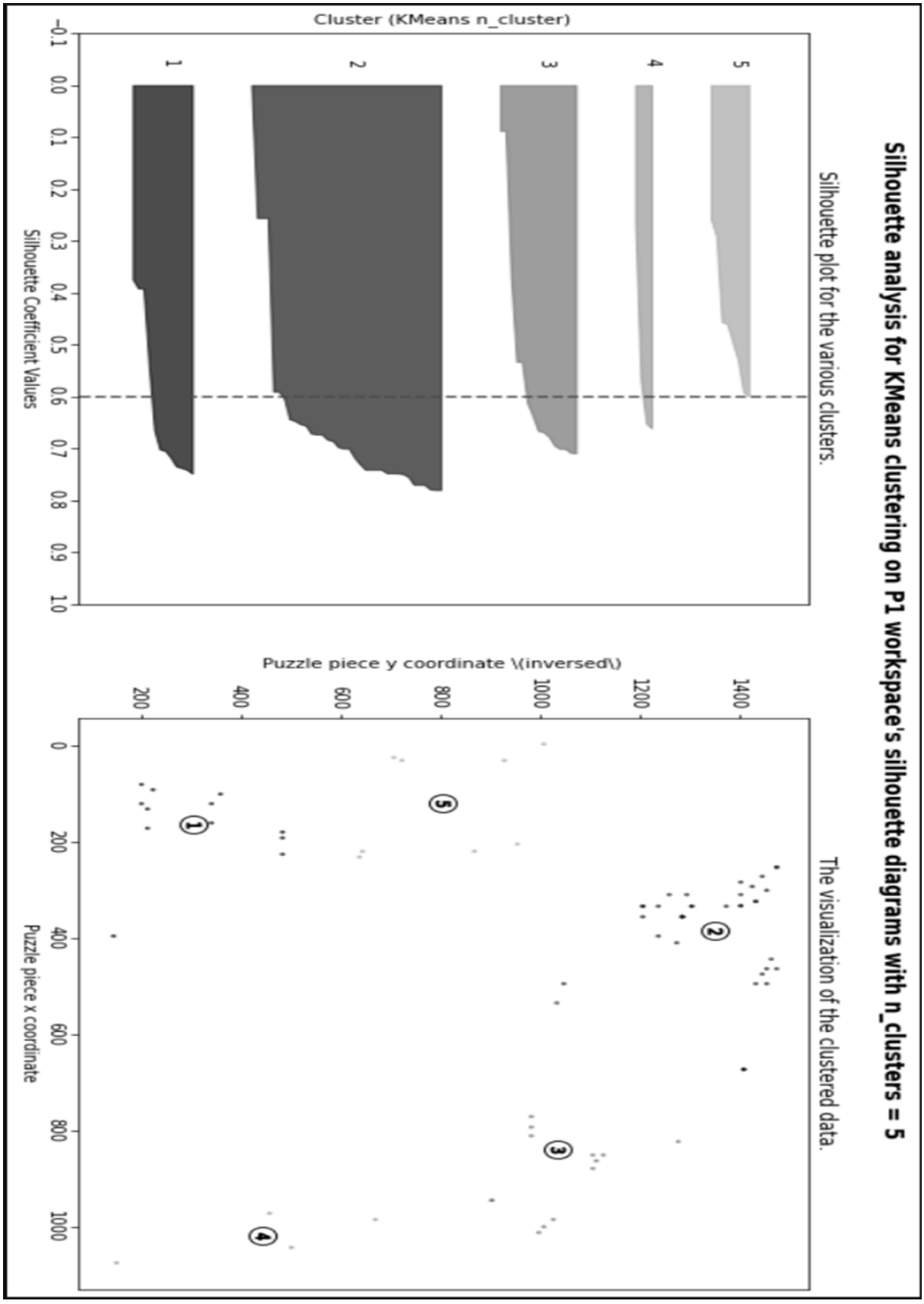


Figure 74: Silhouette diagram of one of the 'optimum' number of KMeans clusters (n=5) determined by the elbow method (generated using Python's sklearn package)

Figure 75: Silhouette diagram of one of the 'optimum' number of KMeans clusters (n=6) determined by the elbow method (generated using Python's sklearn package)

Figure 76: Silhouette diagram of one of the 'optimum' numbers of KMeans clusters (n=7) determined by the elbow method (generated using Python's sklearn package)

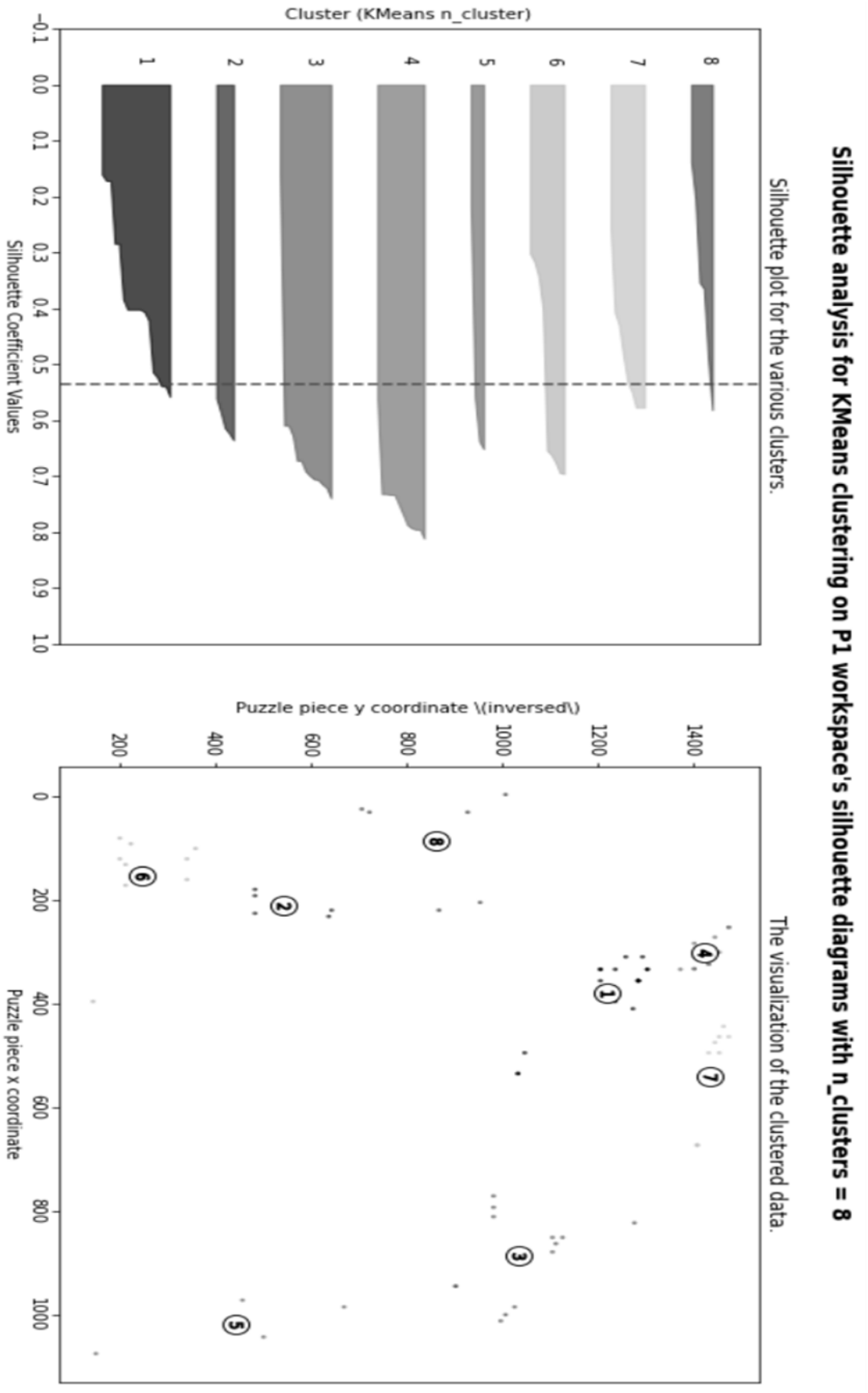


Figure 77: Silhouette diagram of one of the 'optimum' numbers of KMeans clusters (n=8) determined by the elbow method (generated using Python's sklearn package)

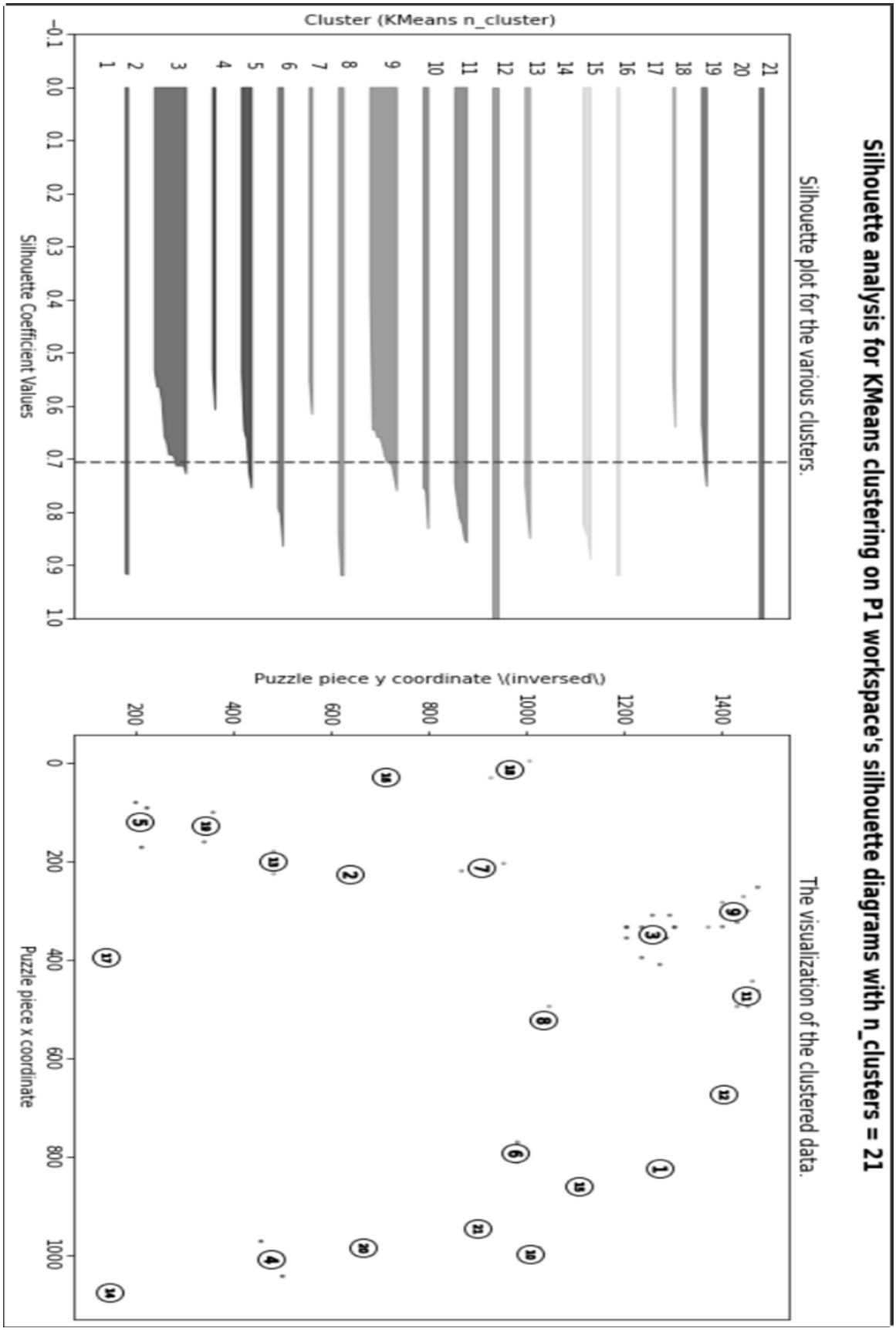


Figure 78: Silhouette diagram of the 'optimum' numbers of KMeans clusters (n=21) determined by the silhouette score (generated using Python's sklearn package)

As suggested, 21 is the closest to the actual intentions of the participant, but the elbow method did produce results that could be classified as implicit sub-groups based on the photo.

5.3.3 Analysis of the Submitted Solutions

All participants produced fairly robust solutions that would work, implying that the level of NP in the pilot study was advanced as these results were not emulated in the secondary or tertiary study final solutions. P1 demonstrated that the brackets were forgotten for the isFresh method, but they were used correctly for the getter method and constructor implying it was a genuine mistake. As noted, many participants chose to indent their solutions – the indentation shows that the participants believed this to be important, despite the struggles of having to individually indent each piece. Indentation in the case of P5 suggested what the erroneous class closing bracket was intended for and helps both the constructor of the code and the observer to read the intentions of the NP without them necessarily explaining the indentation pattern (see Figure 83).

Figure 79: Representations of P1's submitted solutions for CP1 (left) and CP2 (right)

Figure 80: Representations of P2's submitted solutions for CP1 (left) and CP2 (right)

Figure 81: Representation of P3's submitted solution for CP2, CP1 was unobtainable due to video footage corruption.

Figure 82: Representations of P4's submitted solutions for CP1 (left) and CP2 (right)

Figure 83: Representations of P5's submitted solutions for CP1 (left) and CP2 (right)

5.3.4 Post-Puzzle Questionnaires

P3 found CP1 difficult and based on the audio recordings it is likely they would not have completed the solution adequately without assistance, however P3 was more confident in their CP2 solution than their CP1 solution, despite the numerous errors made in the class' construction. In contrast, all other participants became less confident that their solution would work apart from P5.

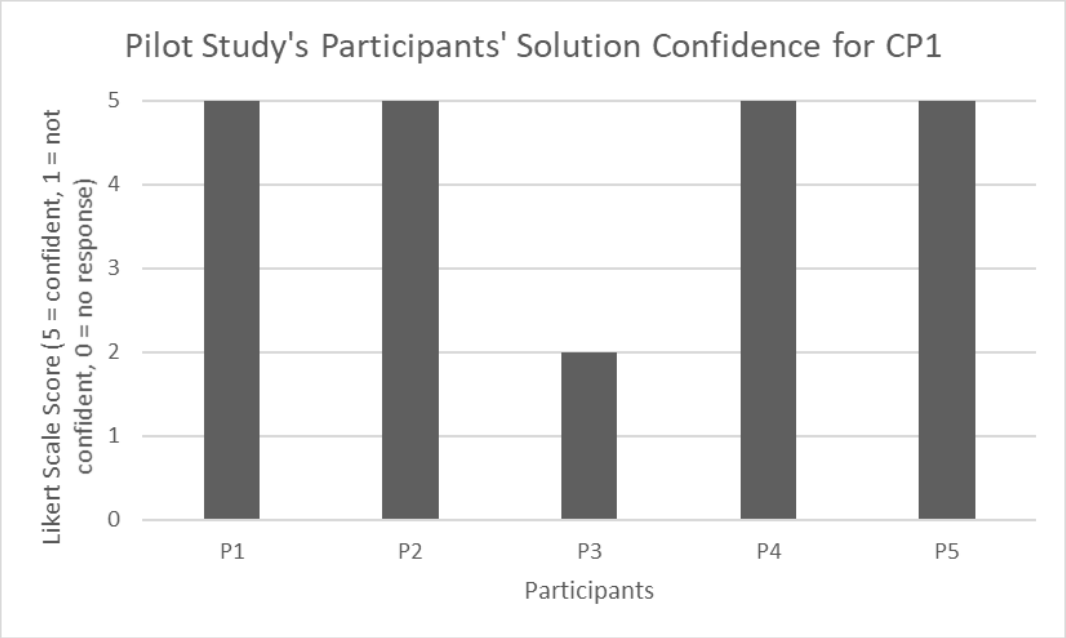


Figure 84: Pilot Study's participants' confidence in their submitted solution for CP1 based on a 5-point Likert Scale – 5 indicates they believed their solution worked without any errors, 1 indicates they would not know if the solution worked.

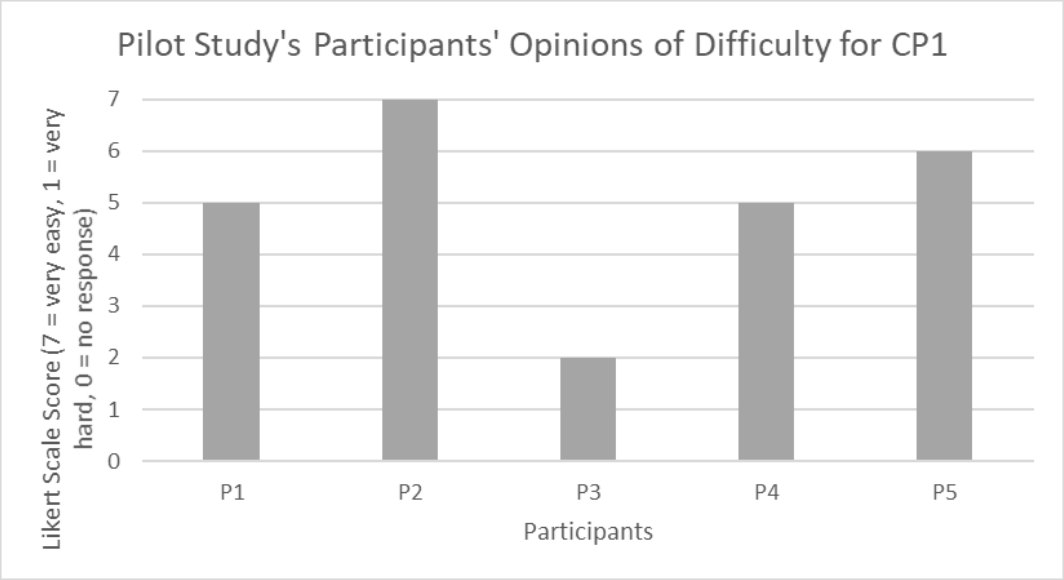


Figure 85: Pilot Study's participants' opinions of difficulty for CP1 based on a 7-point Likert Scale – 7 indicates very easy, 1 indicates very hard.

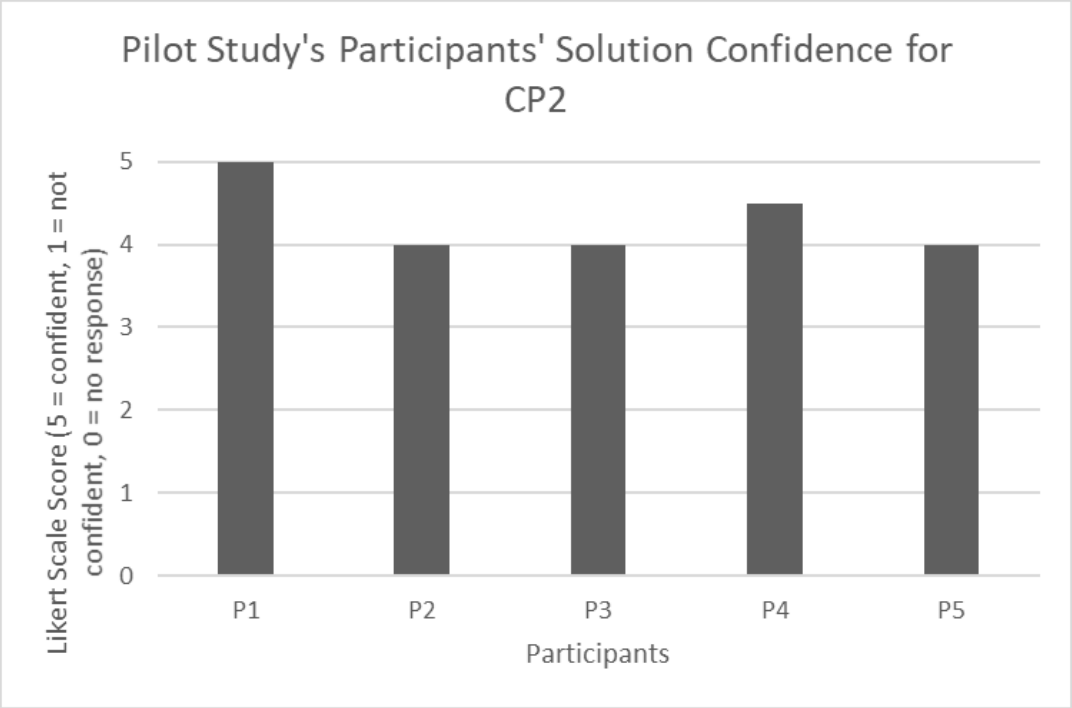


Figure 86: Pilot Study's participants' confidence in their submitted solution for CP2 based on a 5-point Likert Scale – 5 indicates they believed their solution worked without any errors, 1 indicates they would not know if the solution worked.

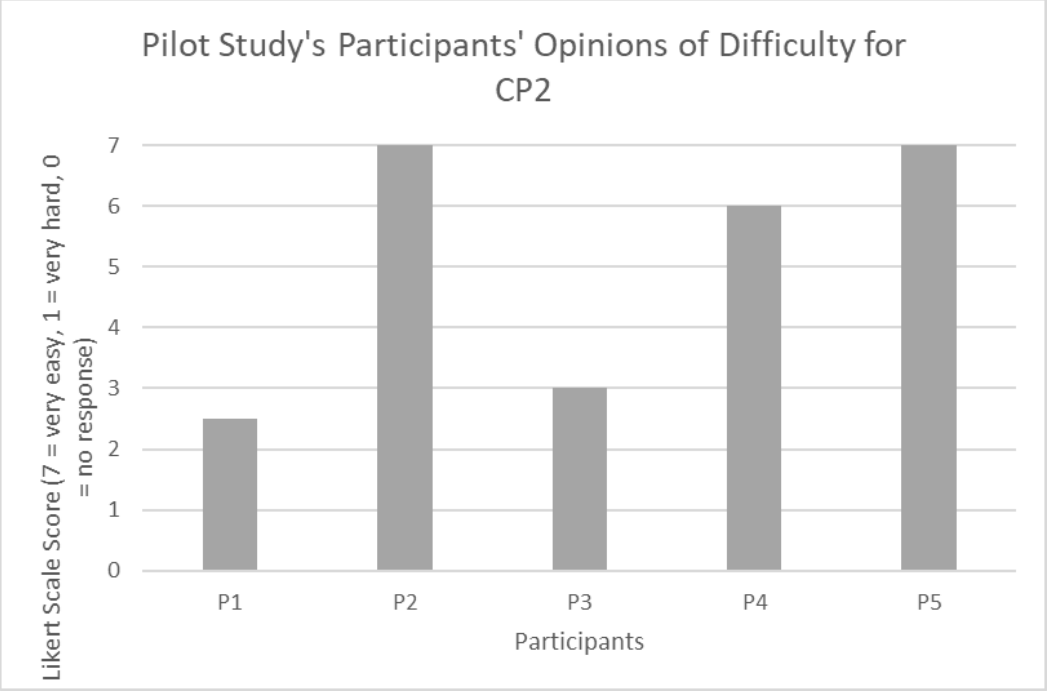


Figure 87: Pilot Study's participants' opinions of difficulty for CP2 based on a 7-point Likert Scale – 7 indicates very easy, 1 indicates very hard.

There was no significant correlation between the answers of the post-puzzle questionnaires and the number of movements, or the solution confidence. However, there was a small correlation between

the difficulty and the number of issues encountered. This correlation is weak, and due to the small sample size is likely not of any significance, but it could suggest that participants did realise that issues had transpired and that this had reduced the level of confidence they had in their solution.

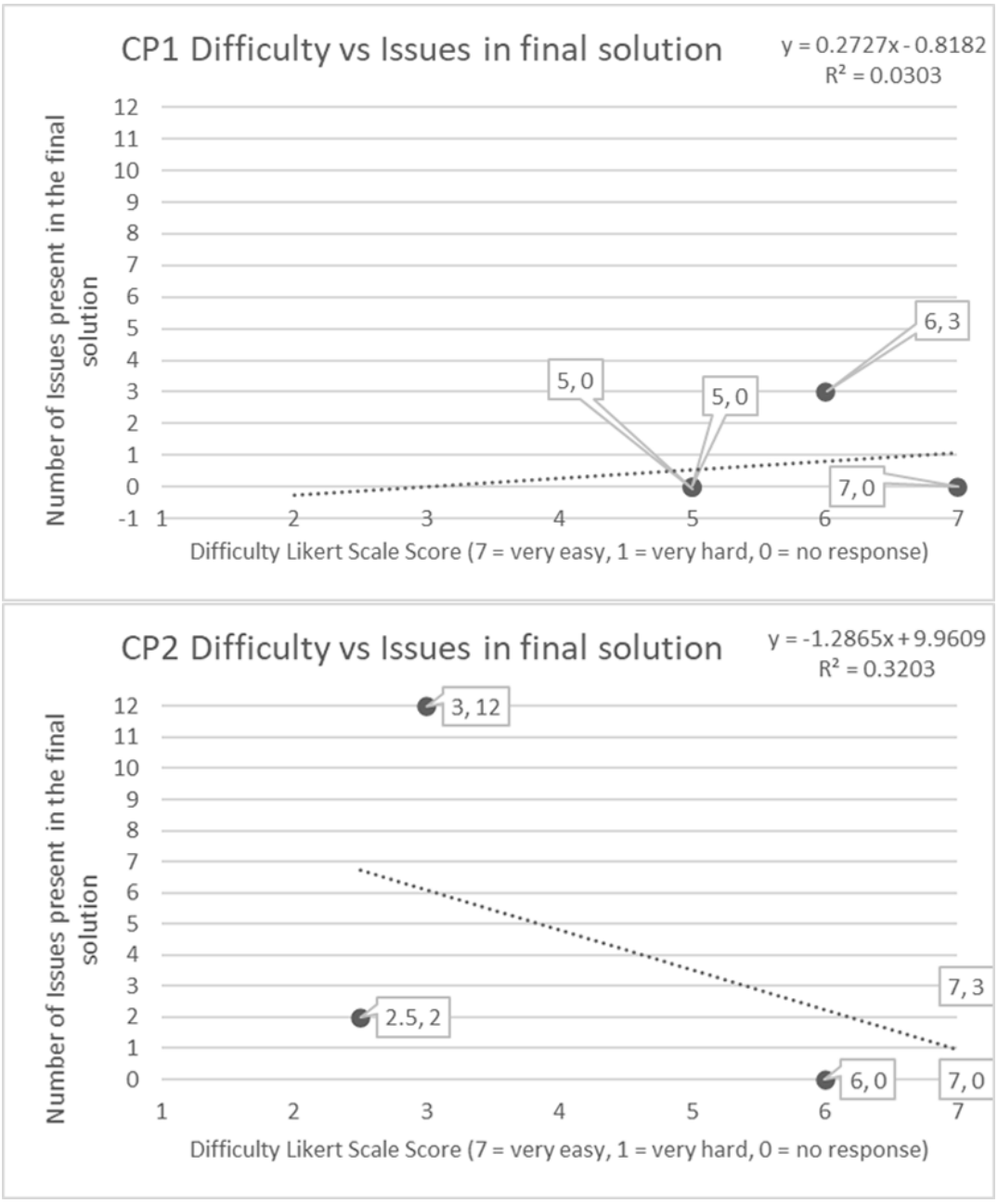


Figure 88: Pilot Study's participants' estimation of difficulty compared to the number of issues with their submitted solution in CP1 (top) and CP2 (bottom).

5.3.5 Analysis of Participants' Speech

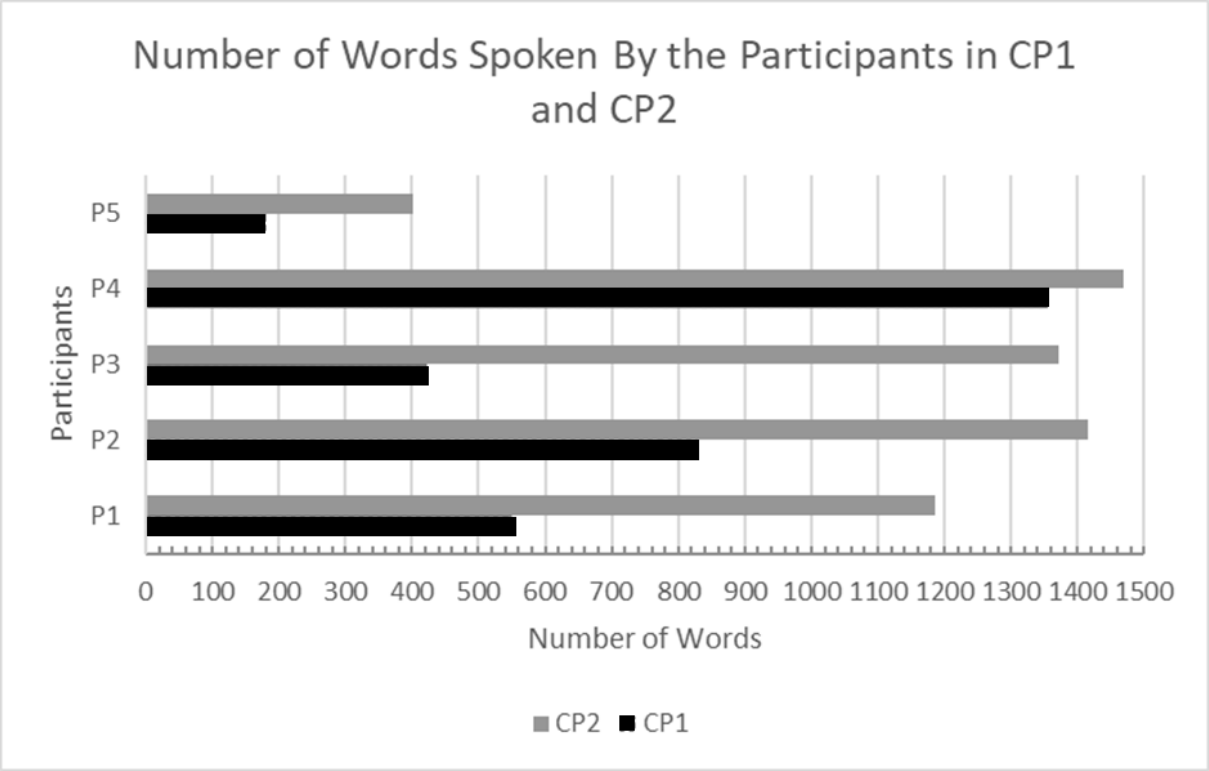


Figure 89: Clustered Bar charts comparing the number of words spoken by the participants during the CP1 and CP2 (CP1: range = 182-1356, M = 667.6, SD = 449.88|| CP2: range=401-1470, M = 1169, SD = 442.31)

Participants tended to speak more in CP2 than in CP1, implying that the audio transcripts did detect more information behind their thought processes. This is hardly surprising, as Parson's 2D problems help to clarify the line order, whereas CP2 required the participants to construct the code segment from scratch.

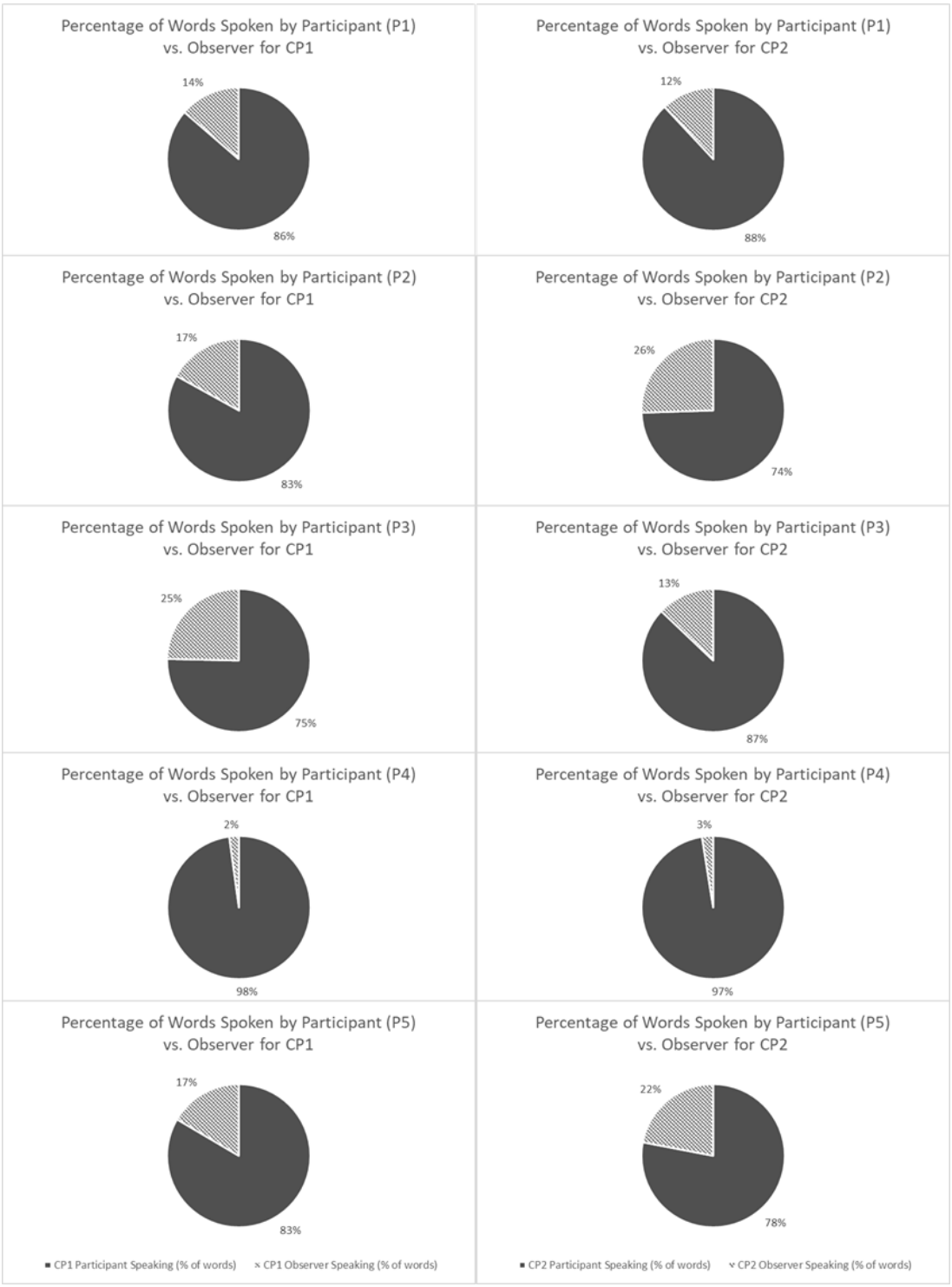


Figure 90: Pie charts showing the percentage of words spoken in the audio recordings were participants' words verses those of the observer (right chart) (CP1: range = 75.36%-97.84%, M = 85.21%, SD = 8.15%|| CP2: range = 74.49%-97.48%, M = 85.03%, SD = 9.07%)

The average participant spoke for 85% of the time during CP1 and CP2's experiment, with all participants speaking more during CP2 than CP1. While P2's feedback session at the end of CP2 gives feedback on the overall experience of the puzzle for P2, the participant only spoke for 35.33% during

the feedback session after CP2. For these informal feedback sessions after each puzzle for P1, P2 and P3, participants only spoke between 17.61% and 56.51% of the time, however, participants tended to reveal their experience, perspective on puzzles and ask for explanations on aspects they did not understand to the observer after the puzzle had concluded.

5.4 Discussion

The pieces for CP2 are shorter in comparison to CP1, and CP2's pieces possess more flexibility in the way they can be used or interpreted; this flexibility apparently made them easier to place. The reason could be that CP1's pieces require more processing and thought about the meaning, or that the participant needed more time to read the piece in CP1 than in CP2.

The way in which participants interacted with paper-based Parson's Problems was novel; previous researchers had explored whether the quantitative data analysis could lead to automating the identification of understanding using puzzle-based tasks, however, the findings of this pilot study demonstrate that it is possible to detect the understanding of NPs using puzzles, and while there was not enough recruitment potential to vary more puzzle types to see the effect it had on the learners' explanations, there is evidence to suggest that a more holistic approach of analysing both their explanations and construction process would lead to identifying their understanding.

Likewise, a novel concept has emerged from these datapoints which has been dubbed the 'workspace'; this idea that the NP could reveal a clearer insight into their understanding through grouping and relating pieces together in order to decipher their meaning before placing them in the final solution space.

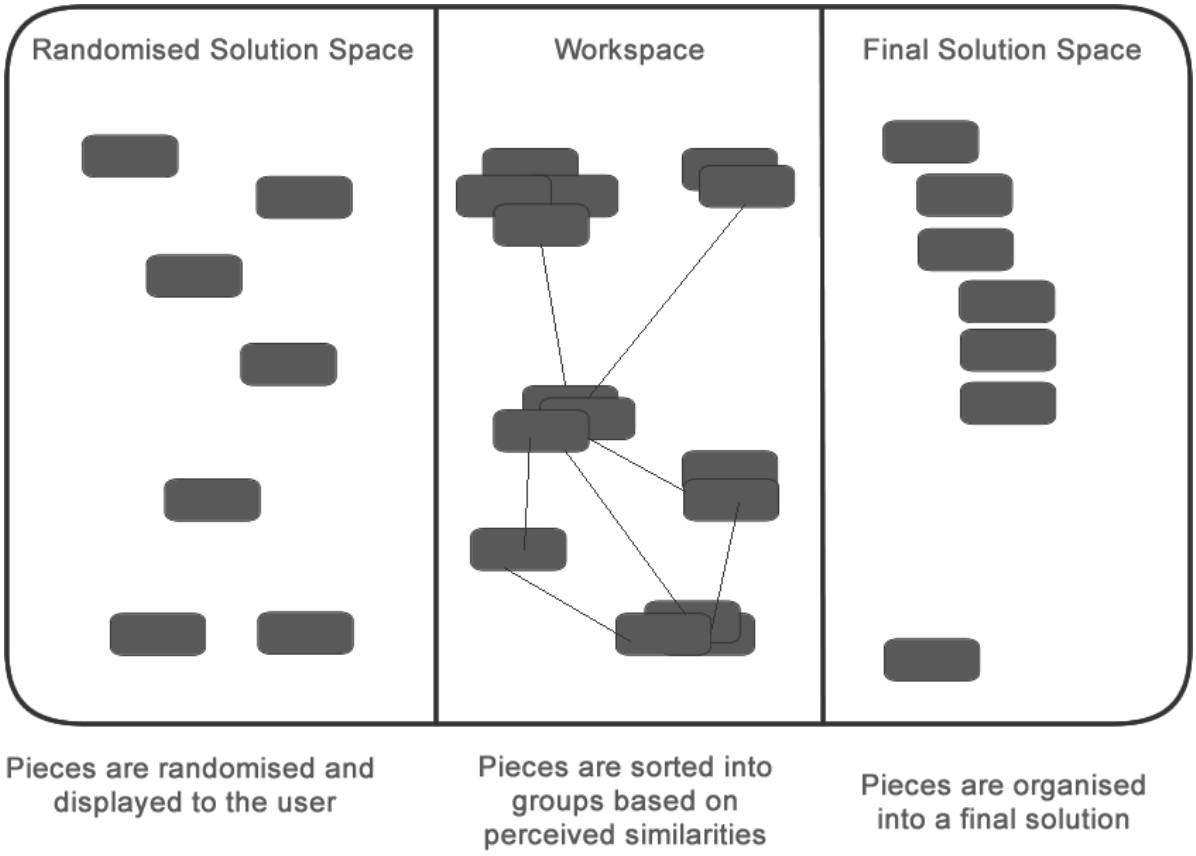


Figure 91: Diagram that illustrates the novel 'workspace' concept for participants to decipher, classify, discuss, and relate pieces to one another before placing them in the final solution space.

There were differing reasons given for why grouping was occurring, namely: Identification: whether they were the same i.e., to lessen time spent looking for pieces; Classification: perceived similarities, i.e.: variables, method visibility, class names; and Line Structure: perceived logic, i.e., arranging the line order before putting the pieces into the final solution.

However, more participants are required to document how widespread this workspace phenomenon is and whether the current interface designs allow for there to be this level of movement or interaction. Therefore, the focus of the secondary study is to repeat this experiment and gather further data on what other interactions participants could have with a flexible working area and to assess whether the information provided from participants who do group pieces together is effective for revealing their level of understanding. A clustering algorithm could be used to identify the proximity of pieces and infer the context behind why they are being clustered. That is rather difficult to implement for a paper-based study, but we will use the video footage to see how distinctly they group pieces and why. With this in mind, it is also important for the secondary study to not be guilty of innovation bias – therefore, no changes were made to the procedure regarding 'forcing' a workspace – instead, it was agreed that participants who were naturally inclined to group should be

studied separately from those who did not in regards to how they grouped pieces and why to see if the findings of the pilot study were based on anomalies.

ID	Hypothesis	Did the pilot study support this?
H-1	NPs of a similar level of understanding will share similar characteristics in their interactions with a particular code puzzle.	Supported; based on the data, those of a similar understanding did generate similar solutions and name and explain pieces in a similar manner.
H-5	There are a finite number of ways in which a code puzzle can be experienced and understood.	
H-2	NP interactions can be classified and categorised	Supported; based on the data, interactions were determinable if a holistic approach was taken – meaning the qualitative data was used to make sense of the movement data and time data. In isolation, movements without the accompanying data were often unclassifiable and potentially meaningless as it would be difficult to know the intention behind a movement without the accompanying audio information.
H-3	Classified NP interactions can be mapped to a level of understanding	
H-4	NPs will make moves that correlate to swap, remove, and add with no other possibilities of movement.	Not supported; based on the data, movements themselves were often unclassifiable except for inserting and removing a piece. Even with the categorisation of insert/add, more information such as the location the piece is being placed in, how many times the piece has been moved and the area it is being transferred to is far easier to classify and assign meaning to. Time, itself, was found not to be a good indicator of understanding or solution quality.
H-6	The reasoning behind a classification of NP interactions will be the same among all participants of that category of interaction.	Supported; participants did insert/remove pieces from the final solution space for similar reasons to one another, that said, participants had different reasons for unclassifiable movements making this metric not as effective as examining their audible reasoning behind movements.

Table 17: How/Whether the pilot study findings supported the original hypotheses

As shown in Table 17, the results of the pilot study were unexpected; consequently, new hypotheses were generated for the secondary study based on these findings. H-1, H-2, H-3, H-5 and H-6 were carried forward to the secondary study, and H-4 was altered for the second study to suggest whether movements could be classifiable if relevant contextual information was present – for example, the

space the piece was being put into, the pieces that had been placed prior, whether it had been grouped etc. – likewise, the data obtained about the novel workspace needed to be examined further so hypotheses were generated around this notion.

These findings conclude that it is not possible to translate purely movement data into a format that could be read and evaluated by a decision tree algorithm (the original intention of this pilot study), instead, we aimed to discover whether these puzzles could be used to diagnose understanding in NPs. This would be of great value as previous research has been inconclusive on such matters; and this would be a necessary step to explore prior to it being possible to automate detection.

5.5 Limitations and Evaluation

The pilot study aimed to replicate past work, however, the software that previous researchers had access to in the field of Parson's Problems were not suitable or available to the researcher due to financial limitations. Yet, because the researcher had to use paper-based Parson's Problems, new observations were recorded which suggests that while the researcher did not manage to successfully mimic the previous work's procedures perfectly, they did manage to discover results that could be potentially useful. One of the major limitations of this study is the limited number of participants that volunteered – five participants is not a number that any statistical significance can be gained from and is too low a number to perform effective Straussian Grounded Theory on. However, the dataset was enough to be able to get feedback on what parts of the study's methodology worked and what parts needed changing. Only one participant attended the optional feedback session, and as the research was to change to interpretivism, there needed to be some way to garner feedback from the sessions as there was no way to tell whether these participants agreed with the observer's observations; as a result, the secondary and tertiary studies offered immediate feedback that was recorded but this still does not account for the missing information here. What would have been useful would be if the researcher had had access to the participants' test scores that related to Java to see if there was any indication from the way they moved their pieces as to how proficient a programmer they were, however, this was not allowed on the grounds of ethical considerations. As there was no access to their grades or ability to do a test in Java prior to the experiment due to time restrictions, it felt difficult to truly assess their ability in Java or languages.

The pilot study is designed to help address issues for future work, and the researcher decided to implement an observer script for future studies as it became apparent that the observer needed to be careful about how they worded their responses to questions asked by the participant in case they influenced the participant's perspective on what they were doing. Participants did tend to ask

questions such as “does this look right?” and this made it difficult for the observer to know if the participant was asking the observer for reassurance, or whether they were asking a rhetorical question. Another issue that contributed to the creation of an observer script was that participants would sometimes go quiet, and the observer did say slightly different things to them – it was better to unify the prompt into a standard statement rather than informally asking them to speak. The final issue that contributed to the creation of an observer script was that if a participant was using a movement that the observer wanted to know more about, it was difficult to think of how to phrase the question without affecting the confidence of the participant in their movement – if that observer had asked a question phrased informally of the ilk of “why are you doing that?” this would cause issues and highlight to the participant that they might be doing something unexpected, therefore, formalising the question to a standard prompt helped to distance the observer. Additionally, the introduction of a yellow card to indicate that the participant wants to ask the observer a question may help participants to feel more comfortable with communicating to the observer. The introduction of a red card to signal that the participant was ready for submission also helps to give the participant the ability to stop the experiment at any time – it also avoids the issue of the observer needing to ask if the participant has finished and can allow participants to fall silent if they need to. P2 particularly showed signs of feeling rushed despite the observer holding back and the introduction of cards may remedy this.

Regarding the questionnaires, the pilot study only asked for the participants’ perspectives after they had completed the puzzle rather than on the perspective of the task itself – this meant that it was difficult for the researcher to gauge whether CP2’s puzzle solution was harder to create than CP1’s puzzle solution or whether the task description was more difficult to interpret – this led to the creation of a pre-task questionnaire. There was also the question of whether the participants knew other languages and paradigms other than Java and, if so, what affect knowing these languages had on the approach to creating a solution (if any).

5.6 Conclusion

Ultimately, the pilot study could be deemed a success; some of the results were not as anticipated for the pilot study had aimed to mimic previous work in the area as closely as possible and such observations had not been recorded before, but the experiment revealed a potentially new avenue for exploring based on the data obtained from P1’s movements. The study helped to refine the study procedure further, and the ideas generated from the findings helped to form plans on how to mitigate certain issues (such as not knowing when the participant has submitted their solution, and issues regarding participant uptake being so low). The study’s findings also shaped future studies’

foci, as the original research philosophy of post-positivism was no longer appropriate, and an interpretivist approach was taken instead. The study allowed the researcher to explore the issues behind using paper-based tools and helped to lay the groundwork for how to analyse the movement data from the footage. This study showed promise for future work, and it was decided that a secondary study would be produced with the identified issues of the pilot study ironed out.

5.7 Chapter 5 Summary

This chapter presented, explored and evaluated the findings of the pilot study.

This study assessed the feasibility of using NP interactions with Code Puzzles to categorise their understanding of programming, with the conclusion that it is feasible to obtain some form of insight into the NP's understanding of programming, but that further investigation was required to determine the prominence of the workspace phenomenon, and also to evaluate whether the analysis of the understanding obtained from participants' interactions by the observer is accurate in accordance to the NP's perspective of their own capabilities and understanding of programming.

Limitations in the study's design became apparent, and further changes were incorporated for the secondary study to minimise the risk of leading participants and to set a more realistic time frame for the experiments. Similarly, changes were incorporated to expand what data is obtained to help evaluate the accuracy of the representation of understanding obtained from analysing participants' movements.

In conclusion, the pilot study revealed that having a post-positivism focus on the examination of Code Puzzle interactions does not allow for the full picture to be obtained, and that future work required a change to an interpretivism research philosophy with less focus on full automation of movement analysis as the movement type and approach did not reveal understanding as clearly as the participants' speech or frequency of interactions with a piece.

Chapter 6. Secondary Study: Understanding NPs and Workspace Influence

This study aimed to address the following question (see Table 18):

Secondary Study's Research Question
How does the workspace influence the quality of the data collated regarding an NP's understanding?

Table 18: Secondary Study's Research Question

Three first year CS undergraduate students from one university volunteered to participate in the tertiary study; they had all completed the foundations in Java module and allegedly knew the basics of Java and Object-Orientated design. The uptake for this study was low, likely due to the study taking place in 2020 (when the COVID-19 pandemic occurred) which impacted the ability for the researcher to offer a financial incentive to participate in the study. Similarly, the university was closed to most students which affected the ability to advertise the study effectively. The study also was conducted over the summer period during Covid, where lockdowns were commonplace in the UK and many students were deferring their examinations – as such, the uptake was very poor, but the three cases documented each give an individual insight into the issues of being able to utilise code puzzles in pre-existing environments.

6.1 Hypotheses

Based on the pilot study findings and previous research (i.e., Ihantola and Karavirta, 2011; Helminen, Ihantola, Karavirta, and Malmi, 2012), hypotheses were developed and formed the basis of the investigation (see Table 19).

ID	Hypothesis	Did the pilot study support this?
H-1	NPs of a similar level of understanding will share similar characteristics in their interactions with a particular code puzzle.	Supported by the pilot study.
H-5	There are a finite number of ways in which a code puzzle can be experienced and understood.	Supported by the pilot study.
H-2	NP interactions can be classified and categorised	Supported by the pilot study.
H-3	Classified NP interactions can be mapped to a level of understanding	Supported by the pilot study.
H-4	NPs will make moves that correlate to swap, remove, and add with no other possibilities of movement.	Not supported by the pilot study.
H-6	The reasoning behind a classification of NP interactions will be the same among all participants of that category of interaction.	Supported by the pilot study.
The hypotheses below this threshold were based on the findings and observations recorded in Chapter 5, and not on previous research and had not yet been explored prior to this study but were supported by the pilot study's evidence.		
H-7	The more incorrect placements of Code Puzzle pieces that NPs make, the easier it will be to indicate misconceptions.	
H-8	NPs of a similar approach to coding will share characteristics in their understanding.	
H-9	NPs will leave pieces that they are least confident about until last.	
H-10	NPs will prefer the Line-By-Line Code Puzzles over the Piece-by-Piece Code Puzzles.	
H-11	NPs will find Code Puzzles useful; this will be determined by whether the majority of participants (over 50%) find the Code Puzzle useful in the post-study questionnaire.	
H-12	NPs will find Code Puzzles as a viable alternative to traditional revision methods; this will be determined by whether the majority of participants (over 50%) find the Code Puzzle useful in the post-study questionnaire.	
H-13	NPs will find the Code Puzzle analysis accurate in regards to their approach; this will be determined by whether the majority of participants (over 50%) find the Code Puzzle useful in the post-study questionnaire.	
H-14	NPs will find the Code Puzzle analysis accurate in regards to their understanding of the underlying concepts; this will be determined by whether the majority of participants (over 50%) find the Code Puzzle useful in the post-study questionnaire.	

Table 19: Original hypotheses specifically for the secondary study

6.2 Secondary Study Procedure, Results and Discussion

The secondary study produced 40 audio recordings and 54 video recordings; no participants' data suffered loss, but one participant (P11) had two audio recordings as there needed to be a pause and emergency room change during their attempt of CP2. However, most participants had more than one video for CP2 due to requesting additional, custom pieces.

6.2.1 Time, Solution Confidence, Perceived Task Difficulty, and Movement Results

Mirroring the pilot study's results, CP1 took less time to complete in comparison to CP2, but more time was spent on individual pieces on average than CP2. Therefore, in terms of time efficiency, 2D Parson's Problems are better as NPs spend less time completing them (see Figure 92).

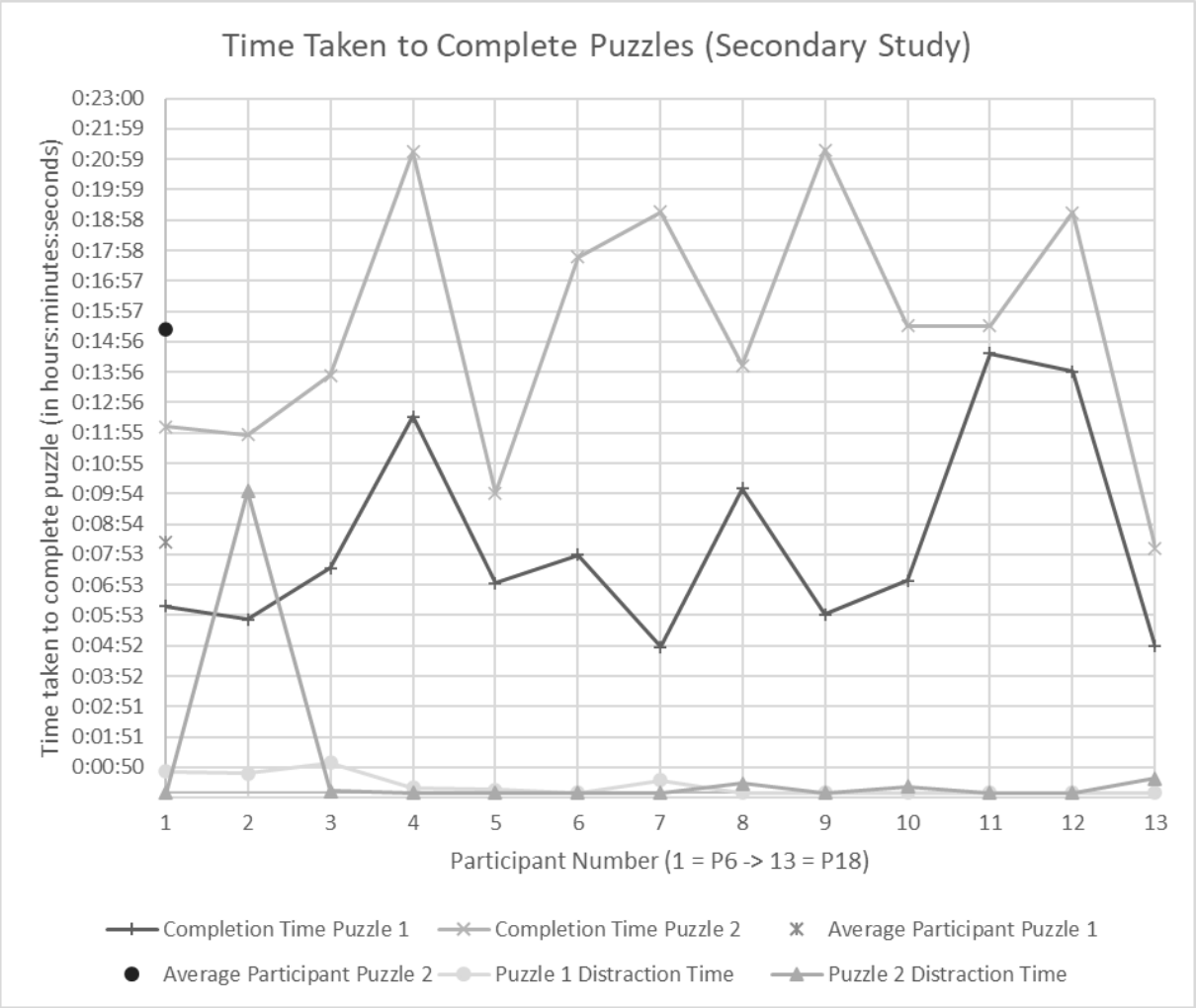


Figure 92: Scatter graph for time taken to complete the puzzle by each participant (CP1: range = 04:50-14:33, M = 08:16, SD = 03:22|| CP2: range=08:06-21:17, M = 15:21, SD = 04:13)) for the secondary study. Distraction time was labelled as moderate or severe interruptions (CP1: range=0:00-1:37, M = 0:27, SD = 0:42|| CP2: range=0:00-0:59, M = 0:16, SD = 0:26)

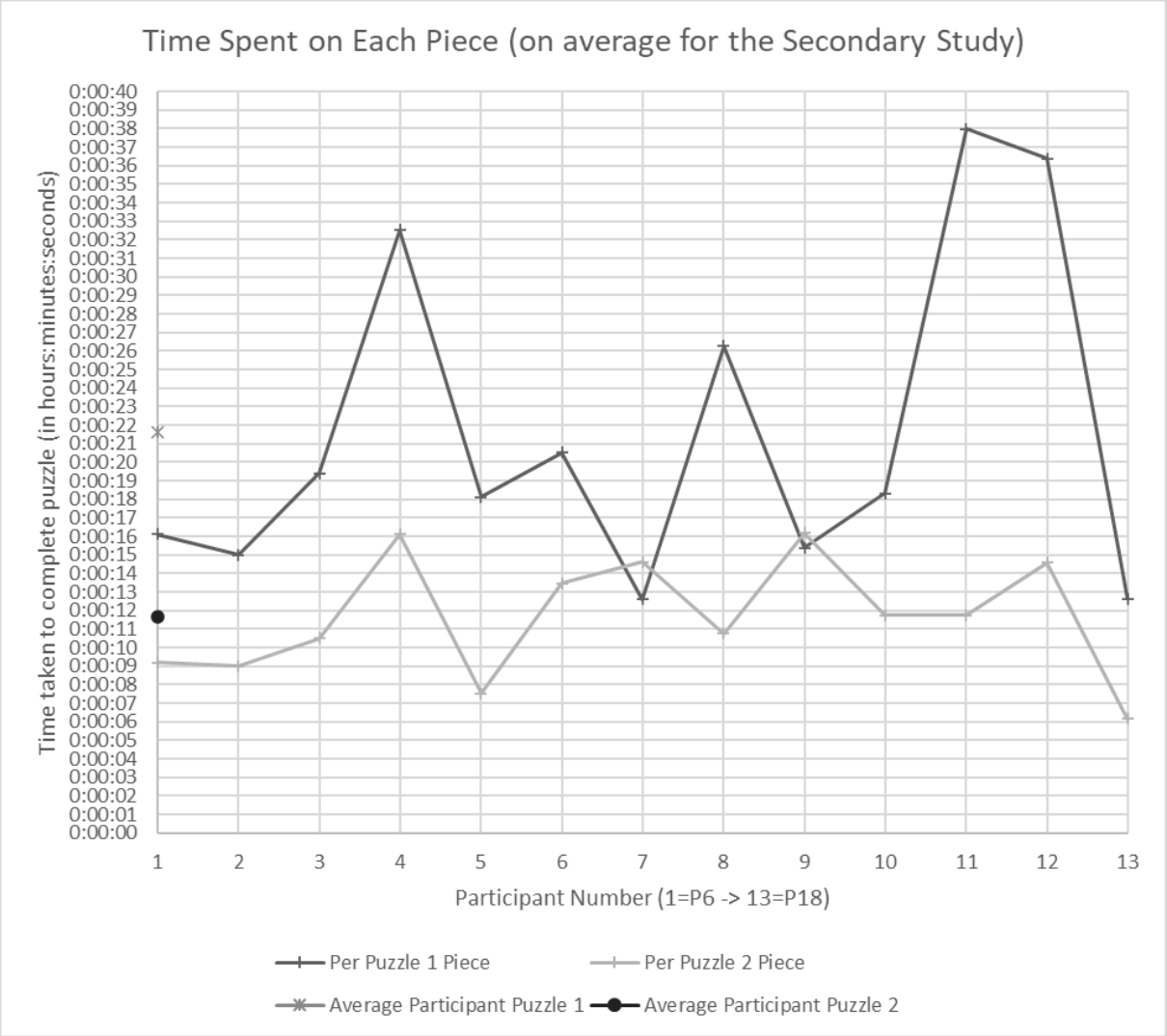


Figure 93: Scatter graph for the average time spent on each piece per puzzle (right) (CP1: range = 00:13-00:38, M = 00:22, SD = 00:09|| CP2: range=00:06-00:16, M = 00:12, SD = 00:03) for the secondary study.

Participants were given the task information sheet and asked to complete a pre-puzzle questionnaire on the perceived task difficulty (based on how difficult they felt it would be to create a solution on their own computer) before they were allowed to view the associated code puzzle pieces for the task (see Figure 94 and Figure 95).

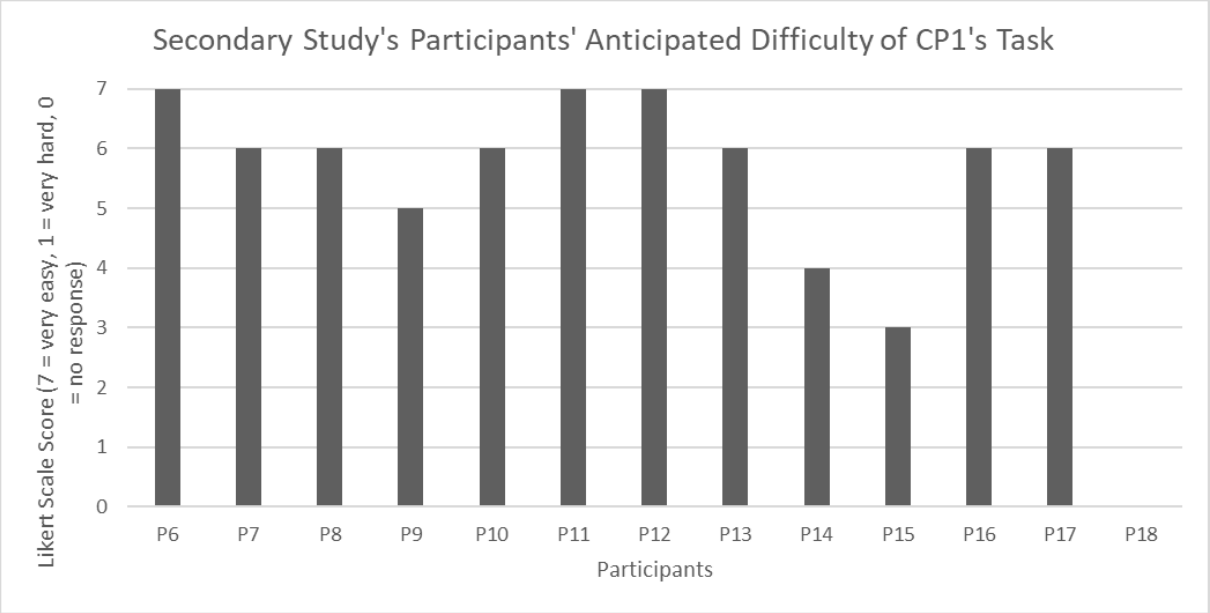


Figure 94: Bar chart for the pre-CP1 (CP1) questionnaire's closed questions on task difficulty for the secondary study. P18 did not submit a pre-CP1 questionnaire. (CP1: M = 'Slightly easy')

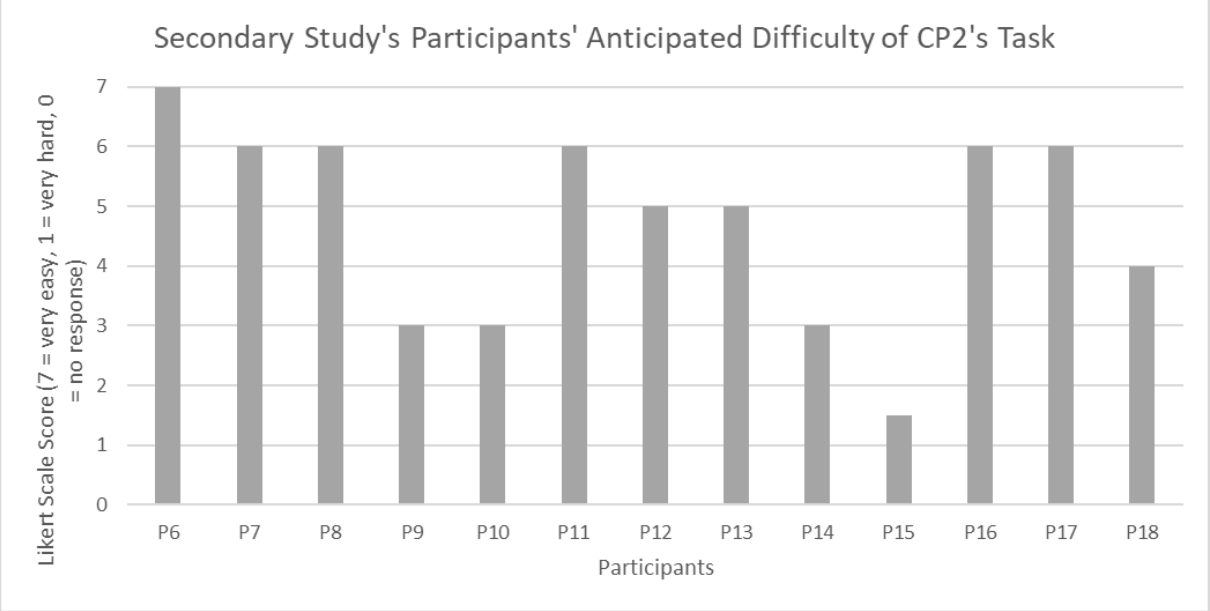


Figure 95: Bar chart for the pre CP2's (CP2) questionnaire's closed questions on task difficulty for the secondary study. P18 did not submit a pre-CP1 questionnaire. (CP2: M = 'Slightly easy')

The average participant believed that CP2's task was naturally more difficult than CP1's task; however, if the mean response was taken from each puzzle, the average participant would choose 'Slightly easy' for both puzzles. That said, the chart comparison shows that there is an inclination towards CP1 being easier than CP2 – even if there are extremes skewing the mean option chosen. This was primarily due to the inclusion of the Date library; like the pilot study, the participants insisted that they had not covered the necessary materials in the Java module to be able to interpret and use text from Java documentation. P13 had the most drastic difference in opinion; they believed

that CP2 would be very difficult as they had not encountered Java documentation before. This, however, is an intriguing point in itself – as highlighted in the background questionnaire, 9% of participants felt that experience was key to being a sound programmer – as though, they imagine, an adept programmer to know all there is to know about a language and therefore never encountering unknowns. Perhaps this is a source of frustration – programming, by nature, includes creating programs that have not been created before and languages typically do advance and change over the years. For example, Oracle (2020) show that they have depreciated libraries, so content taught to previous programmers is not necessarily the same in a decade or so's time. Languages also grow and fall in popularity, and perhaps change is a reason that NPs are fearful and become deterred from completing a task. It is also unlikely for a programmer to write the exact same program twice – each task is always a new experience in terms of the domain.

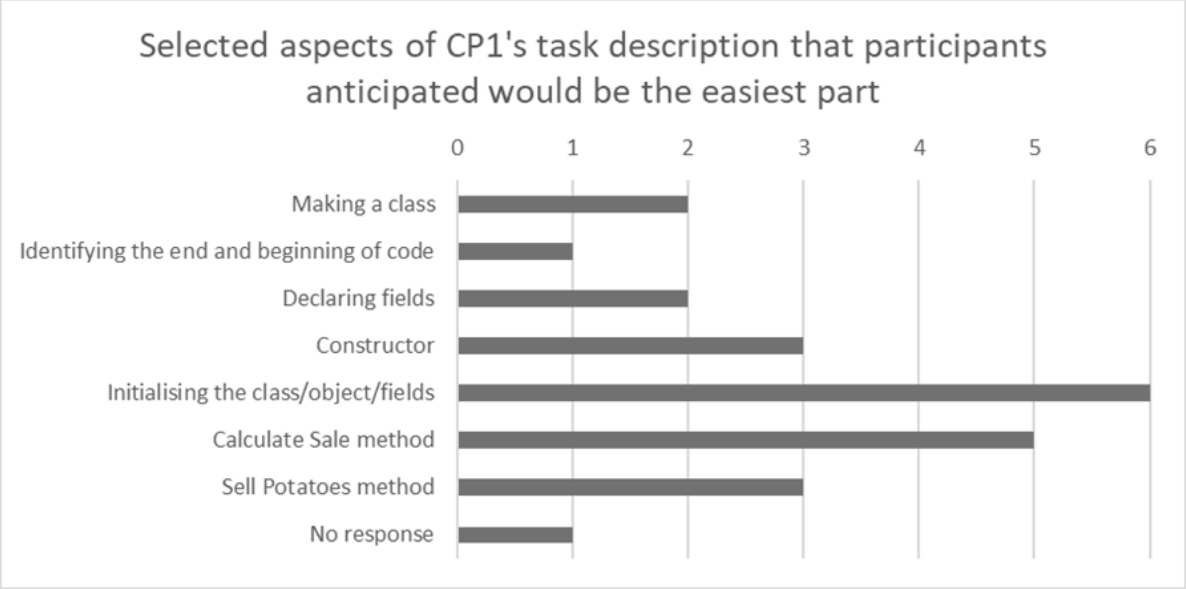


Figure 96: Bar chart showing what participants anticipated would be the easiest part of CP1.

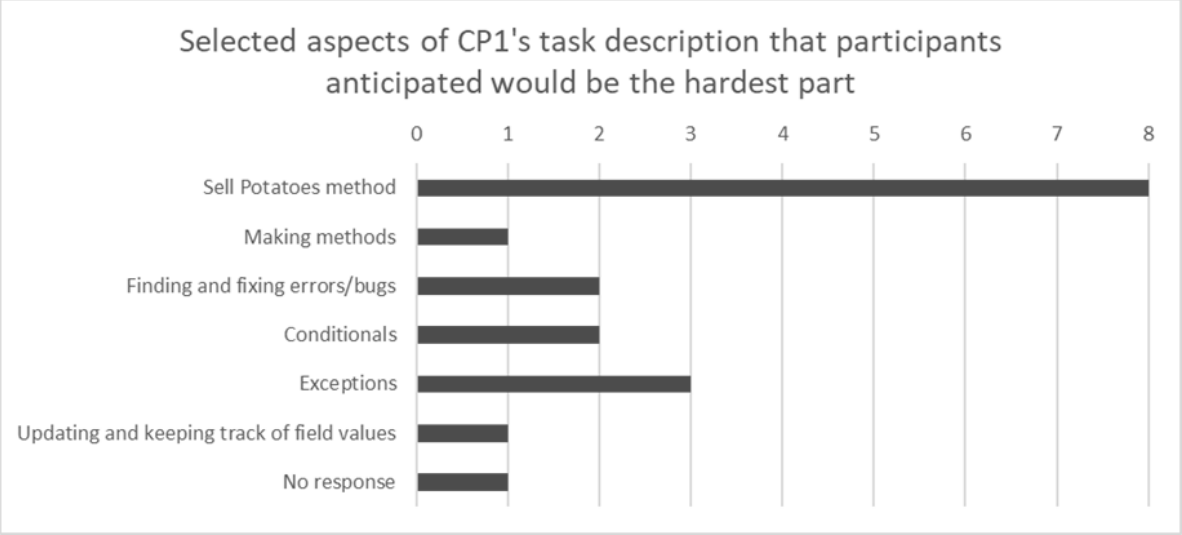


Figure 97: Bar chart showing what participants anticipated would be the hardest part of CP1.

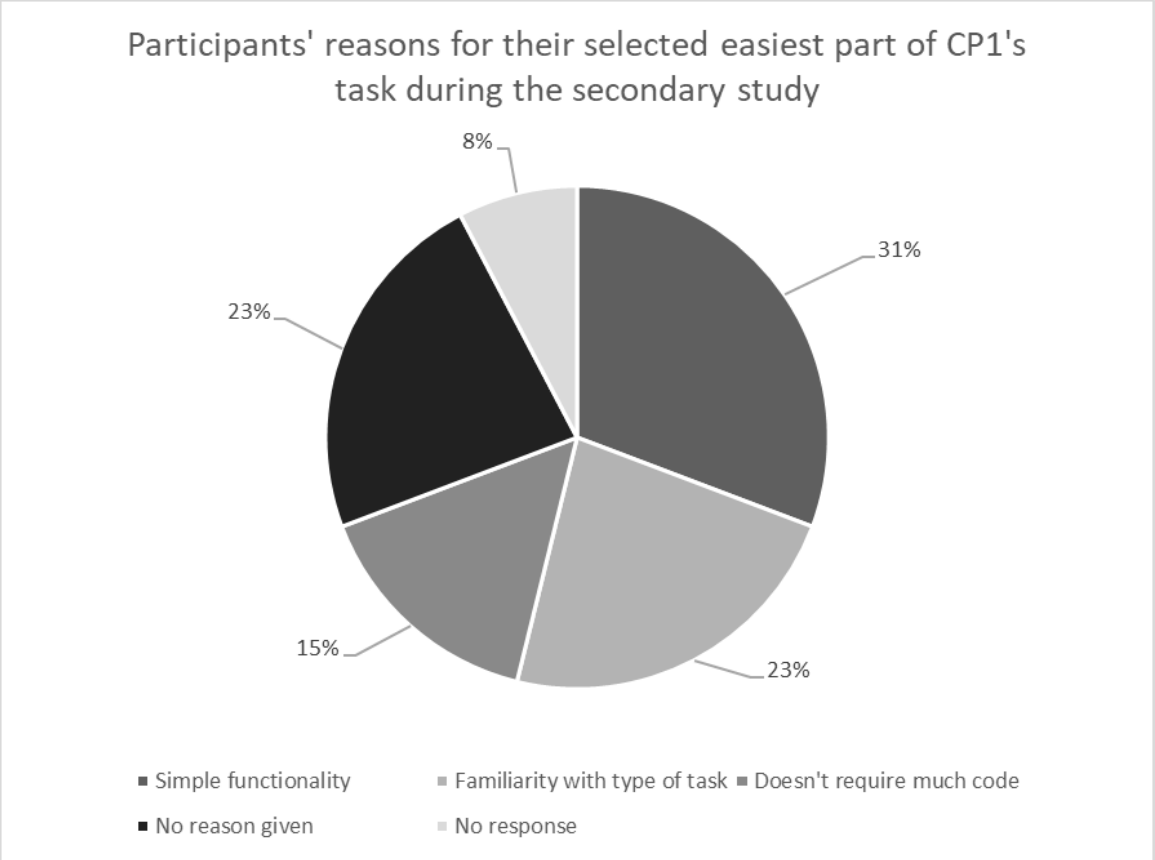


Figure 98: Pie chart showing what participants reasons were for the easiest part of CP1.

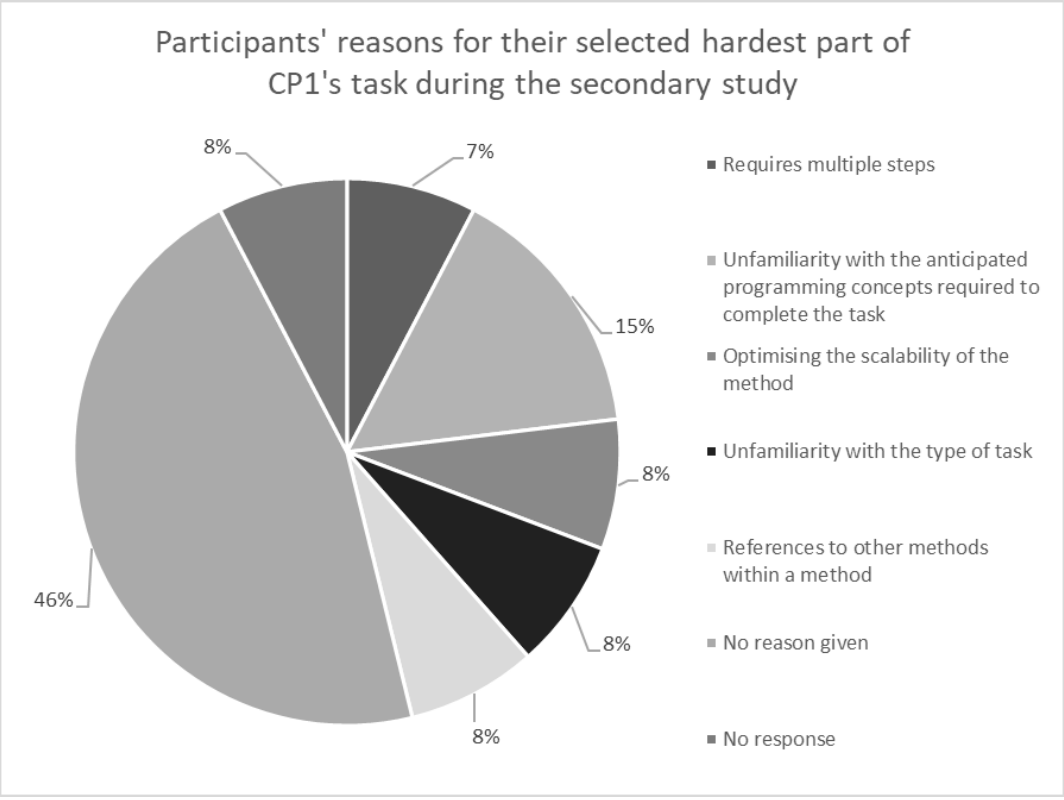


Figure 99: Pie chart showing what participants reasons were for the hardest part of CP1.

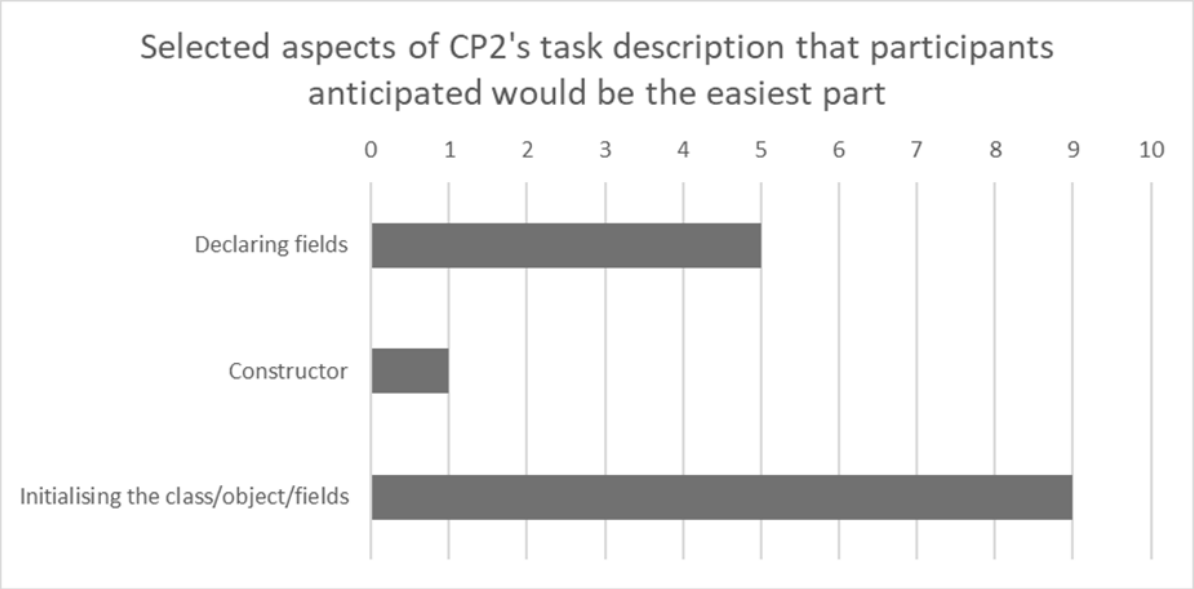


Figure 100: Bar chart showing what participants anticipated would be the easiest part of CP2.

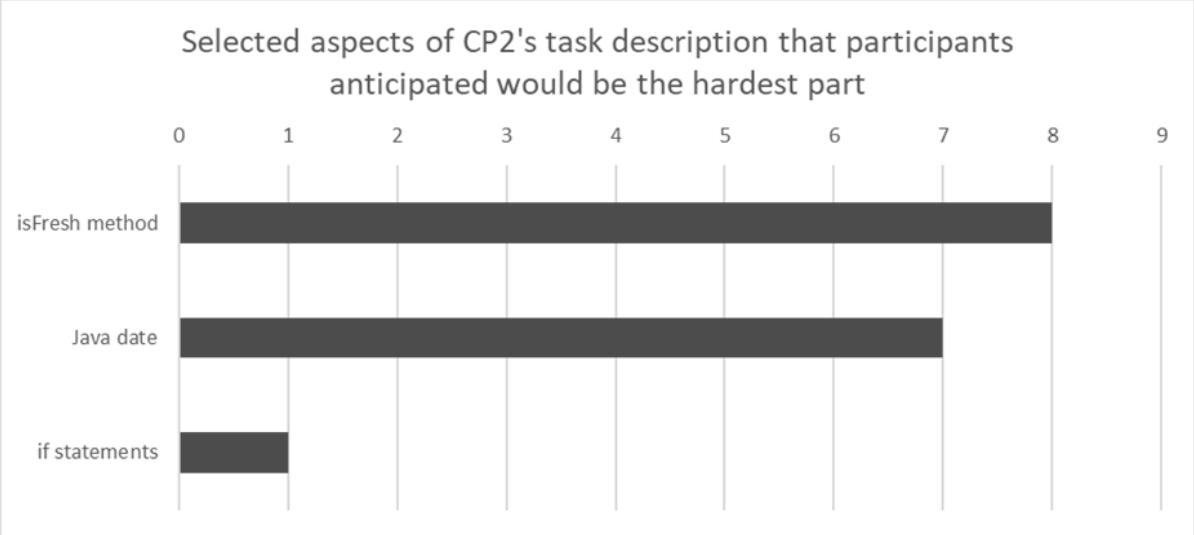


Figure 101: Bar chart showing what participants anticipated would be the hardest part of CP2.

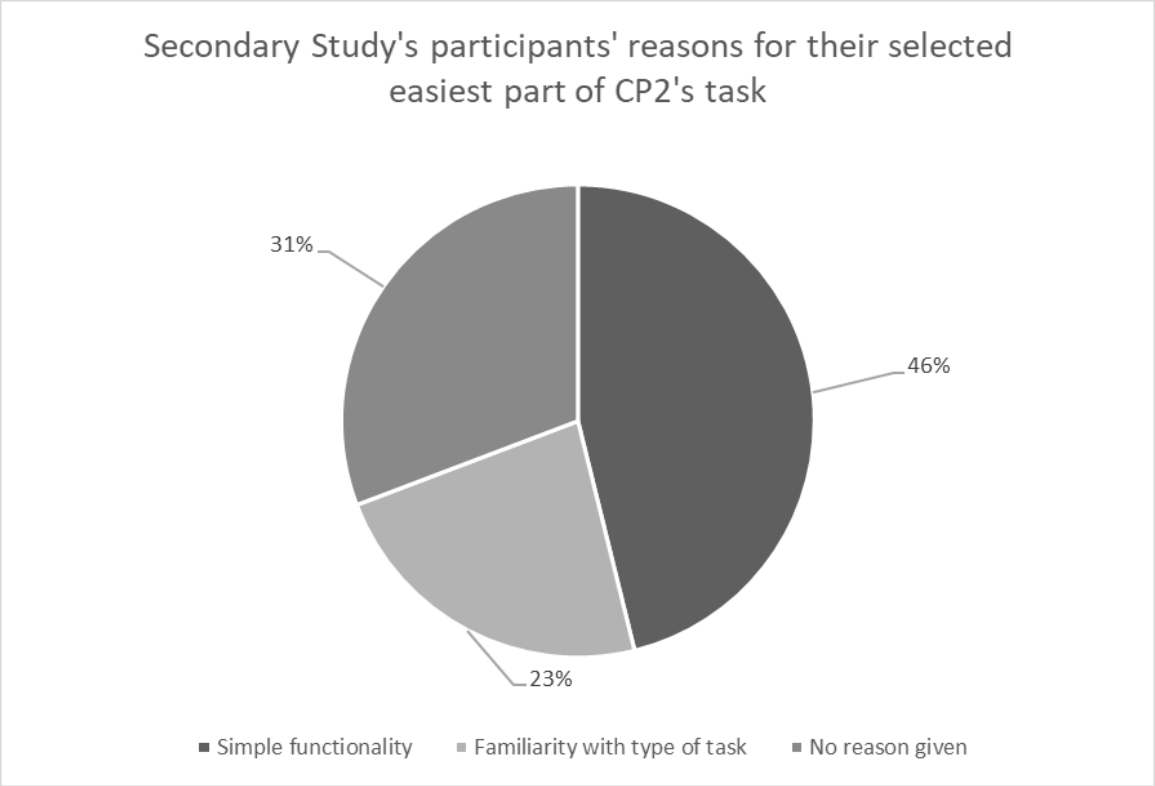


Figure 102: Pie chart showing what participants reasons were for the easiest part of CP2.

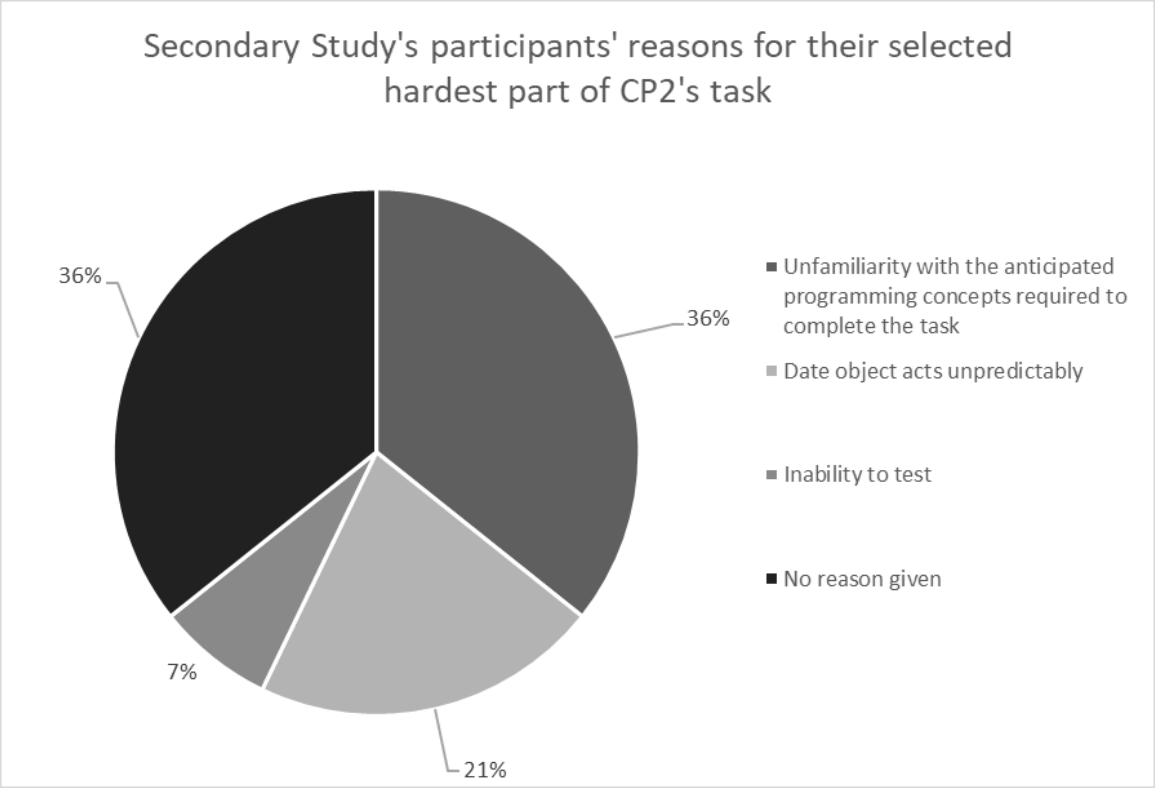


Figure 103: Pie chart showing what participants reasons were for the hardest part of CP2.

Participants generally felt that CP2 was more difficult than CP1, likely due to them focusing on the unfamiliarity of the date class as shown in the pre-CP2 questionnaire. The number of pieces also contributed to the difficulty.

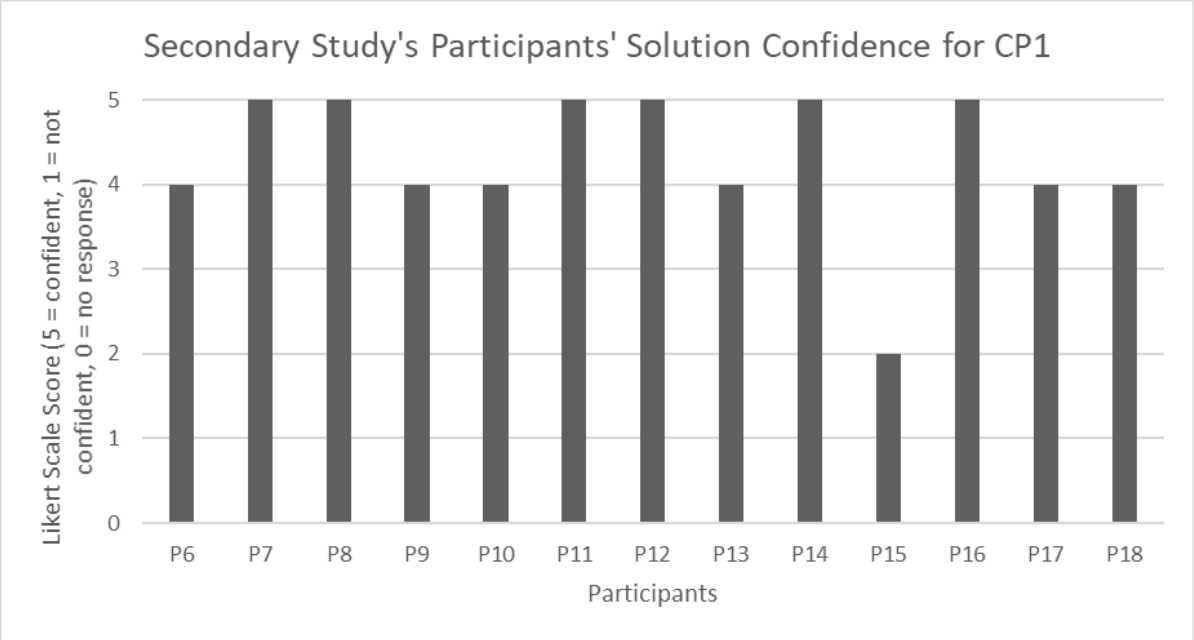


Figure 104: Bar chart showing the post-CP1 evaluation of how confident the secondary study participants were that their solution would work.

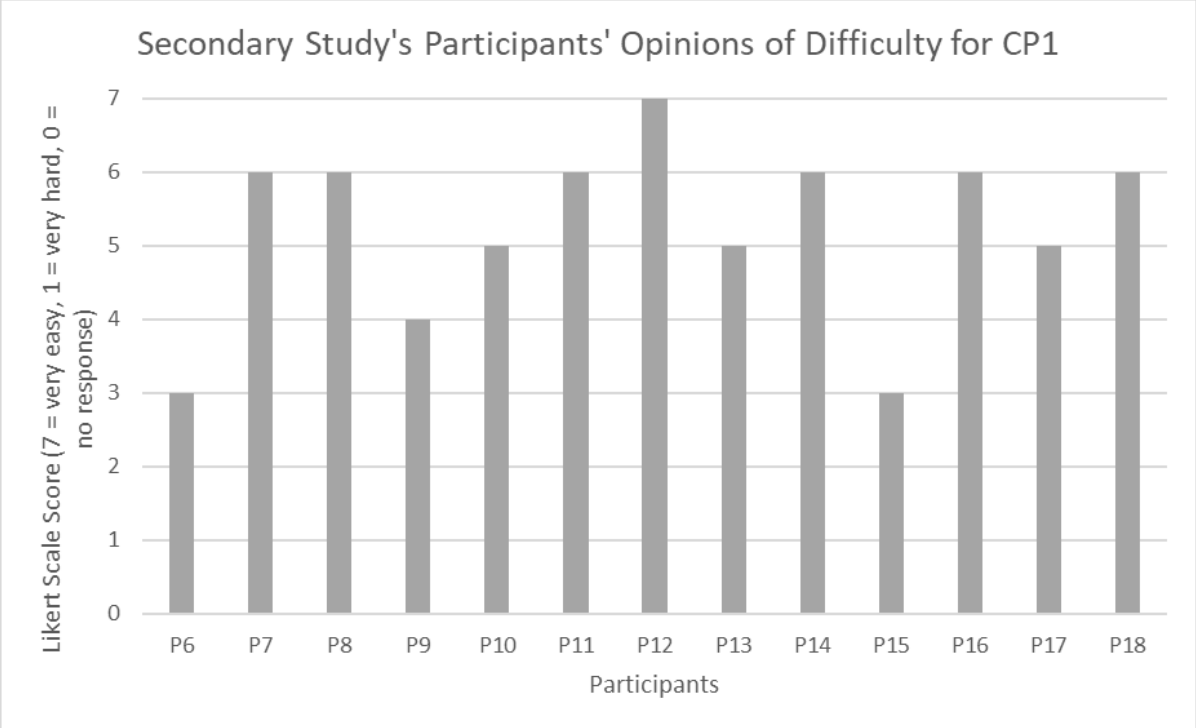


Figure 105: Bar chart showing the post-CP1 evaluation of how confident the secondary study participants were with the achieving the task.

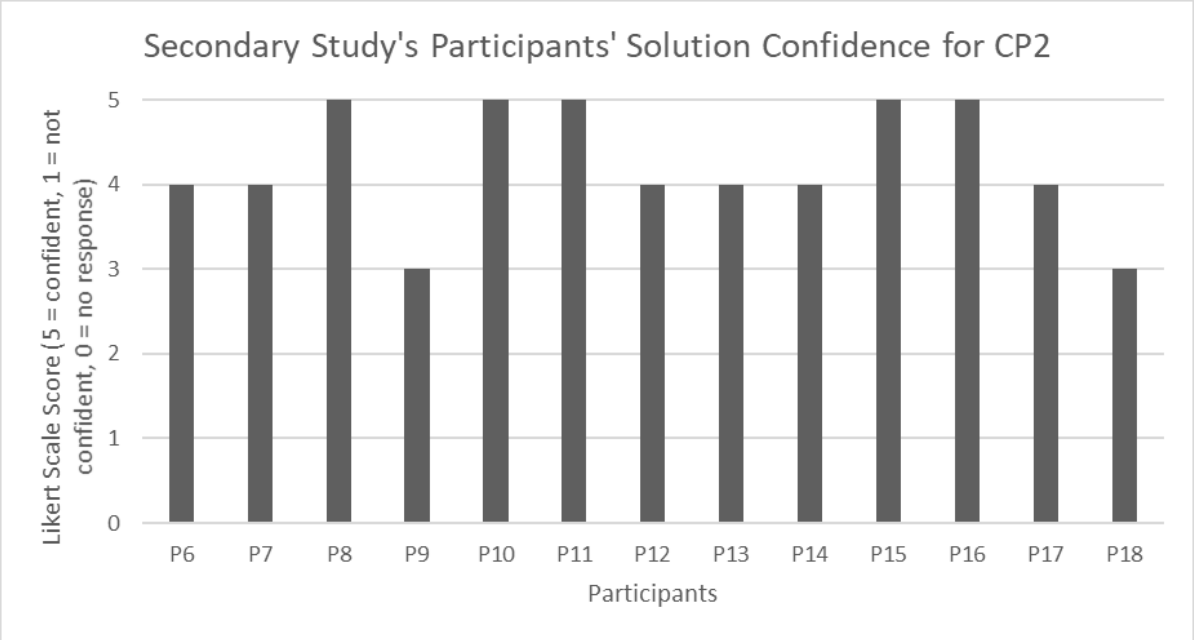


Figure 106: Bar chart showing the post-CP2 evaluation of how confident the secondary study participants were that their solution would work.

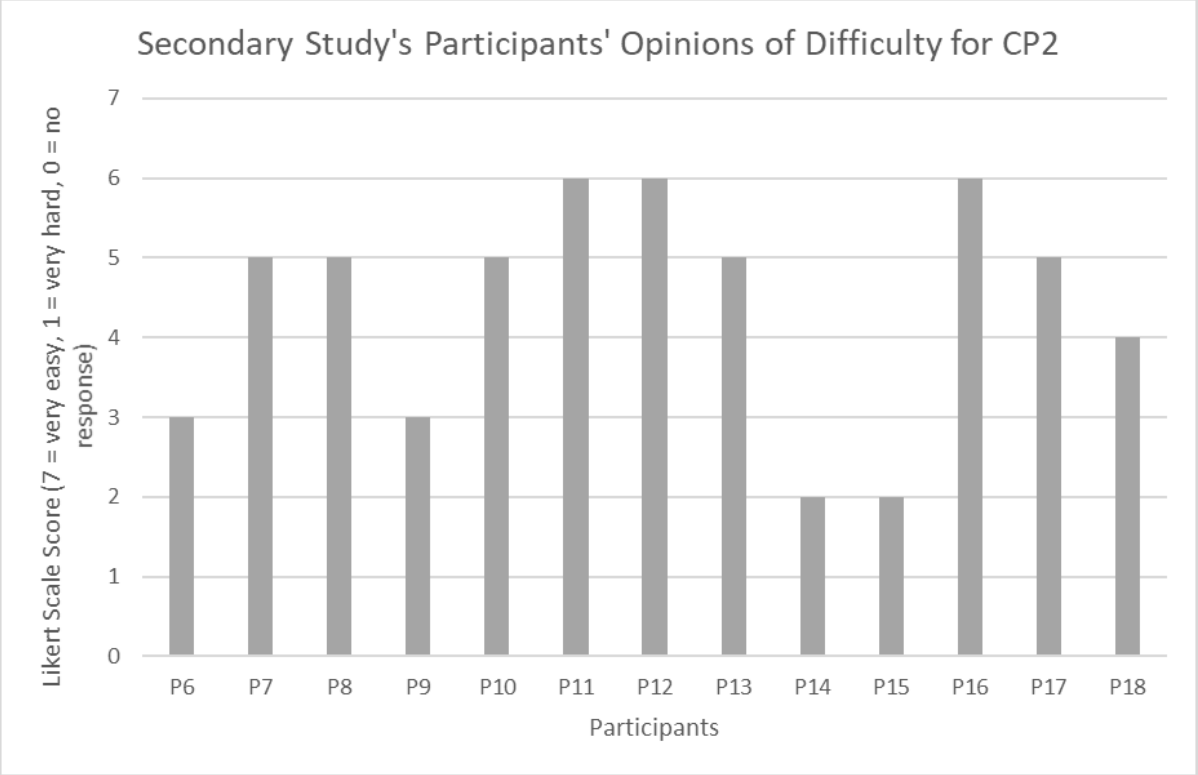


Figure 107: Bar chart showing the post-CP2 evaluation of how confident the secondary study participants were with the achieving the task.

6.2.2 Background Questionnaire Results

11 out of 13 participants completed the background questionnaire, with 2 participants opting out of completion. 1 participant did not specify any languages in their background questionnaire. Therefore, this section will equate 100% of participants as 11 participants.

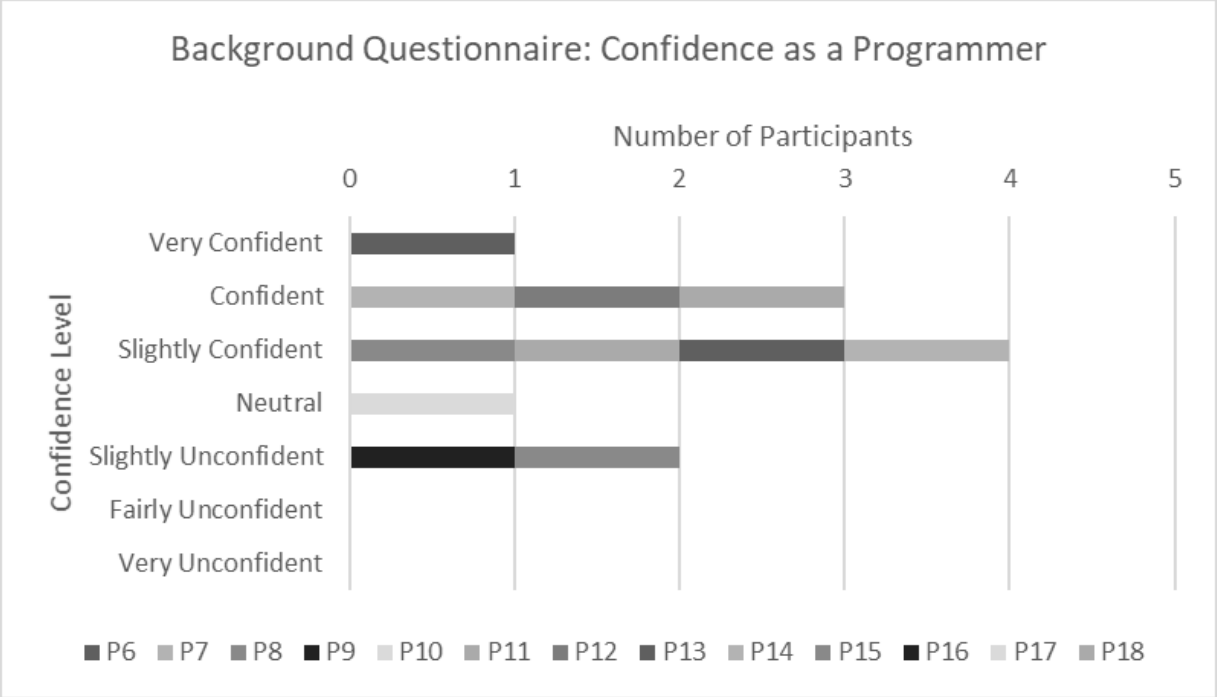


Figure 108: How confident are you as a programmer? (M = "Slightly Confident")

The average participant was slightly confident (see Figure 108). This may indicate a sample bias of a similar nature as the pilot study where only the confident programmers volunteered; that said, while there were two participants who were confident or very confident, there were three who were 'slightly unconfident' which means the sample is a little more balanced that the pilot study.

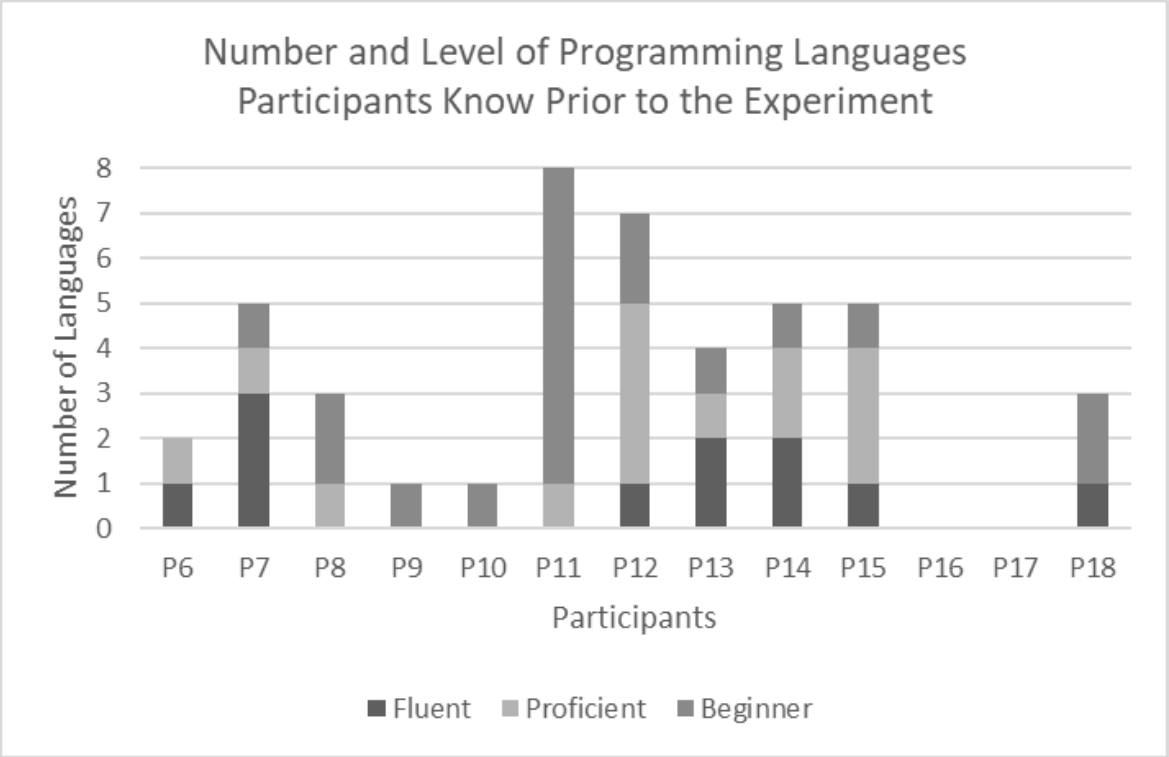
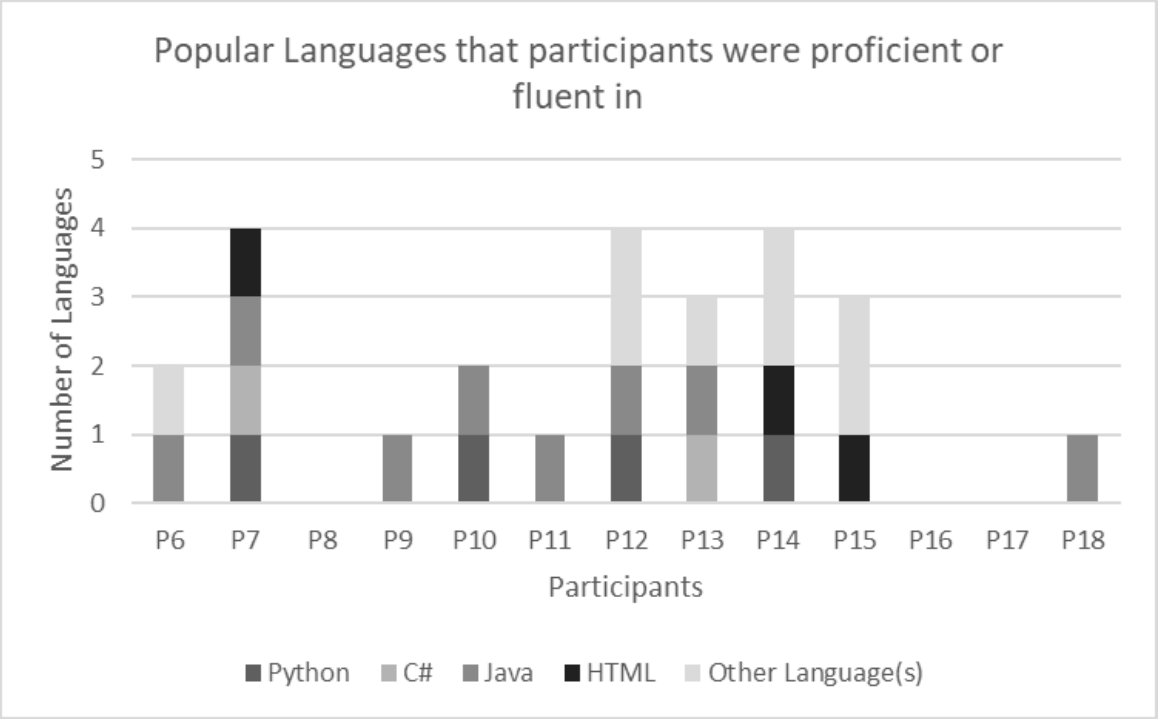


Figure 109: How many programming languages are you: fluent, proficient and beginner in? (Fluent: range = 0-3, M = 1, SD = 1 || Proficient: range = 0-4, M = 1.36, SD = 1.21 || Beginner: range = 0-7, M = 1.73, SD = 1.85)

The average participant had 1 language they were fluent in, 1 language they were proficient in, and 2 languages they were a beginner in (see Figure 109). This may indicate a sample bias to the degree that there are only two participants who are experienced in three or less languages. That said, due to the nature of the undergraduate degree at Aston this was unavoidable as, alongside the compulsory Java module, students do learn other languages in differing modules. As a result, participants were asked to clarify which languages they were familiar with (see Figure 110).



What programming languages are you fluent or proficient in? 'Other Languages' are for languages selected by only that participant and could compromise their identity, so if they put at least one language that isn't explicitly stated on the graph it is counted once. (Java: total = 8 participants || Python: total = 4 participants || HTML: total = 3 participants || C# = 2 participants)

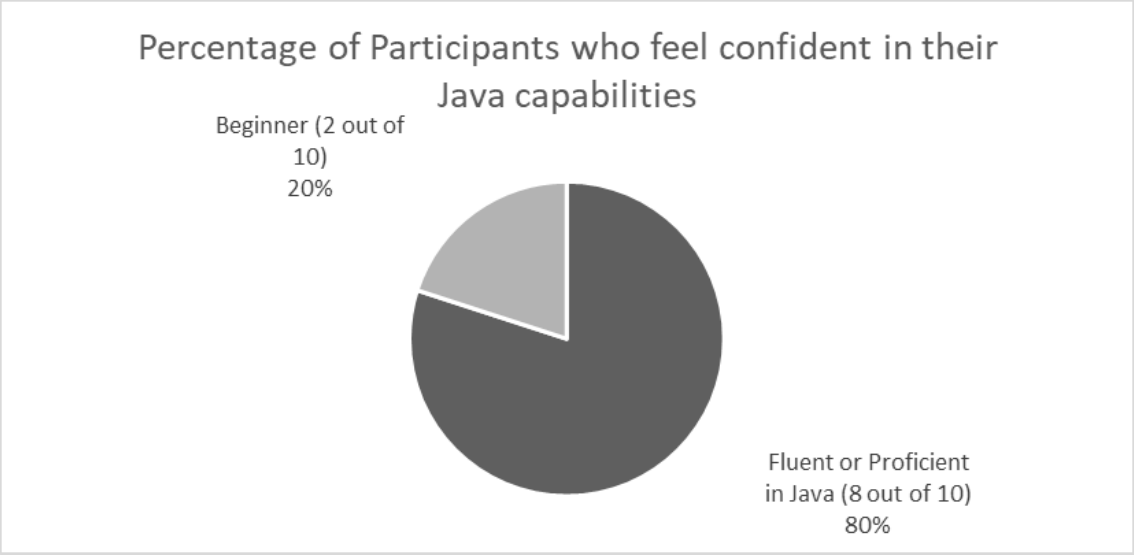


Figure 110: Pie chart for the percentage of participants who answered that they were proficient or fluent in Java.

82% of participants felt proficient or fluent in Java, which further clarifies that the sample collated is likely to have a sample bias towards more confident Java programmers. However, such data is still useful to collect and analyse to see if understanding of different levels of NPs can be identified from the puzzle interactions. An 'Other languages' label was used to mark languages that only one

participant selected to avoid ethical issues associated to revealing the participant's identity. Some participants in the secondary study chose multiple 'Other' languages and this has been accounted for in **Figure 109** and Figure 110, for example, P11 was a beginner in 7 'Other' languages. Originally, the researcher wished to separate the 'Other' languages into programming paradigms, but this proved to need too many assumptions as certain 'Other' languages were potentially multi-paradigm

While as much data was obtained as possible, it should be noted that data saturation for these questionnaires has not been achieved in accordance with the IPA as there are no common phrases or specific words shared between participants. That said, these questions aimed to provide a more informed baseline for the investigation than the pilot study assumed that the participants sharing a Java module on their degree course would be adequate for accounting for differences in understanding. Therefore, it is sufficient for the sake of this investigation that enough data has been collated to analyse similarities, albeit more qualities or themes could transpire if this study was repeated.

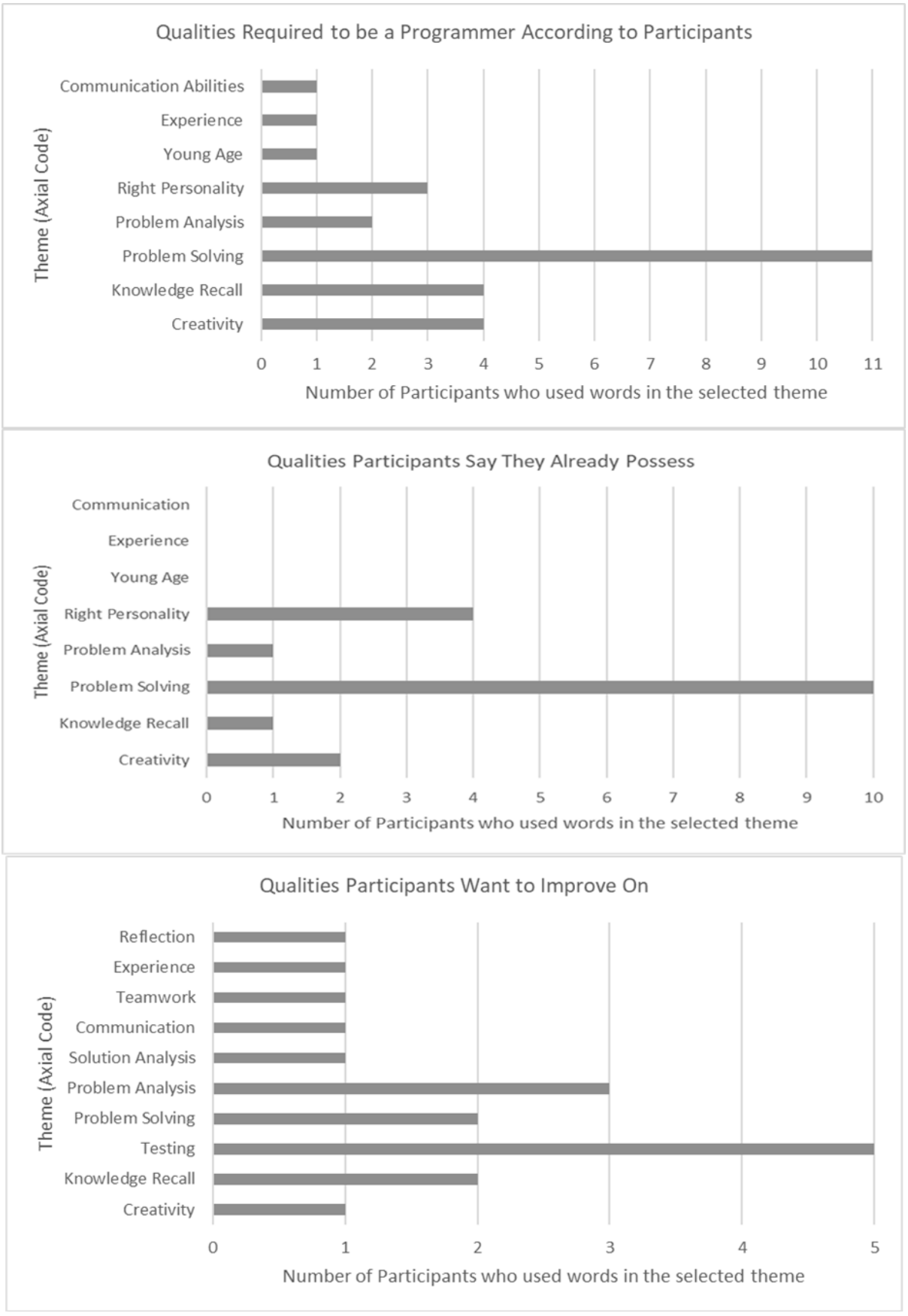


Figure 111: Three stacked bar charts that illustrate the secondary participants' answers to three related questions in the background questionnaire: 'What qualities does a programmer require (in your opinion)?'

(top); ‘Which qualities of a programmer do you feel you have?’ (middle); and ‘Which qualities of a programmer do you feel you need to improve on?’. These were codes created through applying Straussian Grounded Theory to the open-ended question answers given.

100% of participants believed problem solving to be a mandatory skill for a programmer, with 18.18% believing that problem analysis was worthy of particular note regarding the problem-solving process. While this could be a side effect of advertising the study as a way to evaluate and detect the understanding and process that a programmer goes through – ergo, attracting volunteers who believe the purpose of this experiment is important – it should be noted that the advertisement never used the terminology ‘problem solving’ itself. 36.37% of participants felt that knowledge recall and memorisation of the language’s syntax, structure and basic functionality was required, and 36.37% believed creativity was a skill needed by programmers. 27.27% believed that a programmer needed the right personality and temperament in order to excel, with participants commenting on patience and persistence as being signs of a good programmer. While participants tended to relate personality traits to their own experiences, for example – creativity comments tended to talk about designing programs and finding solutions to bugs, whereas patience related to persevering with programming. Creativity and right personality were separated due to how many participants specified creativity as a specific characteristic without alluding to other personality traits. Interestingly, 9% of participants felt that programming needed to be discovered at a young age in order to excel at it, and 9% of participants also felt that skills such as teamwork and communication abilities were paramount to being an adept programmer.

90.91% of participants believed they possessed problem solving capabilities, which suggests that they have personally found the skill important when they are programming. Similarly, 36.36% of participants commented that they had the right temperament and 27.27% of participants believed they were creative enough to become a proficient programmer, even if they lacked the experience or adequate knowledge recall. Conversely, 9% of participants felt they had the knowledge but needed to improve their problem-solving capabilities.

Developing the thematic relationships for the questions related to the qualities required for being a programmer; what qualities they believe they possess and what they wish to improve on were difficult to determine as only 27.27% of participants wrote about more than one theme per question (see Figure 112).

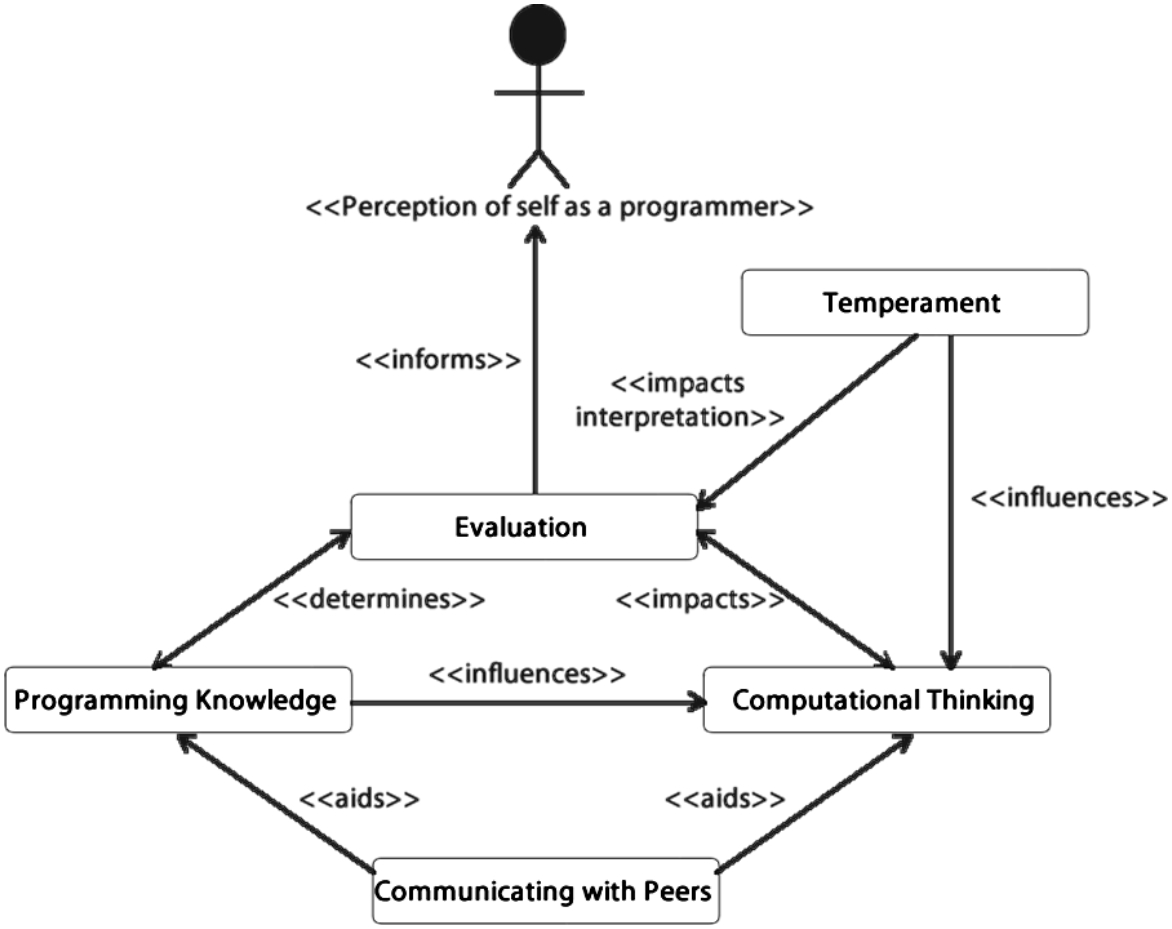


Figure 112: Interpreter's Sensing of the Thematic Relationships for the question "Which qualities of a programmer do you feel you need to improve on?"

This shows that NPs value the overall problem-solving process alongside believing that personality and temperament are contributing factors to success. This is insightful; as participants who did have issues with creating a working solution were recorded to be struggling with knowledge as well as selection of a relevant problem-solving strategy. This reveals that participants can identify the general source of their issue (i.e., problem solving), but struggle to identify the specific part of that general concept that they struggle with (i.e., selecting relevant information, dividing and conquering etc.). This information further suggests that NPs do struggle with communication barriers about explaining the parts they wish to improve on. For example, 27.27% of participants used the exact same phrase – "thinking outside the box" – which is vague, but also suggestive that they have encountered previous programming issues or ways of solving programming that have been beyond their expectations of what programming should be.

The themes for the improvement question are defined in full (see **Table 20**).

Theme Title	Theme Description
Temperament	Two participants identified that the right mindset, or personality traits, could help them improve as a programmer – notably, these were related to the characteristic of creativity, but this theme could be expanded to include other temperamental features, such as patience.
Evaluation	Three participants described issues associated to the solution development and testing itself, implying there is a strong inclination towards programmers evaluating their quality of their programming as a result of their experience with developing past solutions. While there are five unique phrases associated to this category, one participant chose three different phrases from this category in their answer to this question which shows how much emphasis they place on testing, and the processes defined in testing. These participants believe that good programmers identify and resolve mistakes quickly and easily, and can detect bugs without too much issue which likely relates to the issues they have encountered that have impacted their self-confidence the most.
Computational Thinking	Our participants described various steps of problem-solving processes that relate to computational thinking, with a related theme of evaluation and testing. Participants generally used the phrase 'problem solving', but some focused on differing aspects of problem solving in a way that would indicate a different emphasis on what they perceive as an integral part to programming. Participants focused on the task analysis, solution analysis and testing, self-evaluation and their overall approach to the task when discussing this theme.
Communicating with peers	Communication was only used by one participant, who also used Teamworking. This suggests that their focus of the term communication is on communicating to like-minded individuals about programming concepts.
Programming knowledge	Two participants wished to focus on experience of programming and being able to recall programming knowledge as important parts that they needed to improve on. While it is interesting that there is an inclination towards believing that experience will make you a better programming, it may also be implicitly linked to computational thinking and evaluation theme in that patterns in computational thinking can suggest that drawing upon experience and knowledge is a good start when attempting to solve a computational issue. That said, these participants seemed to distinguish between programming knowledge and computational thinking and evaluation, hence, a new theme has been created from these datapoints.

Table 20: Thematic Definitions for the background questions related to quality: 'What qualities does a programmer require (in your opinion)?'; 'Which qualities of a programmer do you feel you have?'; and 'Which qualities of a programmer do you feel you need to improve on?'

Participants were asked to name what the most important aspect of being a programmer was after they had reflected upon their own qualities and experiences. The following figure illustrates that,

despite 100% of participants earlier mentioning the problem solving was important, only 45.45% of participants suggested it was the most important aspect (see Figure 113).

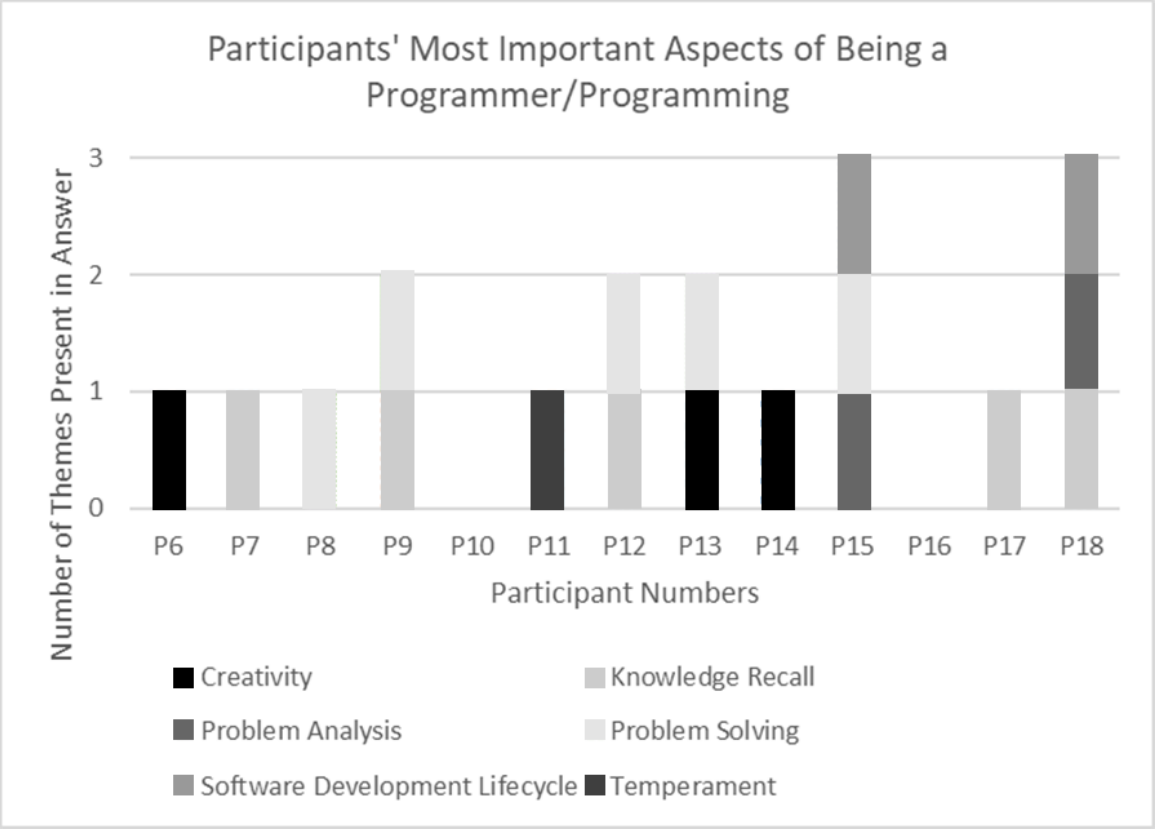


Figure 113: What is the most important aspect of understanding programming (in your experience)?

It is interesting to note that participants rarely identified an aspect that belonged only to one classification; for example, 'creativity' also included the phrase 'abstract thinking' which belongs to 'problem solving'. Knowledge recall encompassed a variety of different aspects – ranging from remembering aspects about how to structure a class and knowing when certain syntax should be used for methods, to how to develop, test and refactor code and make "something useful". Interestingly, the parts of the software development lifecycle highlighted by the participants were requirements gathering, development, testing and maintenance – design was missing. This could be indictive of the perspective of an NP – that, as a programmer, your primary goal is to write code and not consider the bigger picture. This was supported by Kauffmann's (2011) findings as they found that NPs were focusing on the intricacies of code rather than the design aspect. Temperament was the ambiguous classification – the participant suggested 'motivation' was the most important part, implying that there also needed to be innate desire to be a programmer as well as having the correct temperament for programming.

Participants were also asked to document their own steps to solving programming tasks; this was intended to identify their process, so that their recorded process could be compared to their perceived one. Participants all had unique ways of wording their steps, and therefore they were coded and grouped into themes based on the essence of the step (see Table 21).

	1	2	3	4	5	6	7	8	9	10	11	12	13
Read the task							1						
Read the requirements										1			
Understand the scenario			1						2				
Understand the task												1	
Try to understand what is being asked										2			
Identify unknowns in the scenario									3				
Identify classes and attributes from the task													1
Highlight key points												2	
Highlight requirements												2	
Visualise the problem								1					
Visualise the end product										3			
Understand what caused the problem								2					
Understand the problem				1					1				
Reword the part of the task I am not confident in												4	
Define most important elements						1							
Identify associations between code elements													4
Create skeleton code													2
Identify what needs to be done	-	1	-	-		-	-						
Understand how the solution would work								3					
Figure out a general solution	-	-	-	2		-	-						
Write a prototype						2	-						
Start to fill out code													3
Create a plan	-	-	2	-		-	-						
Split into small parts	1	2	-	3		-	2		4				
Start at the beginning of the class							3						
Create instance variables							4						
Create the constructor							5						

Assess what classes are required	-	3	-	-		-	-						
Perfect the methods							6						
Consider the format of the code						3	-						
Create pseudocode	-	-	3	-		-	-						
Plan the rest of the code						4	-						
Do the quick easy parts	2	-	-	-		-	-						
Do parts of the task I am most confident with												3	
Try multiple times												5	
Solve each part one by one	-	-	-	4		-	-						
Build the classes	-	4	-	-		-	-						
Build the constructors	-	4	-	-		-	-						
Think of alternate ways to represent solutions in the code								5					
Write the code						5	-	6		4			
Implement code	-	-	4	-		-	-						
Plan how to do the harder parts	3	-	-	-		-	-						
Assess what parts need to communicate with each other	-	5	-	-		-	-	4					
Identify what the components are doing								5					
Hand-written notes	3	-	-	-		-	-						
Analyse what the objects did to solve the problem								6					
Diagrams	3	-	-	-		-	-						
Build Accessor Methods	-	6	-	-		-	-						
Build Mutator Methods	-	7	-	-		-	-						
Write as much as I can	4	-	-	-		-	-						
Reflect on the written code									6				
Improve it	-	-	5	-		-	-		7				
Test it	5	8	-	5		-	-			5			
Run it	-	8	-	-		-	-						
Adjust code	6	9	-	-		-	-						
Modify code										6			
Simplify code						6	-						
Refactor code													5
Repeat	->5	->1	-	-		-	-						
Get frustrated										7			
Ask for help										8		6	

Table 21: How participants answered the question of "Can you describe the steps you take to solve a programming task? " The number represents the step order for how they would approach an issue

6.2.3 Movement Frequency and Types of Movements Made

All participants made more movements in CP2 than CP1, which is to be expected considering the number of pieces. However, the types of movements did vary between participants – due to the difference in camera angle on the video recordings, the researcher could clearly see that participants did make micro-movements towards pieces – known as a 'shifting' movement – which usually signified that participants were considering removing or putting back a piece, but decided not to. In the context of participants repeatedly shifting pieces upward, it was noted that the audio recordings indicated that this is the action participants took when they were 'testing' the pieces – as in, they wanted to check that all of the pieces made personal sense to them and were checking each line carefully. The shifting movements were also catalogued, but unfortunately, due to the angle of the camera, the workspace and grouping movements could not be analysed to investigate whether a machine learning algorithm could correctly identify the number of groups made.

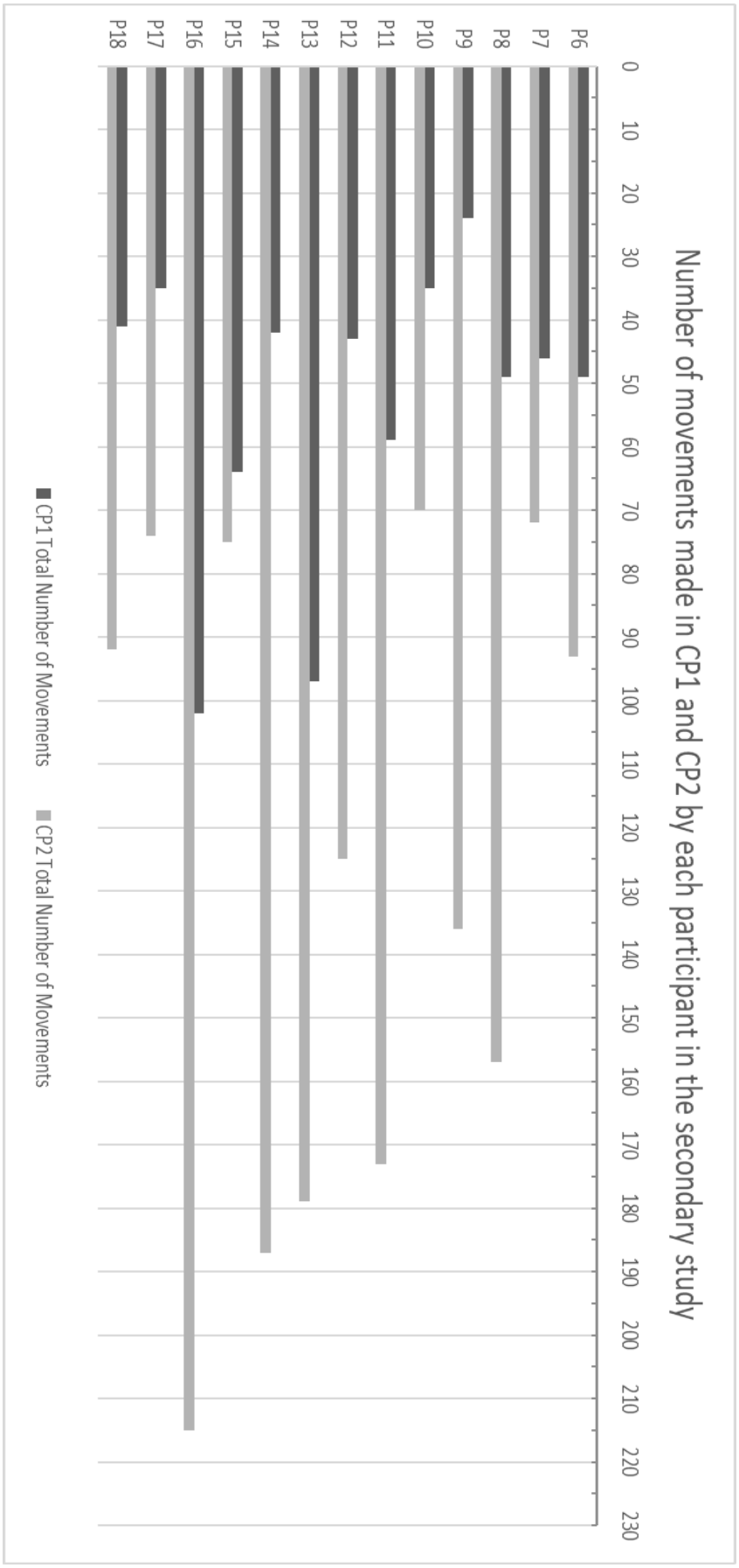


Figure 114: Number of movements made by participants in the secondary study for CP1 and CP2

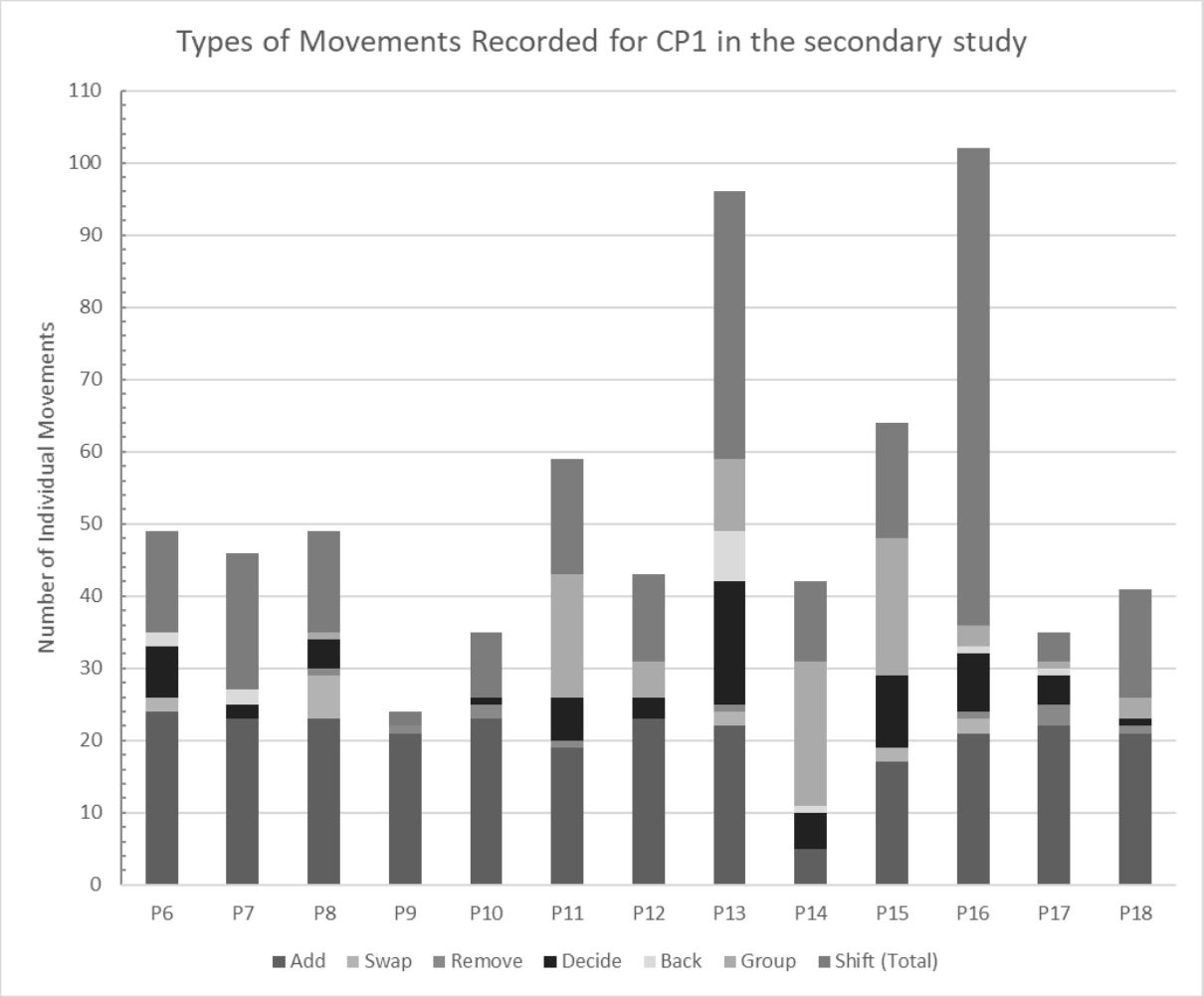


Figure 115: Types of movements made by participants in the secondary study for CP1

The types of movements indicate that participants typically did one movement per piece in CP1, but not in CP2, except for P14 who chose to group the pieces and then add them in sections to the final solution space. The workspace was again not prominent in all participants – but still did occur in 4 participants quite clearly. The reasonings behind groupings varied, some participants wanted to rearrange the pieces in a way that made sense to them before they added the pieces to the final solution space, others wanted to group the pieces that did not know what to do with together, and others wanted to just group the end brackets to make it easier to close off the various elements of the class.

For CP1 only due to time constraints, the total number of movements were split and the excess movements recorded for each piece (see Figure 116, Figure 117 and Figure 118).

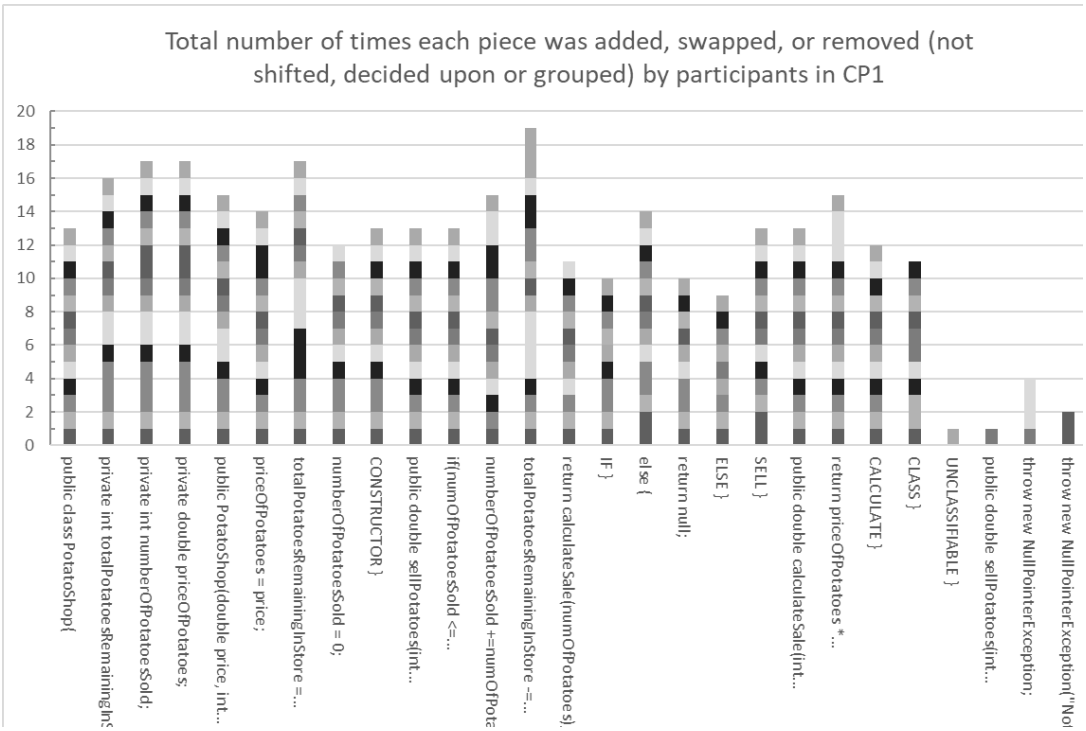


Figure 116: Number of add, remove, and swap movements (not decide, back and shifts) recorded in CP1 of the secondary study

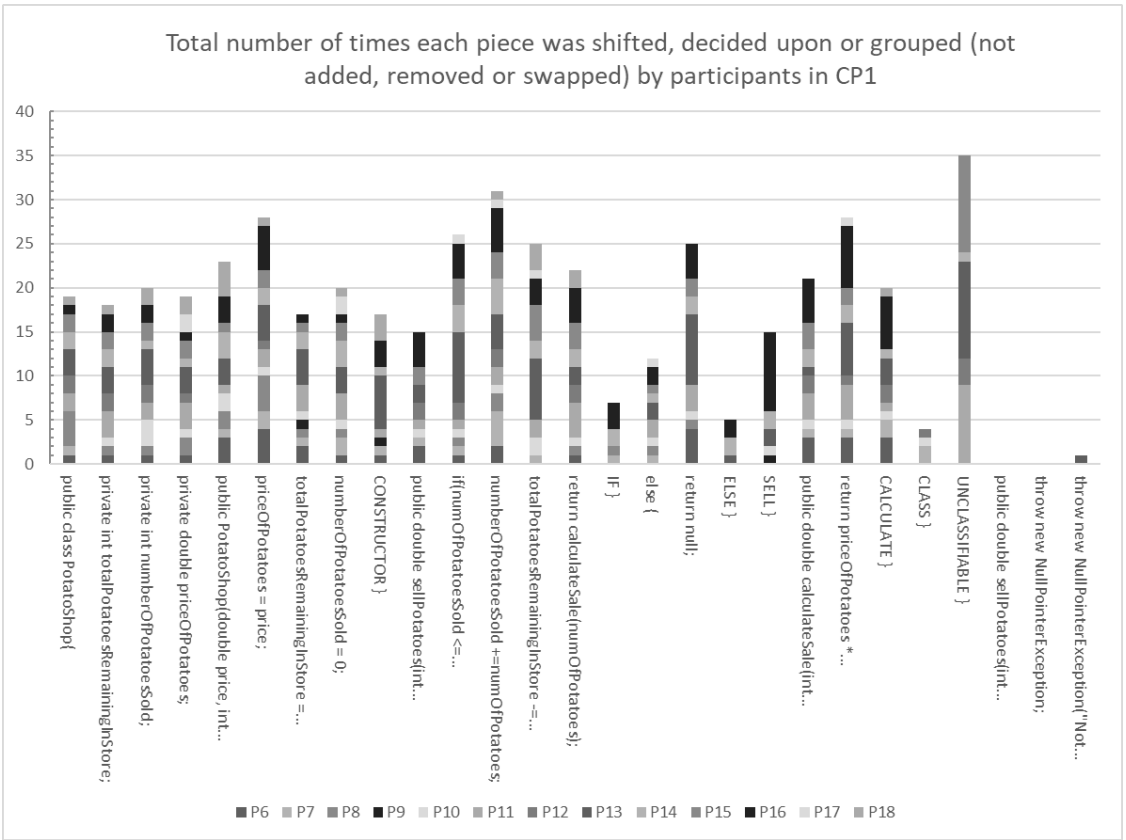


Figure 117: Number of shift, decide and grouped movements (not add, remove and swap) recorded in CP1 of the secondary study

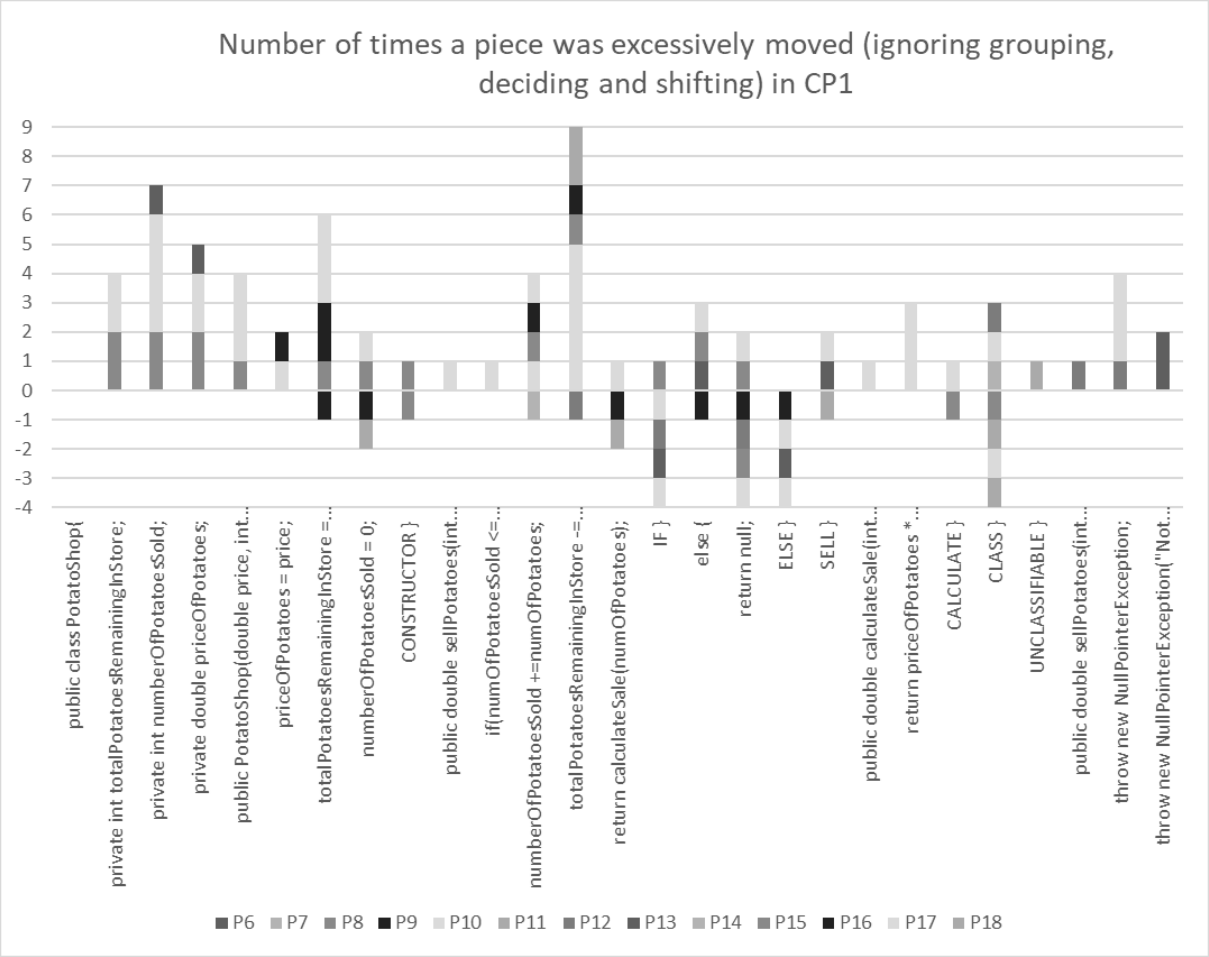


Figure 118: Number of excess movements recorded in CP1 of the secondary study

The CP2 movements showed a stark contrast in terms of shifting movements; when compared to CP1, participants demonstrated either shifting aspects or limited shifting whereas CP2 did not show such a clear-cut difference. The participants that used shifting did so often to indent, or realign, their code and did also do so for testing purposes. Another reason shifting occurred was when participants did not think their method looked right and that they wanted to check each piece. Another notable difference is that the remove movement was more prominently seen – this could be because there are more pieces for the participant to move in comparison to CP1, but it commonly meant that the participant was uncertain about the piece's context or the concept behind the piece. P15 prematurely discontinued due to frustration and becoming overwhelmed by the number of pieces presented to them – interestingly, they chose to group the pieces as their main movement but then failed to be able to find the pieces. Not being able to find grouped pieces may insinuate that grouping is a mechanism that occurs when the participant's experiencing cognitive overload and the grouping attempts to reduce the amount of cognitive load by assigning pieces to groups prior to moving them and allowing them to locate the pieces. In P15's case, it is possible that they were

overwhelmed as they struggled to complete a coherent solution for CP1, and for CP2 were unsure of how to start a class.

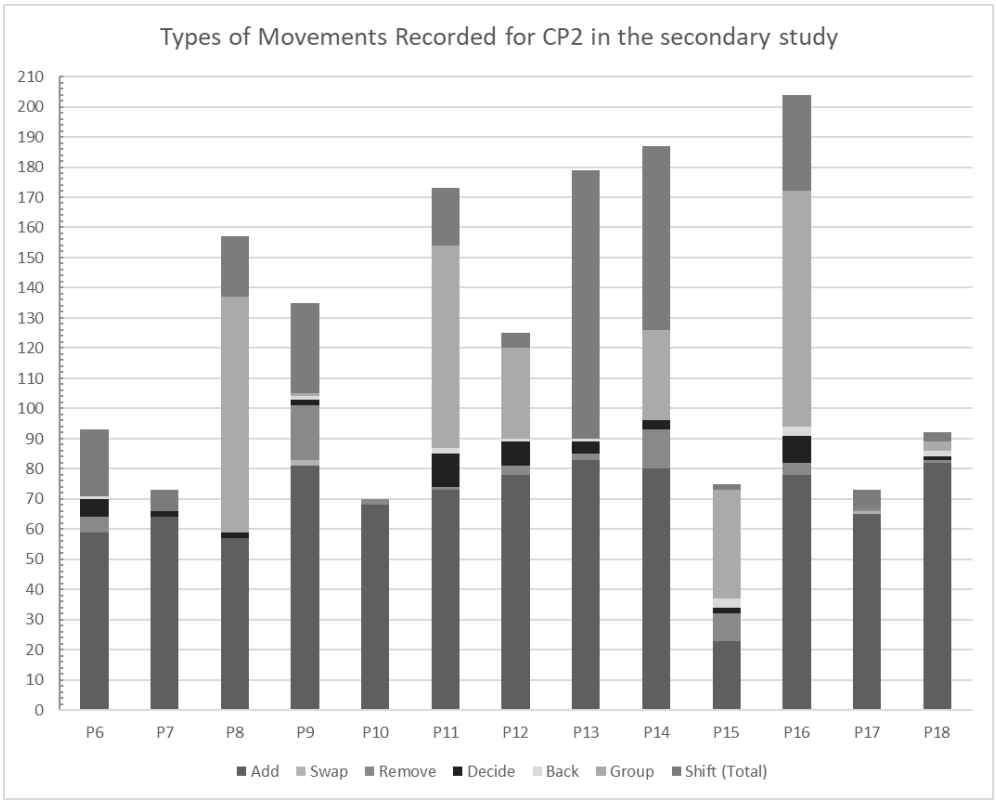


Figure 119: Types of movements made by participants in the secondary study for CP2

There were noticeably a higher variety of movements in CP1 than in CP2 when examining Figure 120 to Figure 132, with participants generally performing a higher percentage of more add movements in CP2 than CP1 – however, there are more pieces in CP2 than CP1.

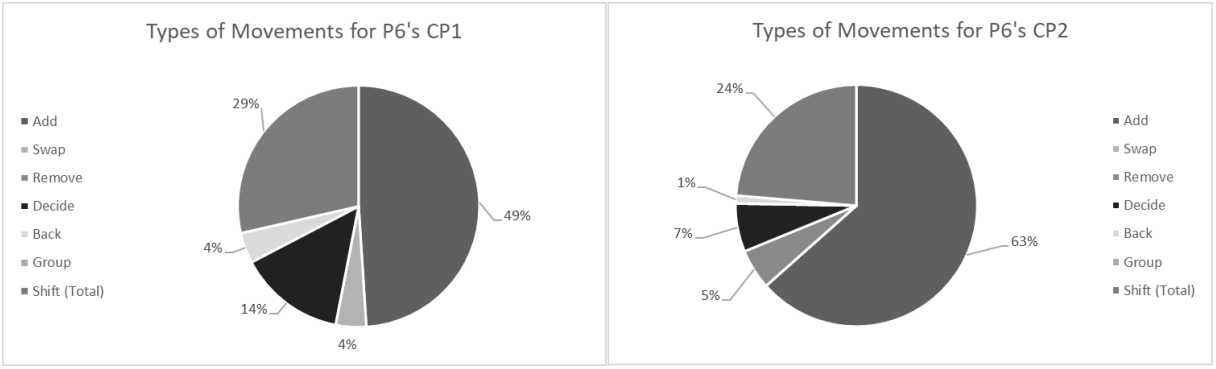


Figure 120: Percentages of the types of movements made by P6 for CP1 and CP2

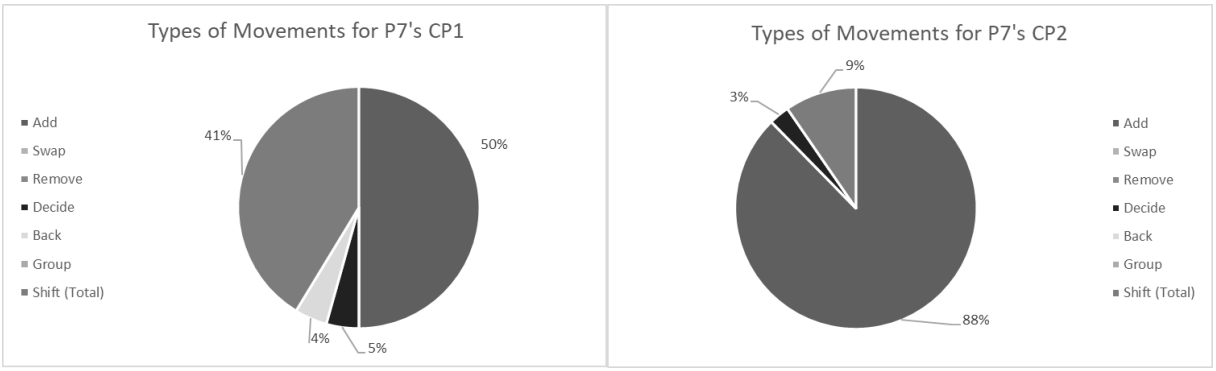


Figure 121: Percentages of the types of movements made by P7 for CP1 and CP2

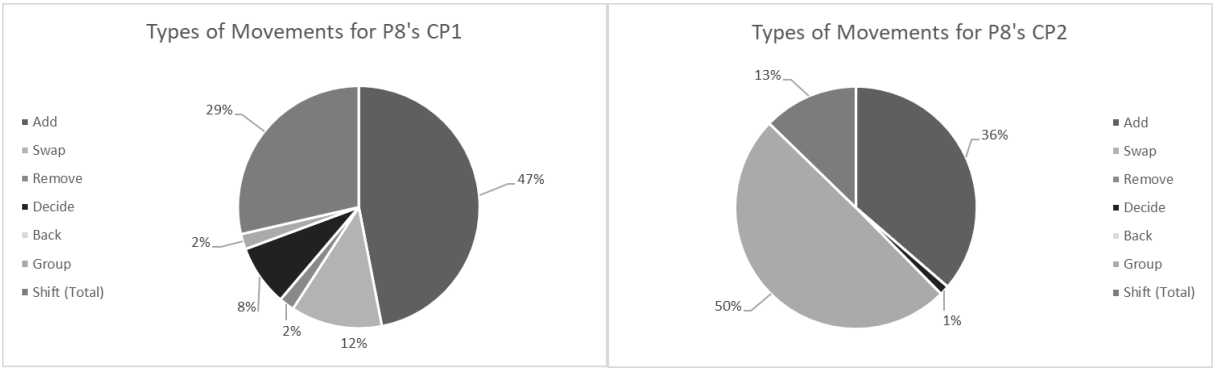


Figure 122: Percentages of the types of movements made by P8 for CP1 and CP2

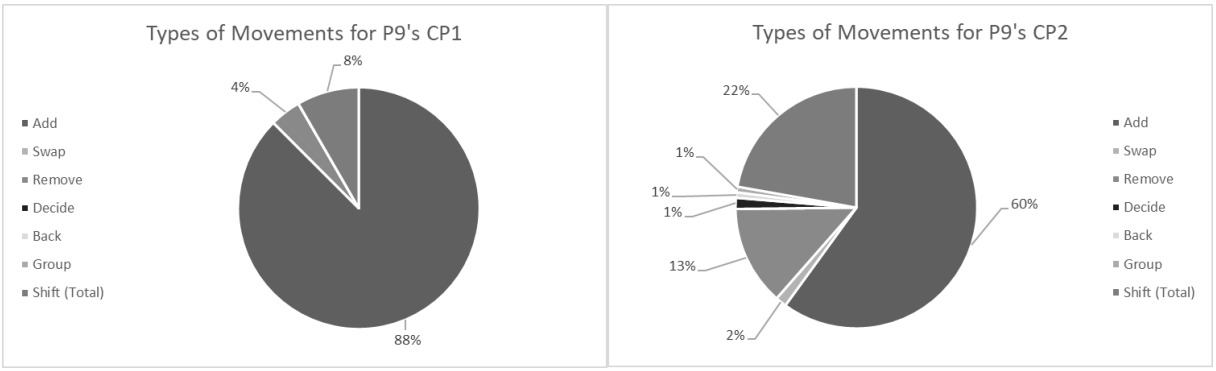


Figure 123: Percentages of the types of movements made by P9 for CP1 and CP2

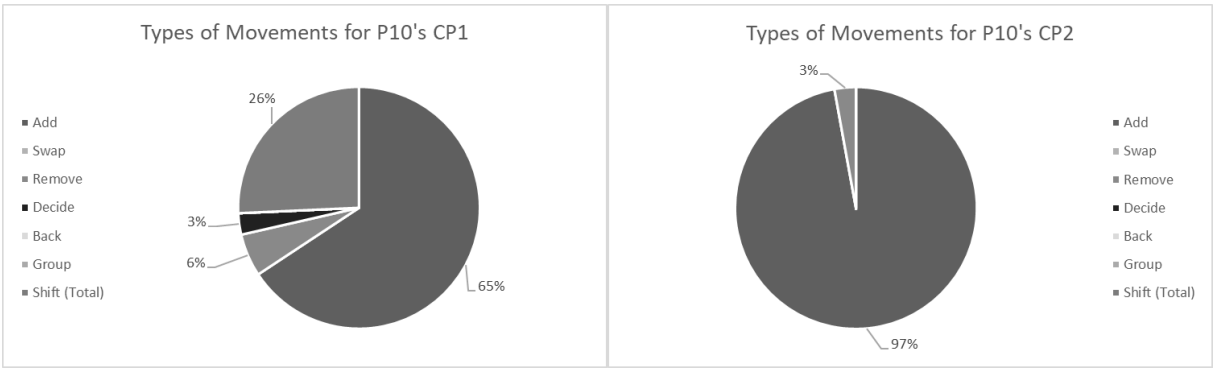


Figure 124: Percentages of the types of movements made by P10 for CP1 and CP2

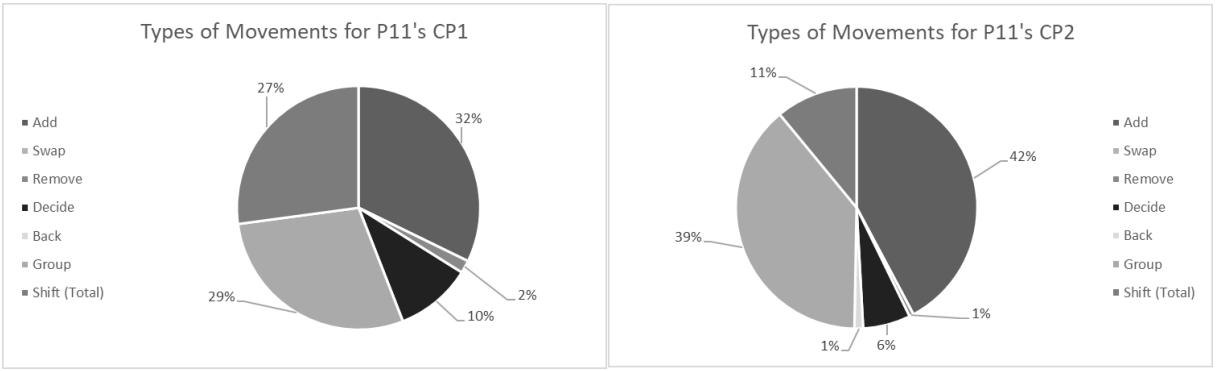


Figure 125: Percentages of the types of movements made by P11 for CP1 and CP2

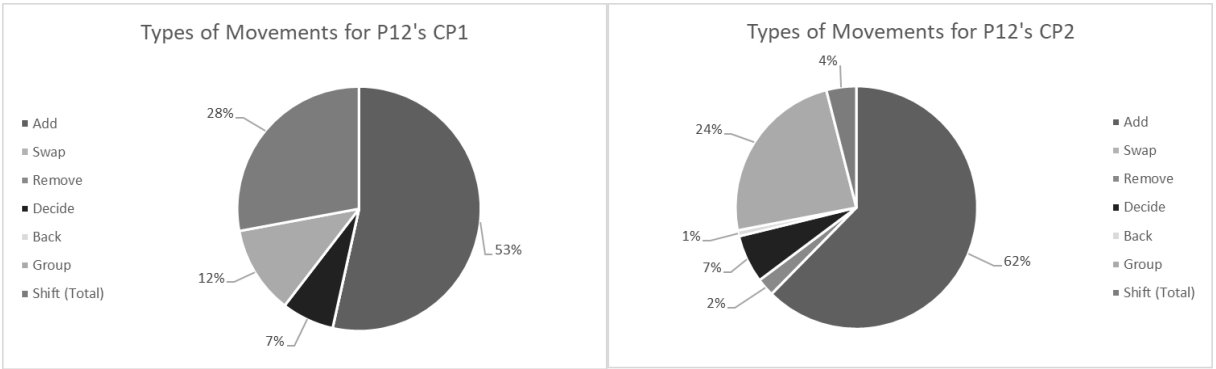


Figure 126: Percentages of the types of movements made by P12 for CP1 and CP2

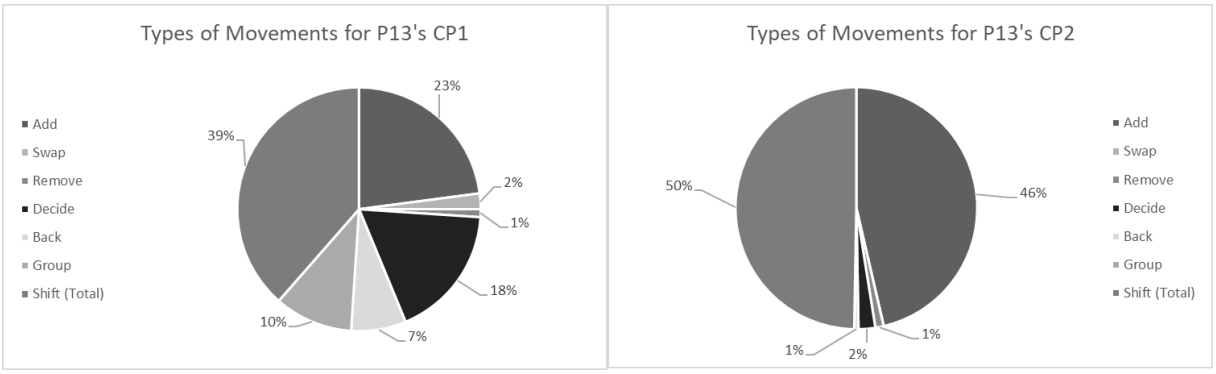


Figure 127: Percentages of the types of movements made by P13 for CP1 and CP2

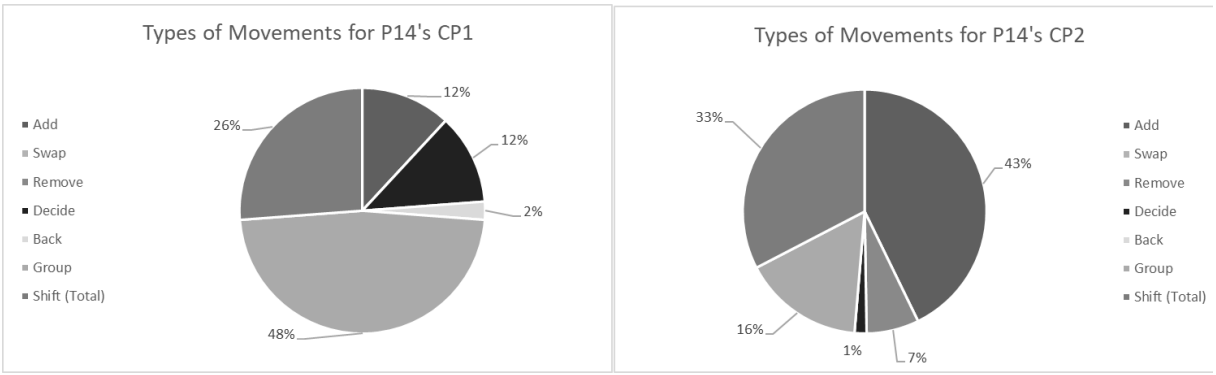


Figure 128: Percentages of the types of movements made by P14 for CP1 and CP2

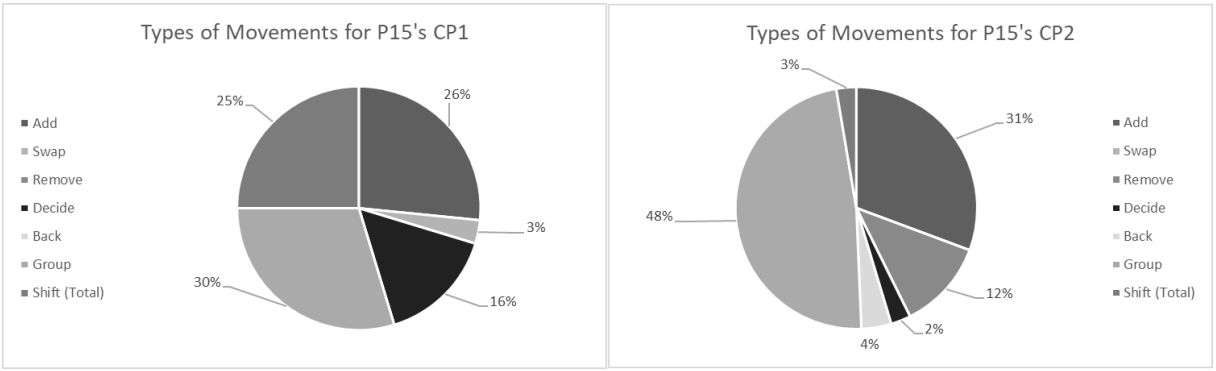


Figure 129: Percentages of the types of movements made by P15 for CP1 and CP2

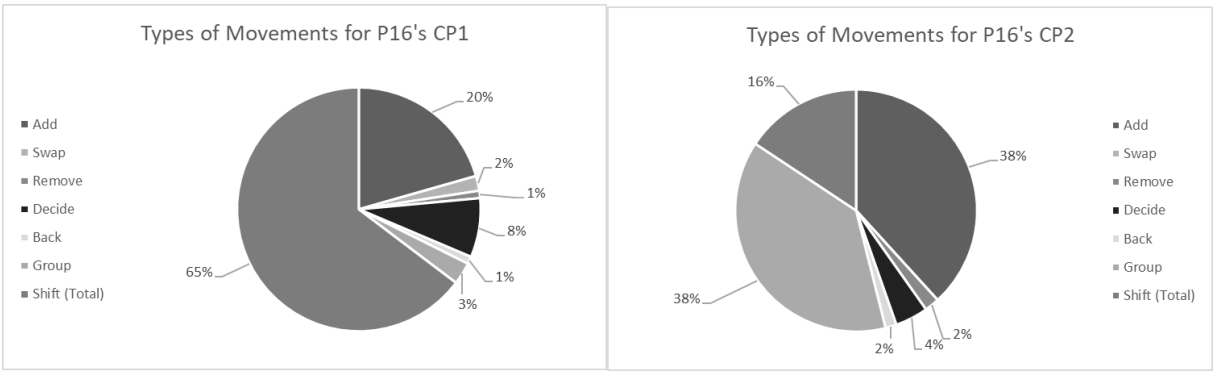


Figure 130: Percentages of the types of movements made by P16 for CP1 and CP2

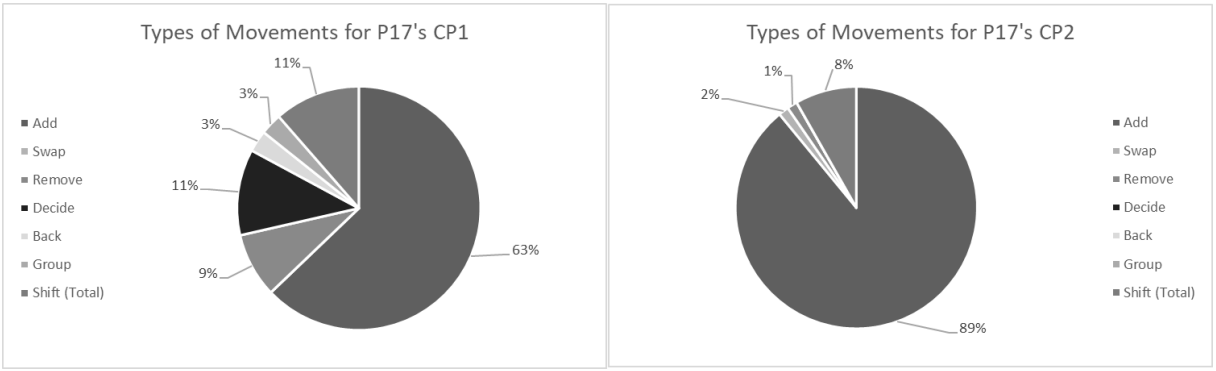


Figure 131: Percentages of the types of movements made by P17 for CP1 and CP2

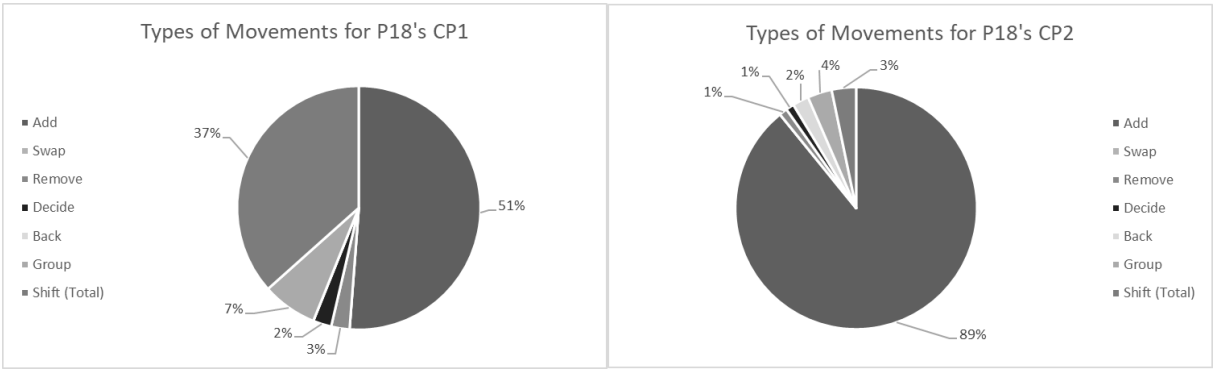


Figure 132: Percentages of the types of movements made by P18 for CP1 and CP2

‘6.2.4 Final Solutions

The figures below illustrate the final, submitted solutions for each of the participants the frequency of mistakes in the final solutions were greater than that of the pilot study.

Figure 133: P6's final solution for CP1 (left) and CP2 (right)

Figure 134: P7's final solution for CP1 (left) and CP2 (right)

Figure 135: P8's final solution for CP1 (left) and CP2 (right)

Figure 136: P9's final solution for CP1 (left) and CP2 (right)

Figure 137: P10's final solution for CP1 (left) and CP2 (right)

Figure 138: P11's final solution for CP1 (left) and CP2 (right)

Figure 139: P12's final solution for CP1 (left) and CP2 (right)

Figure 140: P13's final solution for CP1 (left) and CP2 (right)

Figure 141: P14's final solution for CP1 (left) and CP2 (right)

Figure 142: P15's final solution for CP1 (left) and CP2 (right)

Figure 143: P16's final solution for CP1 (left) and CP2 (right)

Figure 144: P17's final solution for CP1 (left) and CP2 (right)

Figure 145: P18's final solution for CP1 (left) and CP2 (right)

As shown, participants created a variety of solutions with the allowed customised pieces – but generally, due to the effort of creating a piece and thinking that the piece must not be necessary to complete the puzzle if it does not already exist, participants did not choose to do this.

6.2.5 Post-Study Questionnaire Results

Surprisingly, 84% of participants found that the observer had accurately identified their approach to programming, and 69% agreed that their understanding of programming had been correctly identified by the observer. 23% of participants believed that the understanding did not reflect their entire understanding of programming as some felt that the challenges of the puzzles were too easy to pick up on the issues they struggled with. 8% of participants did not respond and can be assumed to believe that the study was not accurate in their understanding but that they did not wish to upset the observer. For the 8% of participants who claimed the approach was partially inaccurate, this was due to them not being able to look at the full Oracle documentation on the Date library or had the chance to revise Java documentation prior to the session.

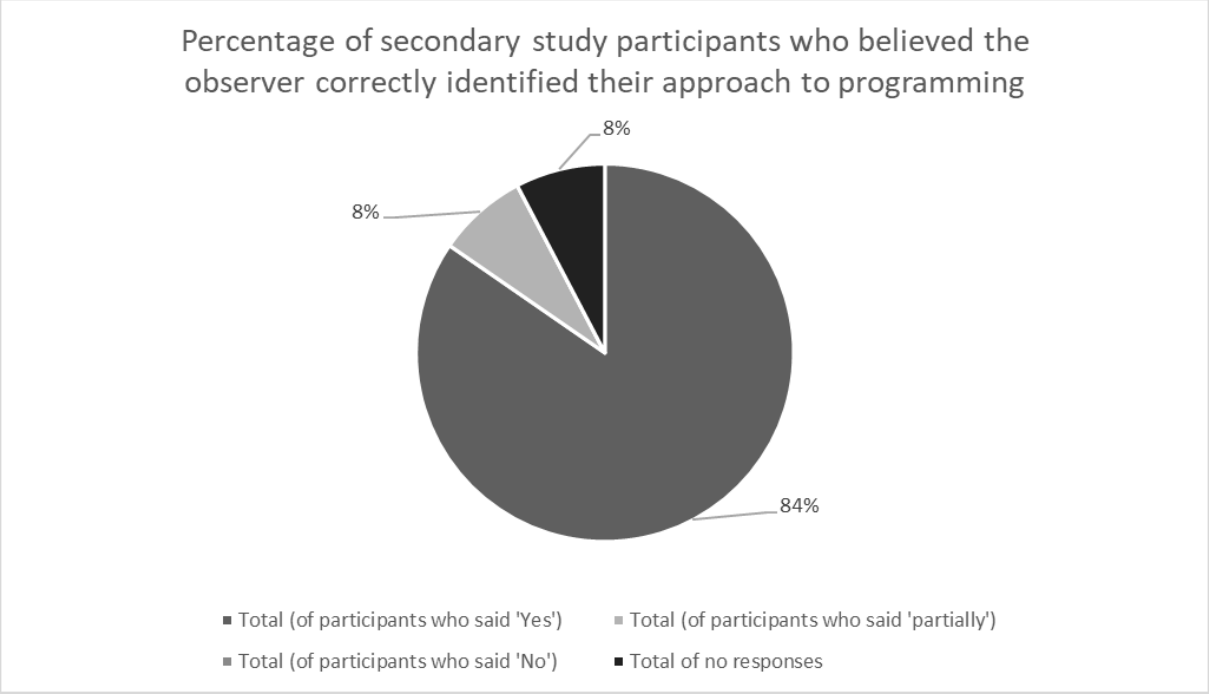


Figure 146: Collective answers from participants during the Post-Study Questionnaire: 'Do you feel that the study accurately portrayed your approach? Why do you feel this way?'

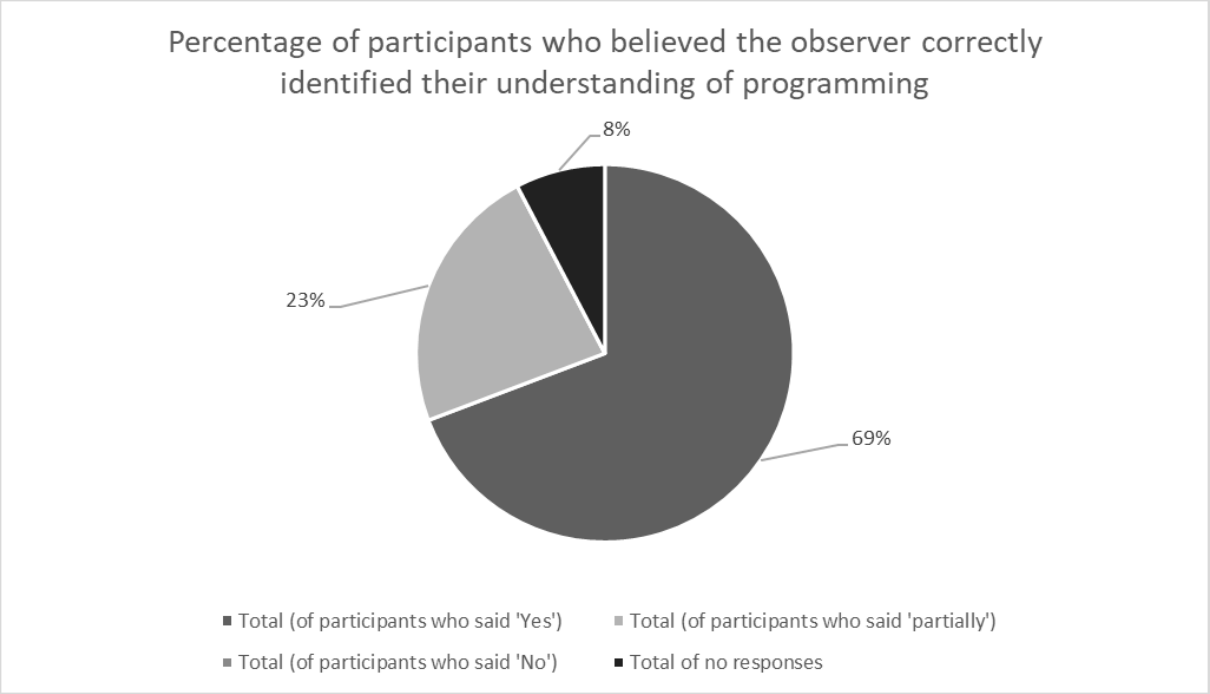


Figure 147: Collective answers from participants during the Post-Study Questionnaire: 'Do you think the analysis did reflect on your understanding or were the findings inaccurate? (Please be honest)'

These are fair criticisms, as in a realistic environment, participants would be able to search through documentation – that said, the provided after method documentation and explanation of the Date class was given to participants which raises the question of what the participant would exactly do differently with the documentation if they had full access to the Oracle website. Perhaps, though this is just speculation, the participant wanted to try out coding associated with the Date object and that is part of their approach which they would not be able to simulate.

The main issue here is how high the perceived accuracy was regarding the tool being used by the observer straight after the end of CP2 to identify the understanding and approach in real time, without the pre-processing of data. This goes against the expectations of the pilot study, as it was deemed improbable for the observer to be able to process so much data within minutes and arrive at an accurate conclusion. That said, even though the observer was busy recording the participant and conducting the study, from watching the participants' movements and listening to their reasoning for those movements, the data suggests that this, alone, is sufficient to diagnosing NP understanding. That said, there are drawbacks to this process – even if the data obtained from the observer is accurate, the session per participant lasts approximately 40-60 minutes – which, is an unrealistic amount of time for a practitioner to spend per CS student on an average degree course. Perhaps enough information could be obtained from using one puzzle instead of two – which would make the total time required to typically less than 20 minutes and slightly more feasible for a practitioner to be able to achieve in an office appointment with a student. On the other hand, a diagnostic tool for one

or two NPs who are genuinely struggling, and in need of programming support services, seems like a more feasible application of this tool as identifying their understanding can lead to tailored support.

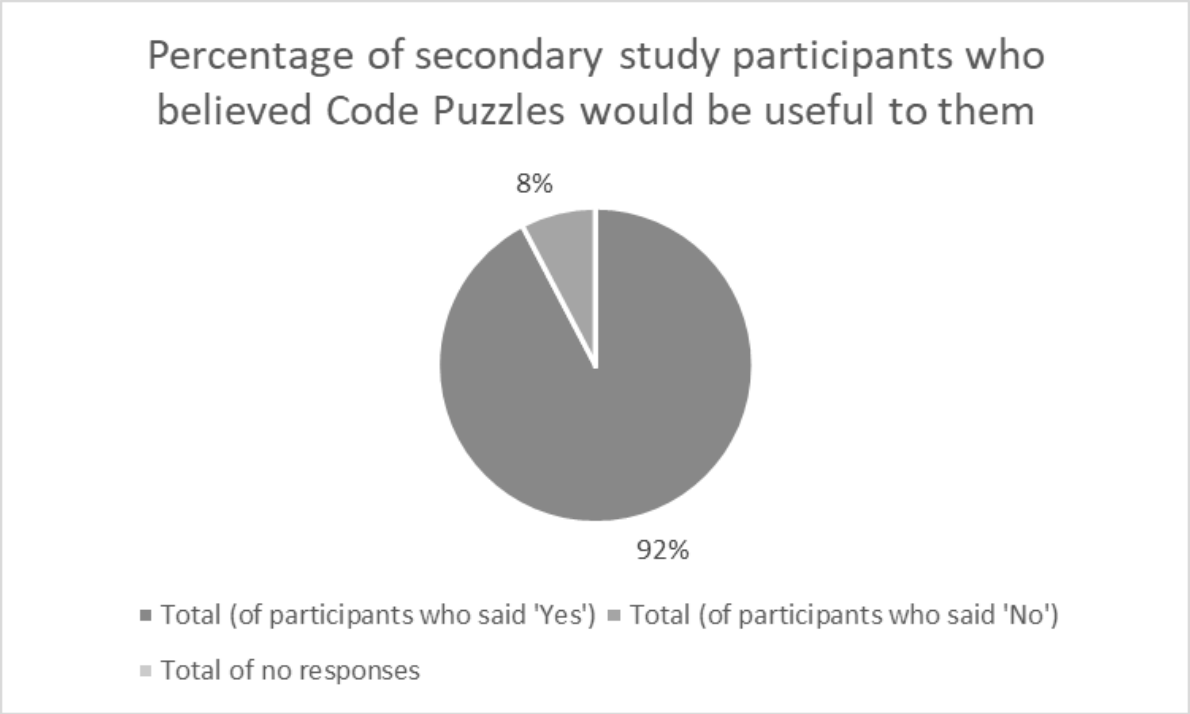
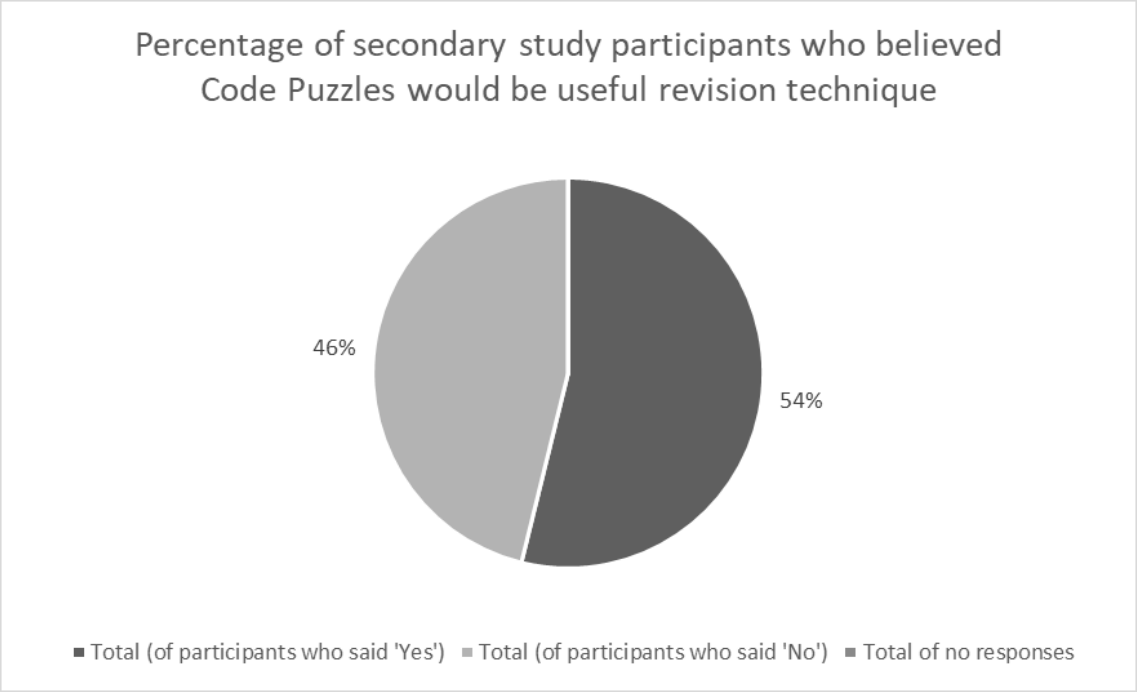


Figure 148: Percentage of participants who believed code puzzles would be useful to them.

When asked 'Do you believe the Code Puzzles would be of use to you?', 92% of participants agreed that code puzzles would be useful to them personally if the program could identify their understanding or approach. 69%% of participants believed that code puzzles would be beneficial to use in conjunction other revision tools, whereas 54% of participants believed that code puzzles could be a useful revision tool by itself. From this generally positive feedback, participants enjoyed using code puzzles and found the experiment useful for understanding their own understanding – 2 participants even commented that they knew what to focus on in order to improve themselves as programmers in the open-ended questions from the post-study questionnaire. This, in itself, implies there is great potential for code puzzles to be used as discussion points to allow programmers to explain their difficulties to tutors without the communication barrier being present.



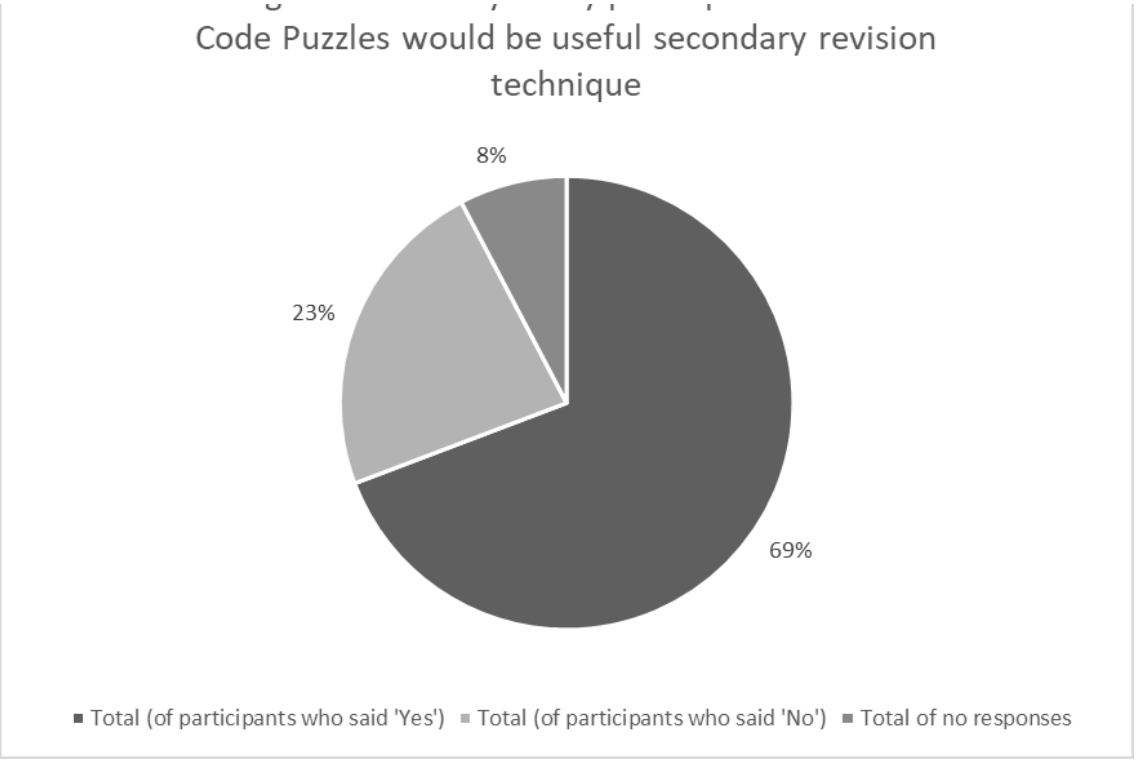


Figure 149: Percentage of participants who believed that code puzzles would be a useful revision technique

In light of the research performed by Denny et al., it seemed prudent to ask whether the participants felt that tools such as Code Puzzles could be used as a revision technique – it is clear that the code puzzles were not a popular concept as a revision technique, and therefore the 92% that claim code puzzles would be 'useful' to them did not implicate that it would help them learn programming

concepts effectively. The definition of 'useful' was not defined by most participants, perhaps alluding to social desirability bias in the data – however, it seemed from the comments that participants did feel that it was an enjoyable exercise that allowed them to reflect on their own approaches and improve what to revise.

6.3 Chapter 6 Summary

This chapter presented and evaluated the findings for a secondary study that aimed to replicate the pilot study with a larger sample size and refined study protocols (i.e., observer having an observer script, yellow card for the participant wanting to ask a question to the observer, red card for participant ending the study).

The secondary study results suggest that using paper-based Code Puzzles as a diagnostic tool is a potentially effective way of learning more about the understanding of the NP using the tool. This study examined NP preconceptions to tasks prior to commencing the puzzles, and what qualities they believed an 'expert' programmer would have. This study also investigated how NPs would describe their approach to programming and compared this to how they completed the puzzles to see whether there is any similarity and found that most participants felt their approaches and understanding of the programming concepts found in the puzzle were accurately identified.

The secondary study also documented more instances of participants using the grouping type of movement.

Overall, this chapter concludes that paper-based Code Puzzles show promise for diagnosing understanding in NPs, however, there are concerns about how this tool would effectively translate to an electronic device and the premise for the tertiary study is to see whether this translation can be incorporated in a pre-existing learning environment.

Chapter 7. Tertiary Study: Observing Coding on a Realistic Learning Environment

This study investigated whether the secondary study methodology could be transferred effectively to a realistic educational tool commonly used to teach in a university setting (Blackboard Collaborate Ultra). This study offered the opportunity to compare the results of the pilot and secondary study to whether the same degree of analysis could be obtained from observing the student typing on a computer or IDE as only their submitted lines of code and the audible dialogue were obtained – not their movements. Therefore, the tertiary study establishes a comparable baseline to the previous studies (see Table 22).

Tertiary Study's Research Question
How effectively can we identify the novice programmers' understanding by observing the way they code via an online educational environment?

Table 22: Tertiary Study's Research Question

Originally, it was intended that the tertiary study should mimic the secondary study as much as possible and wanted to observe students interacting with an IDE. However, this study was impacted by the COVID-19 policies; therefore, it had to be conducted over Blackboard Collaborate Ultra – a learning and teaching tool commonly used by UK-based universities at the time of investigation. While this tested whether the study could be repeated using a realistic online teaching tool, movement data was impossible to record in the previous format as it was unclear when a participant would 'pick up' a piece and pieces could not be moved due to the interface restricting the puzzle pieces to appear statically (see **Figure 150**).

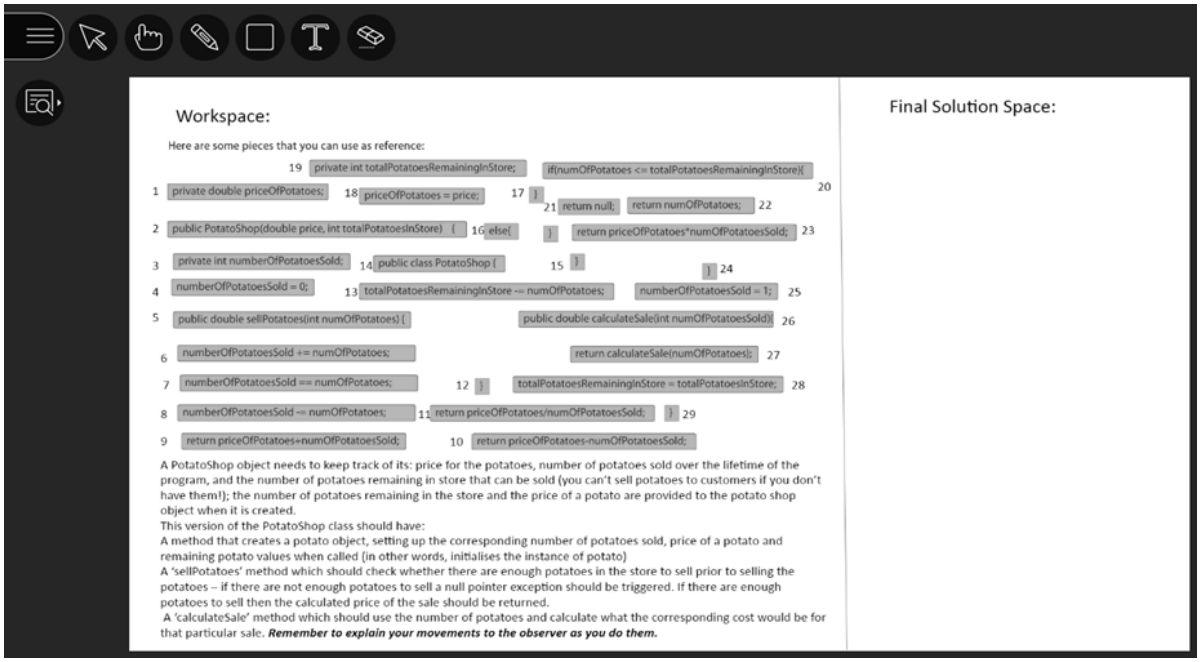


Figure 150: Tertiary Study interface presented to participants (including the rubber icon and T icon that caused the issues)

As a result, this study aims to provide a comparison point for the data collected from previous studies as it allows for the observer to observe the participants interacting in a different way with the puzzle tasks. That said, only three participants were recruited for this study – however, this is still worthy of note as these three participants were of a similar confidence to previous participants, but they produced solutions that had far more mistakes than previous participants, despite the ability to type their own code.

A total of three first year undergraduate Computer Science students from one university volunteered for the pilot study; they had all completed the foundations in Java module and knew the basics of Java.

This study chose to carry forward the hypotheses from previous studies (see Table 23).

ID	Hypothesis	Did the pilot study support this?
H-1	Novice programmers of a similar level of understanding will share similar characteristics in their interactions with a particular line of code.	Supported by the pilot and secondary studies.
H-2	Novice Programmer interactions can be classified and categorised	Supported by the pilot and secondary studies.
H-3	Classified Novice Programmer interactions can be mapped to a level of understanding	Supported by the pilot and secondary studies.
H-7	The more incorrect placements of Code Puzzle pieces that novice programmers make, the easier it will be to indicate misconceptions.	Supported by the secondary study.
H-8	Novice programmers of a similar approach to coding will share characteristics in their understanding.	Supported by the secondary study.
H-9	Novice programmers will leave pieces that they are least confident about until last.	Supported by the secondary study.
H-10	Novice Programmers will prefer the Line-By-Line Code Puzzles over the Piece-by-Piece Code Puzzles.	Not supported by the secondary study.
H-11	Novice programmers will find Code Puzzles useful; this will be determined by whether the majority of participants (over 50%) find the Code Puzzle useful in the post-study questionnaire.	Supported by the secondary study.
H-12	Novice programmers will find Code Puzzles as a viable alternative to traditional revision methods; this will be determined by whether the majority of participants (over 50%) find the Code Puzzle useful in the post-study questionnaire.	Supported by the secondary study.
H-13	Novice programmers will find the Code Puzzle analysis accurate in regards to their approach; this will be determined by whether the majority of participants (over 50%) find the Code Puzzle useful in the post-study questionnaire.	Supported by the secondary study.
H-14	Novice programmers will find the Code Puzzle analysis accurate in regards to their understanding of the underlying concepts; this will be determined by whether the majority of participants (over 50%) find the Code Puzzle useful in the post-study questionnaire.	Supported by the secondary study.

Table 23: Tertiary Study's Hypotheses

It was determined from these hypotheses that the study would compare the results obtained from this study to previous studies, in which the accuracy of analysis and reasoning behind the typed code was crucial for comparison purposes. However, it was unknown prior to the study commencing as to whether the accuracy would be different or the same for this metric.

7.1 Methodology

3 CS undergraduate students at the end of their first year at Aston University volunteered to take part in the tertiary study in 2020, with the criteria that they must be an enrolled Aston University CS student that had attended at least one term's worth of Java Foundations module content.

7.1.1 Advertising and Recruitment Process

Students were recruited through a Blackboard announcement from a shared second term module that all CS and combined honours CS students had access to. No financial incentives were offered; however, the study was promoted as a revision opportunity to students.

7.1.2 Procedure and Data Collection

The tertiary study consisted of 60-minute to 90-minute observation sessions where participants were asked to create working Java class in two, separate paper-based Code Puzzles via Blackboard Collaborate Ultra in the presence of an observer. Participants were given the background questionnaire prior to the observation session but were given the tasks during the session to analyse before proceeding. Participants were asked to write their opinions of Task 1 and Task 2 prior to commencing the respective puzzles and asked to complete a post-puzzle questionnaire after each puzzle. Due to interface restrictions, Code Puzzle (CP1) presented 29 unmovable pieces in the style of a 2D Parson's Problem (where each piece correlated to one line of code) and Code Puzzle 2 (CP2) presented 59 one-word unmovable suggestion pieces on the left-hand side – including distractors – and participants were asked to type their answers on the right-hand side of the screen using the text tool. Participants were asked to complete two Code Puzzles related to creating a Potato Shop Java class (CP1) and a Potato Java class (CP2) (see Appendix 11.1.1, Appendix 11.1.2, and Appendix 11.1.3 to view what the interface looked like).

CP1 was always completed before CP2, as randomising the order was not possible with the sample size. For both puzzles, participants were asked to create a "working" class using the puzzle pieces that fit the requirements of the task description while adopting a think-aloud protocol. After each Code Puzzle, participants completed a post-puzzle questionnaire which asked them to rate their

confidence on how well the solution would work on a 5-point Likert scale and rate how difficult they found the puzzle on a 7-point Likert scale. P19, P20 and P21 experienced feedback after each puzzle due to the limitations of recording Blackboard Collaborate sessions.

All participants' video recordings were successful, and every questionnaire was completed aside from P21's post-study questionnaire which was never returned.

7.1.3 Data Analysis

In comparison to previous studies, there was no split between the audio and video recording and customised pieces did not require the observer to stop the recording to go and make the requested pieces for the participant as the participant was typing text that would eventually appear on-screen to the observer once the participant had pressed enter. Unfortunately, the restrictions of the interface meant that the process of typing – or their keystrokes – was not a collectable set of data, and that only the text they believed was correct, and not their process to making the line of code, was viewed by the observer. The concept of analysing the time difference between pieces also required a different metric as there is no 'pick up' or 'hold' aspect to the pieces and therefore the calculated time intervals used the time between edits that appeared on screen. The video clearly showed where participants placed pieces, meaning that the audio transcripts do contain anonymised print screens from the Blackboard Collaborate sessions which was useful when memoing and encoding the transcripts.

7.2 Results

7.2.1 Time Observations

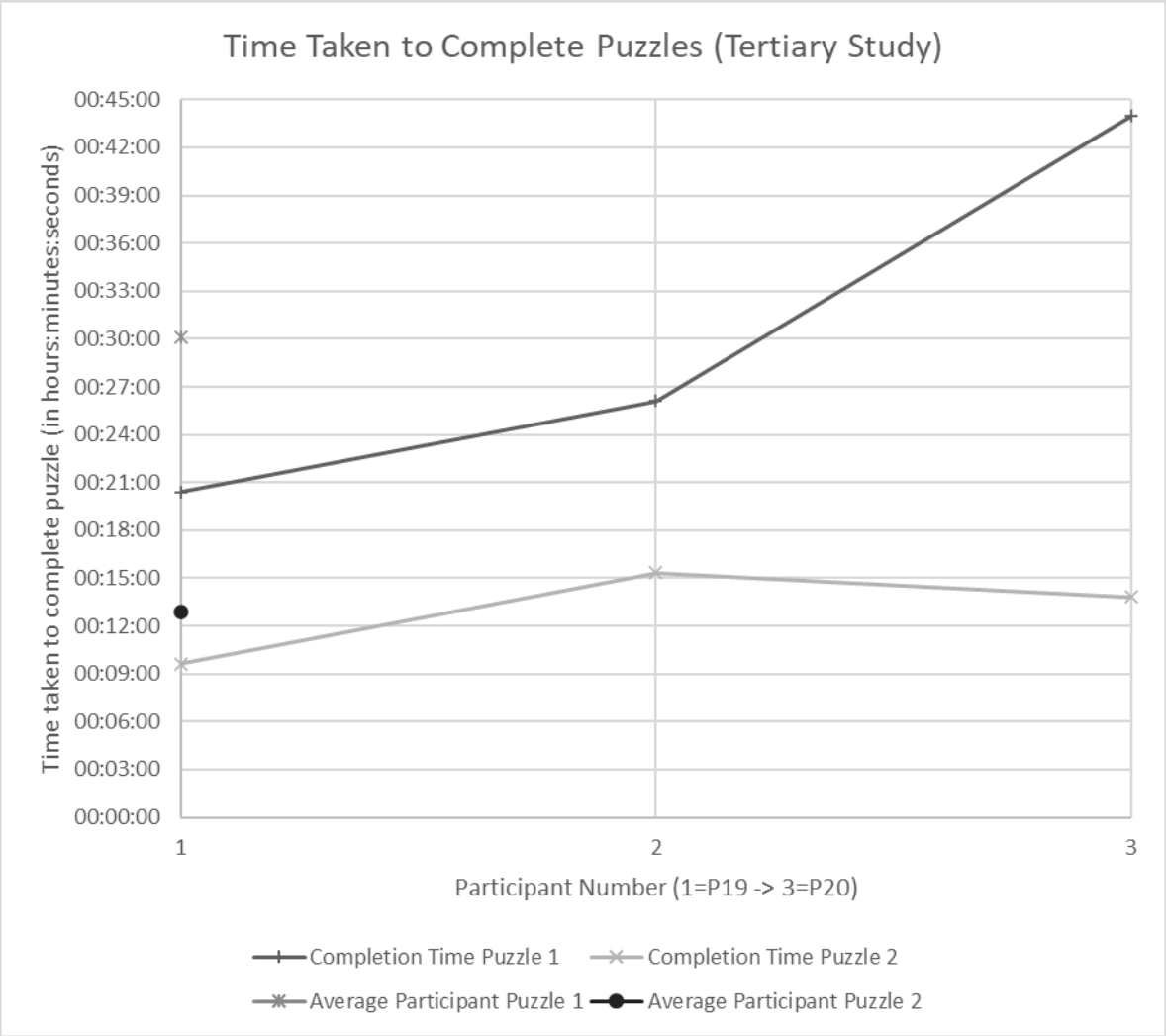


Figure 151: Scatter graph for time taken to complete the puzzle by each participant (Puzzle 1: range = 20:23-43:57, M = 30:08, SD = 12:18|| Puzzle 2: range=09:37-15:20, M = 12:55, SD = 02:58) for the tertiary study.

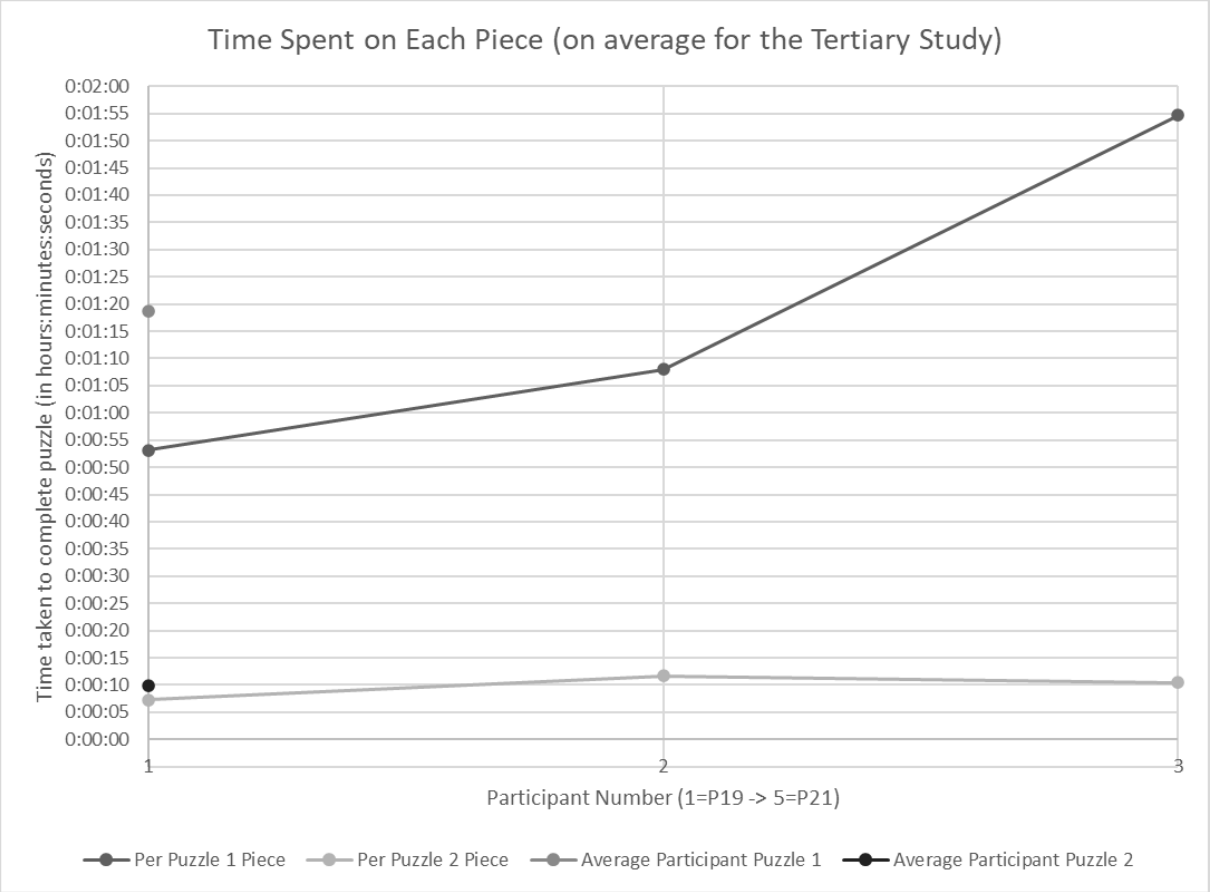


Figure 152: Scatter graph for the average time spent on each piece per puzzle (Puzzle 1: range = 00:09-01:55, M = 01:19, SD = 00:30|| Puzzle 2: range=00:03-00:12, M = 00:10, SD = 00:04) for the tertiary study.

Participants, on average, took longer to complete Puzzle 1 and Puzzle 2 than the average participant in the pilot or secondary studies. This was likely due to the aforementioned interface issues, alongside participants needing to type out the code instead of simply moving a pre-written piece of code across. This suggests that puzzles would be a more time efficient way of observing a novice programmer than watching them code in real time and therefore a more feasible way of studying a novice programmer.

As shown, participants completed CP2 quicker than CP1 and spent less time on each piece of CP2 than CP1.

7.2.1.1 Code Puzzle 1 Time Intervals

The time intervals were calculated by the time taken for the pieces to appear on screen. P19 spent the most time on creating the calculateSale method; they struggled to identify which parameter to use for the method and initially wrote 'potatoStock'. Likewise, there was a confusion between the

two methods – sellPotatoes and calculateSale. It is possible that the participant would naturally combine the two and having the task specify the two methods separately contributed to the issues (see **Figure 153** and Figure 154).

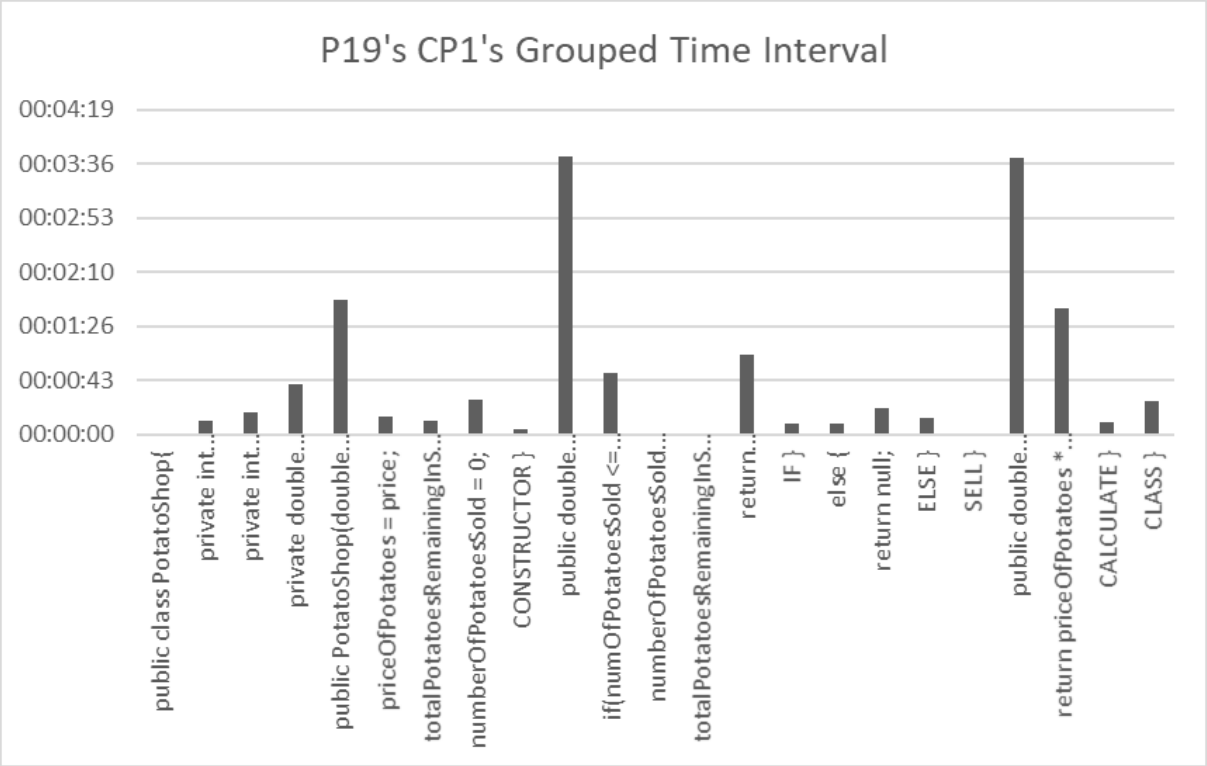


Figure 153: A bar chart of P19's Grouped Time Intervals – i.e., the sum of the time taken to place each piece

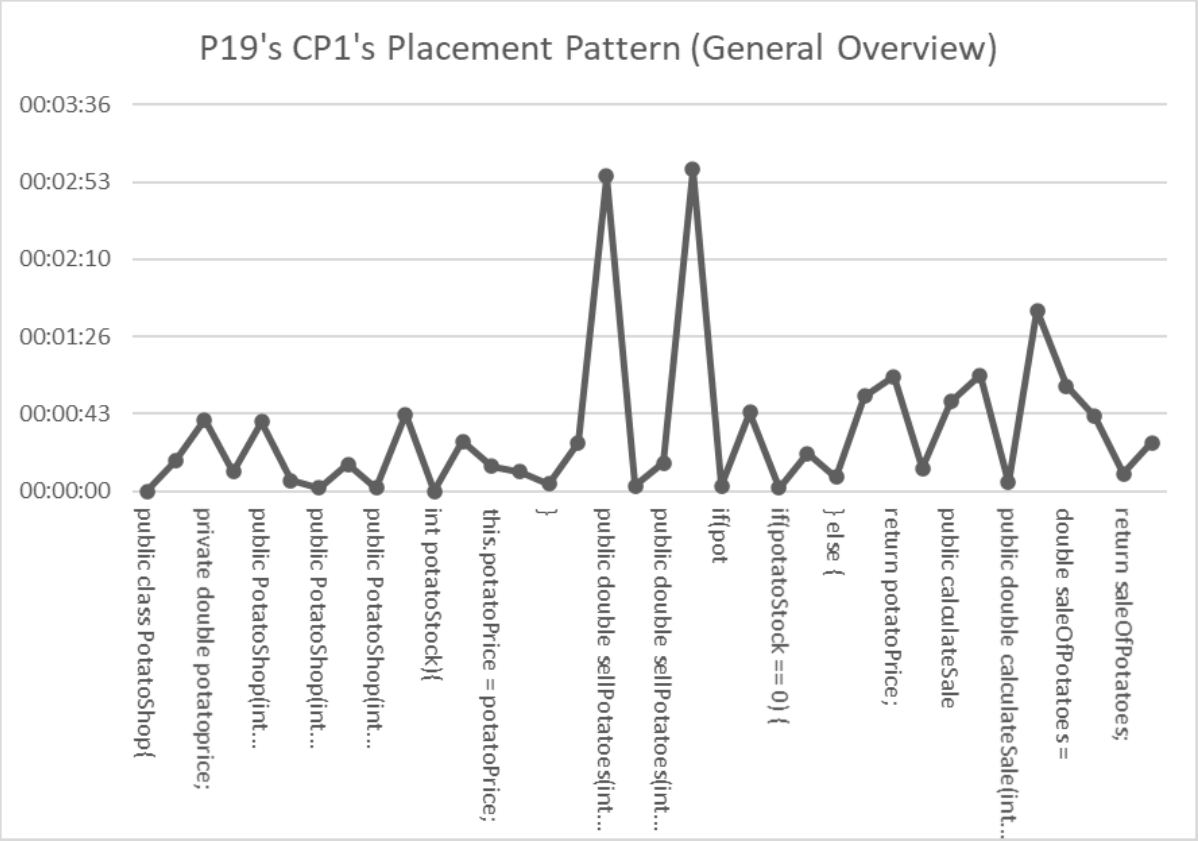


Figure 154: A line graph presenting a general overview of P1's time placement pattern – i.e., the time for each individual movement – for CP1.

P20 suffered a power outage during their puzzle construction, which caused a spike at the beginning of their placement pattern and was the cause of the 8:24 time for the field declarations. The constructor also took a long time to create as the participant needed to reacquaint themselves with the environment. The participant chose to use numerical identifiers instead of typing the piece in full (see Figure 155 and Figure 156).

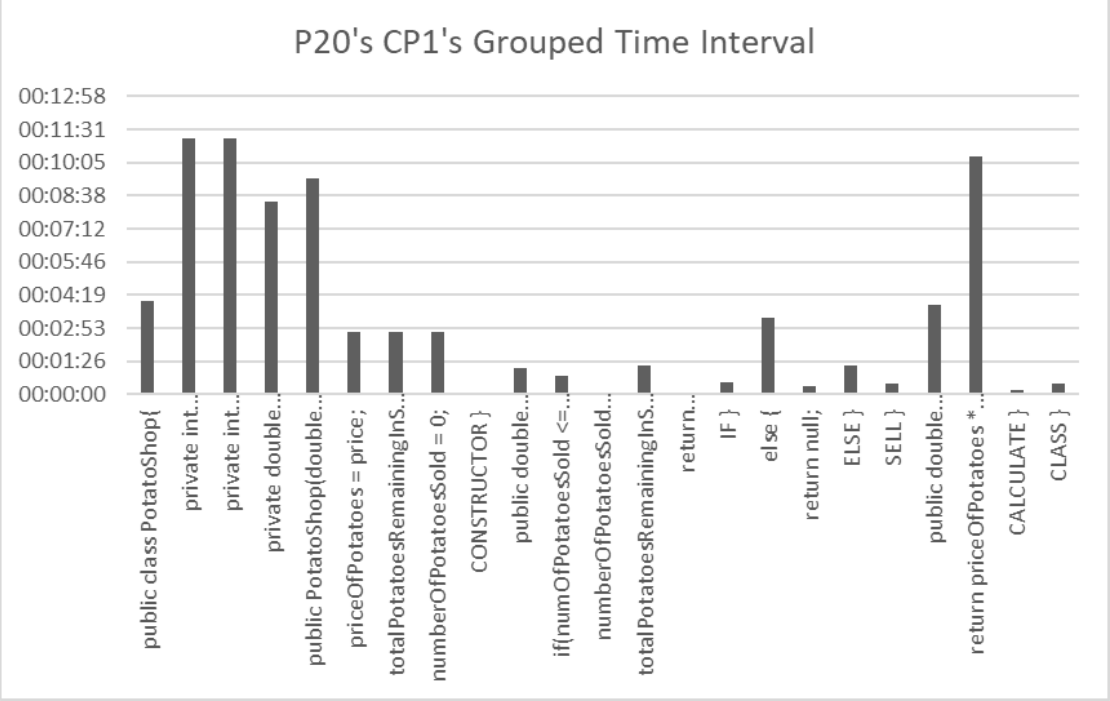


Figure 155: A bar chart of P20's Grouped Time Intervals

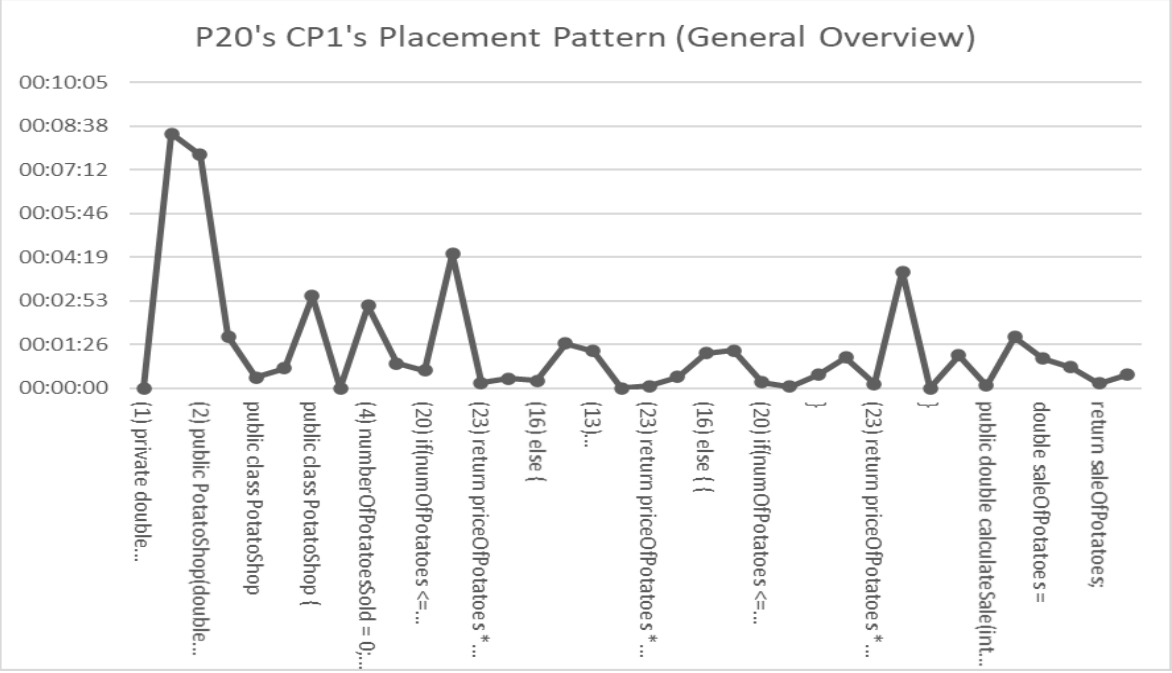


Figure 156: A line graph presenting a general overview of P20's time placement pattern for CP1

P21 seemed to struggle with closing brackets – despite them only being a one-character piece – as the closing if and constructor were forgotten but added later during their pattern placement (see Figure 157 and Figure 158). P21 showed commentary that indicated they were looking through each suggested piece – including the red herrings – and chose the correct version of line 6 but this took

them 2:17. For the sellPotatoes method, P21 struggled to identify an appropriate return statement and seemed to suggest that the number of Potatoes would indicate the consequent price of the sale – they did not link the calculateSale method to sellPotatoes.

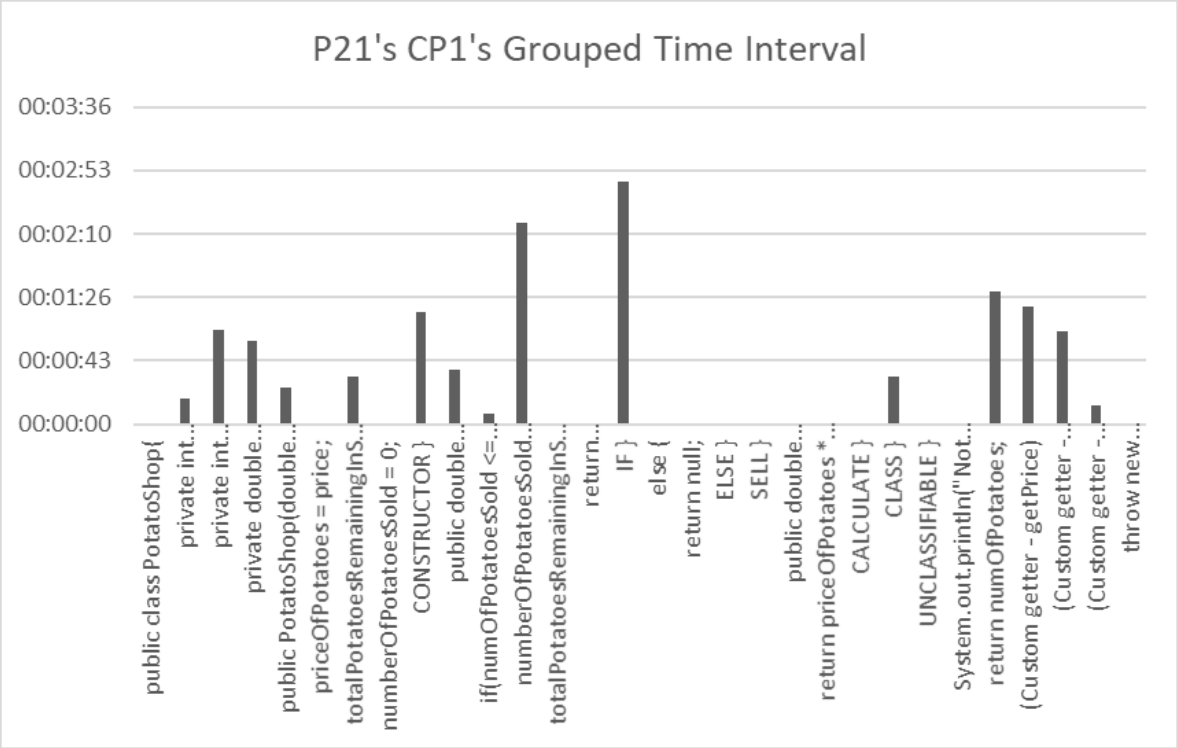


Figure 157: A bar chart of P21's Grouped Time Intervals

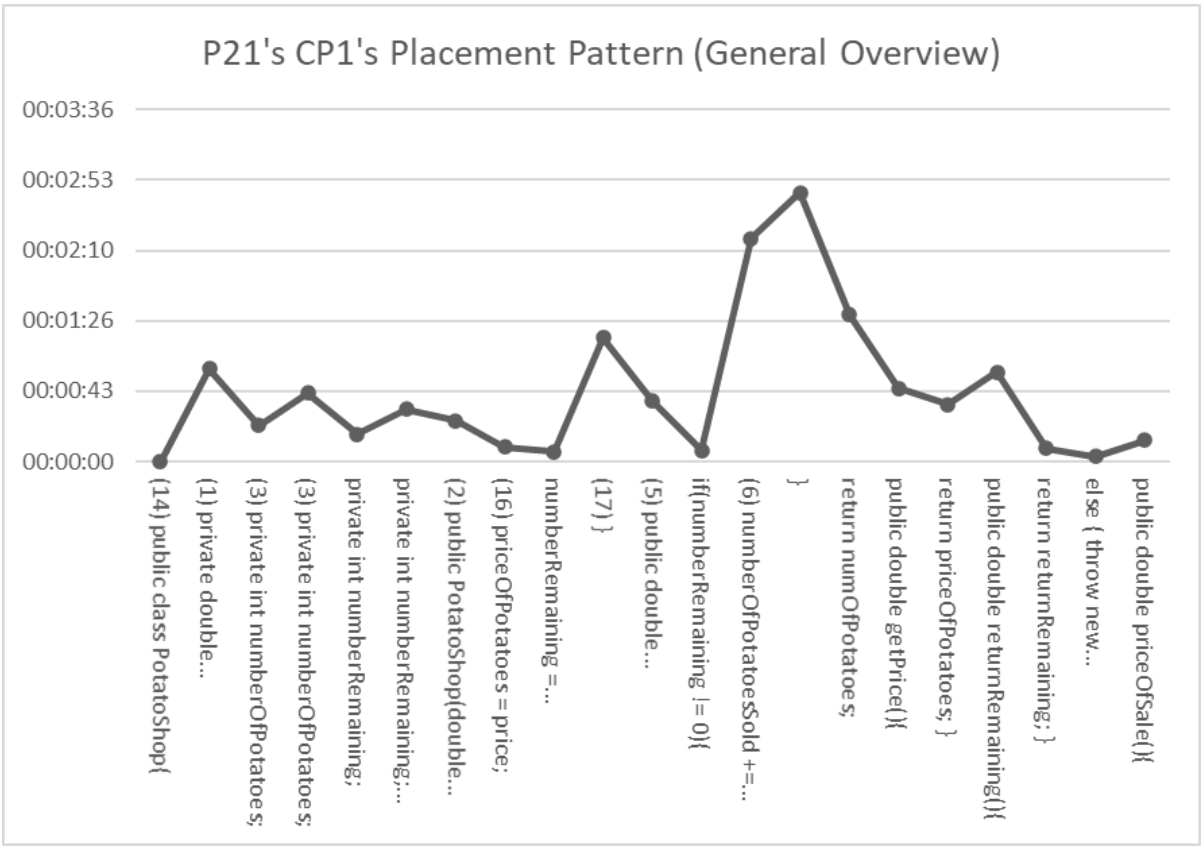


Figure 158: A line graph presenting a general overview of P21's time placement pattern for CP1

The average time spent on each piece is starkly different from previous studies – the methods and if statements for CP1 had some of the lowest time allocations in comparison to the return statements. None of the participants produced perfect solutions; P19, P20 and P22 chose to create custom solutions for CP1 and CP2 and this was likely because the pieces displayed to the users onscreen were not movable meaning that the participants needed to type out the pieces themselves. P20 attempted to map the pieces in CP1 to the ID number on the side of the pieces displayed onscreen but soon became confused by the numbering system that they chose to abandon this approach in CP2. Due to the technical difficulties experienced, the average time spent on CP1 was greater than that of CP2 which was the opposite finding to the pilot and secondary studies – this implies that the way the information is displayed is a key issue to consider as participants more easily understood the CP2 layout over the CP1 layout due to the amount of whitespace available. All participants chose to complete the puzzle in chronological order – from top to bottom – and the times at the beginning of the sample solution reflect that the largest portion of time was spent at the start of the class (see Figure 159).

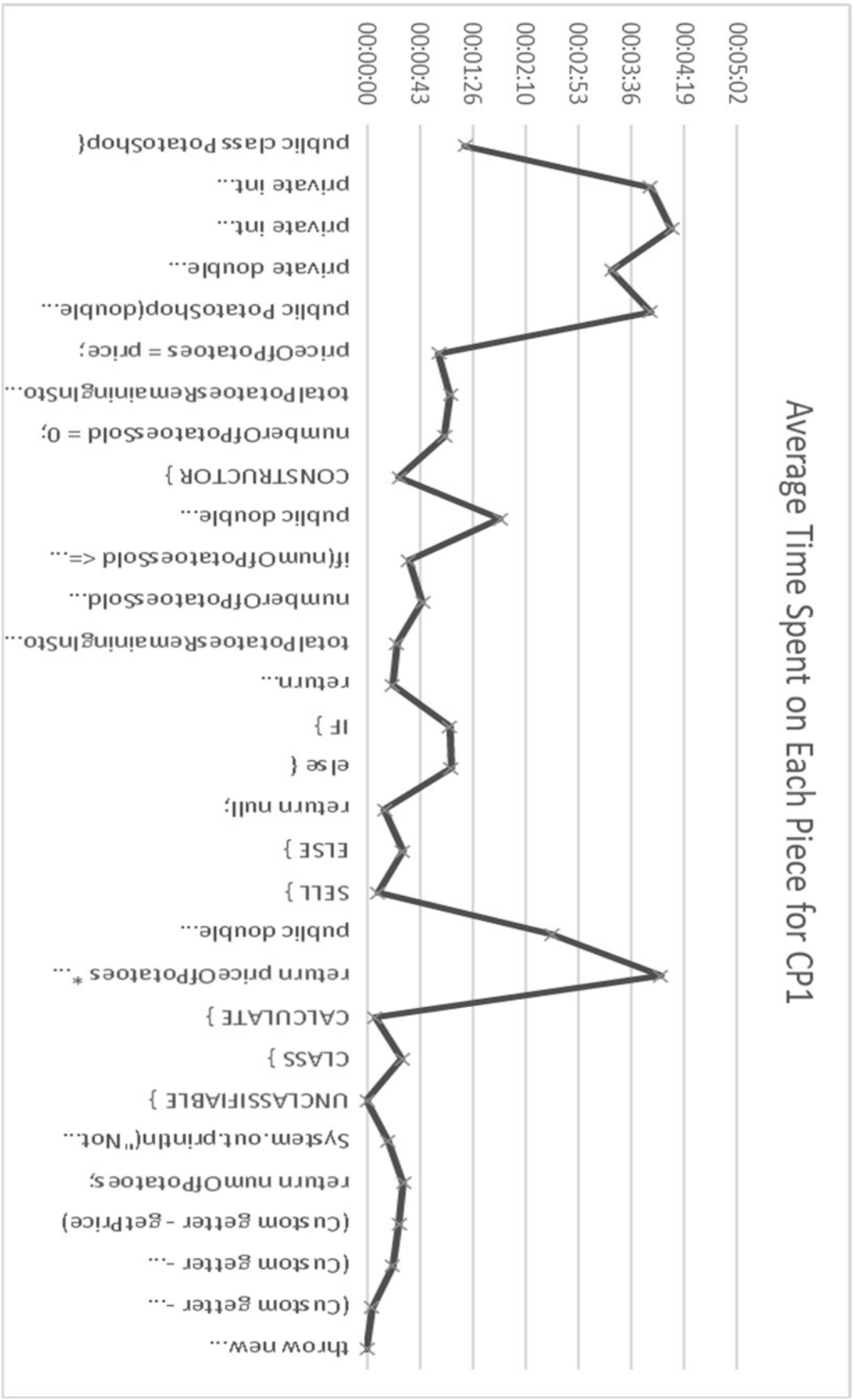


Figure 159: A line chart which demonstrates the average time each participant spent on each part of CP1.

Ultimately, the participants seemed to struggle with the pieces and often chose to create their own pieces to achieve similar goals – the pieces at the end of the chart after the unclassifiable point are customised pieces. P21 wished to create getter and setter methods and later divulged that such methods were important to them and to the accessibility of the class – this does indicate that the participant was thinking of the overall practicality of the class even if the asked for methods were not completed satisfactorily.

7.2.1.2 Code Puzzle 2 Time Intervals

P19 struggled to create the isFresh method and spent a long time thinking about how the return of a boolean could work – to the extent of creating a custom boolean field named 'expired' and using that as a return. This suggests there was an issue with returning values, as the participant only created a class that would assign a true or false factor to the expired boolean which was never returned (see Figure 160 and Figure 161).

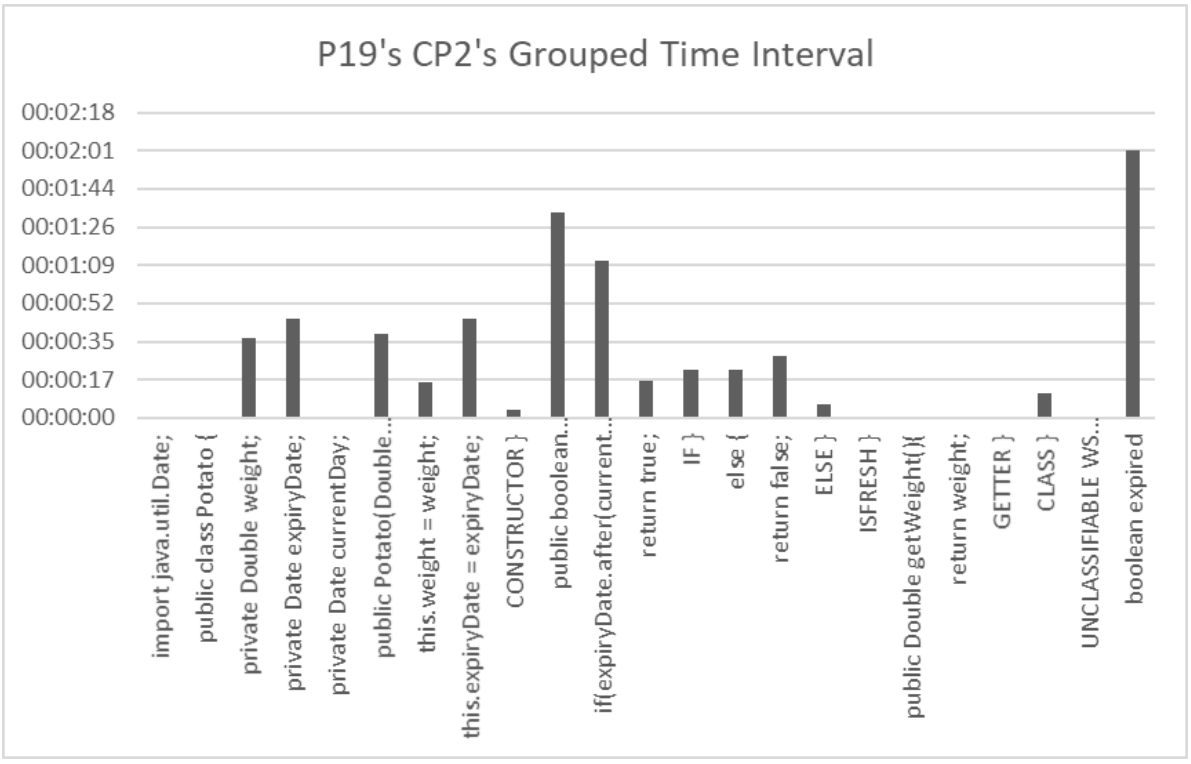


Figure 160: A bar chart of P19's Grouped Time Intervals

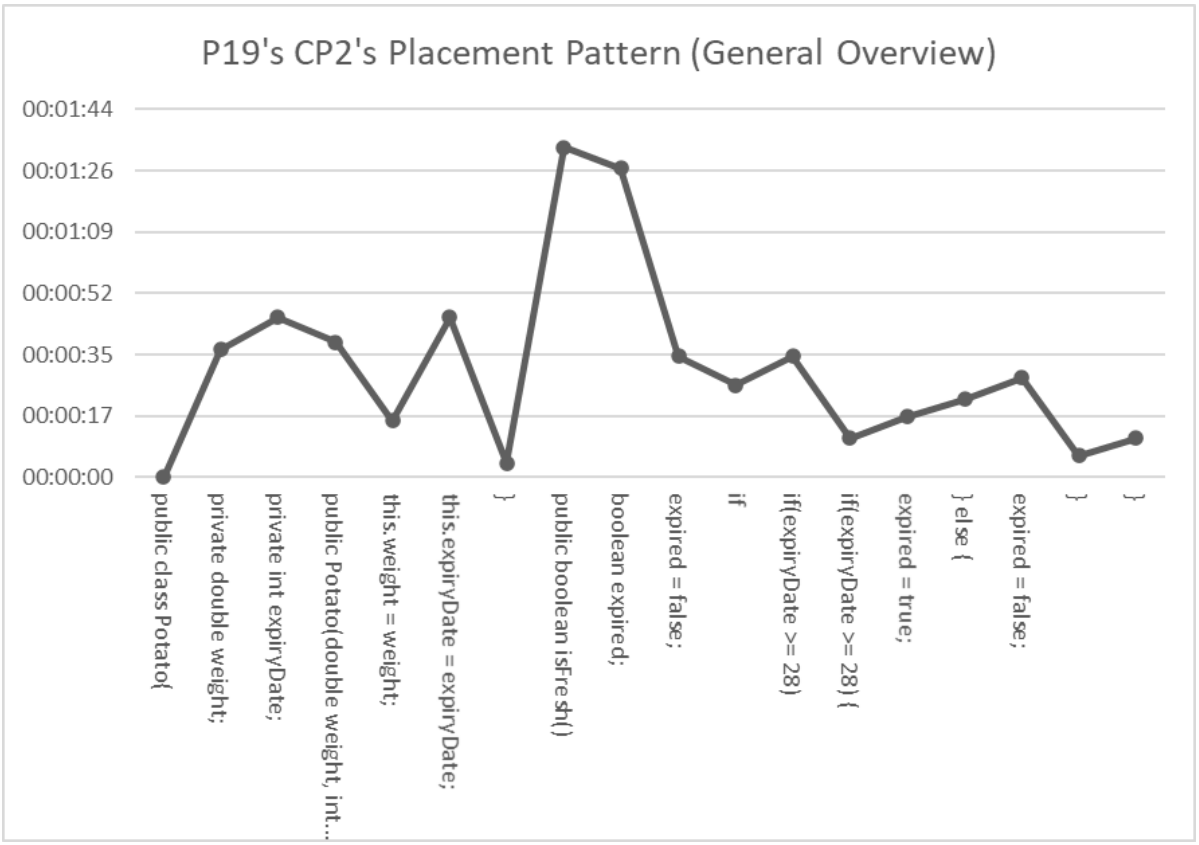


Figure 161: A line graph presenting a general overview of P19's time placement pattern for CP2 and a table documenting the top five pieces that had the longest time intervals.

P20 struggled to create the if condition for the isFresh method, and created a condition that would work to some degree but is not as good quality as the previous studies' answers (see Figure 162 and Figure 163).

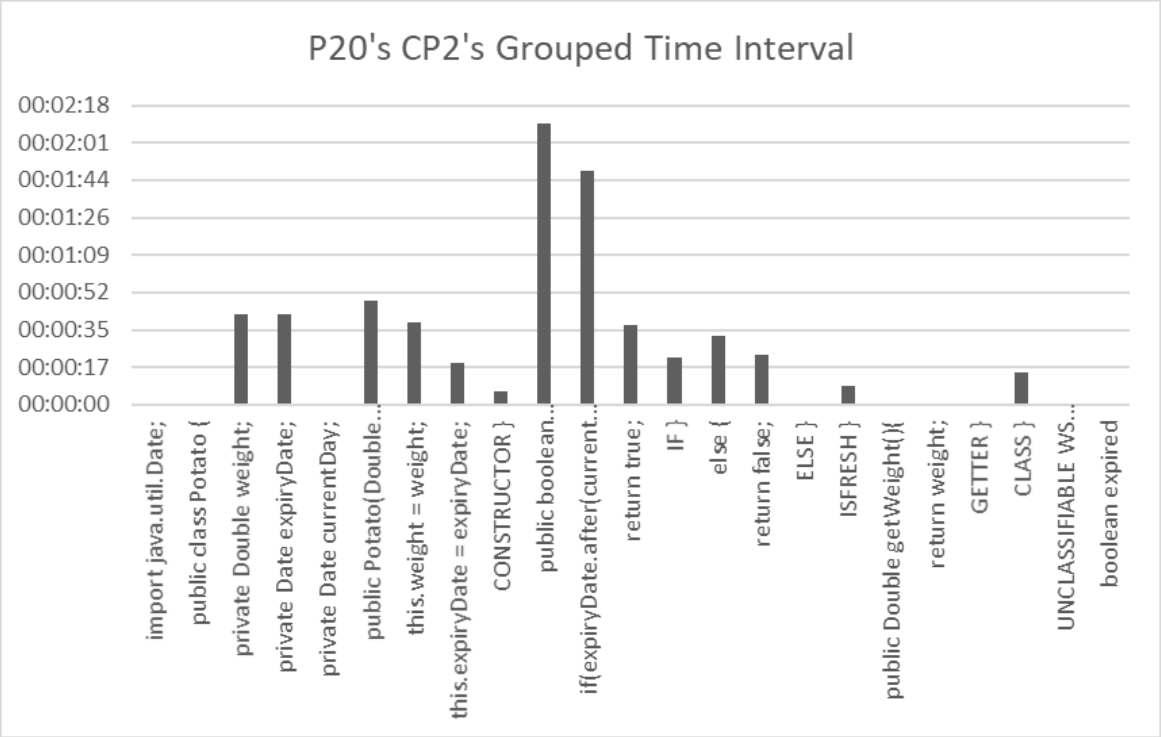


Figure 162: A bar chart of P20's Grouped Time Intervals

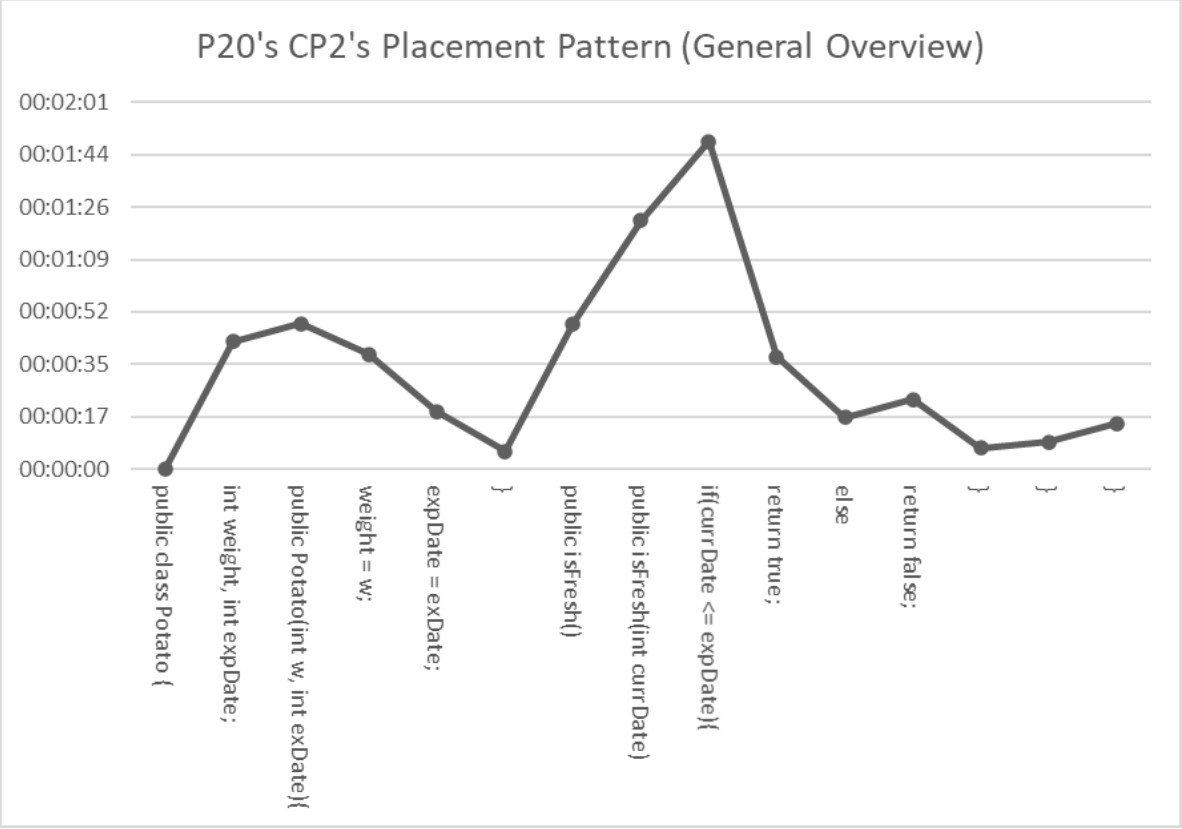


Figure 163: A line graph presenting a general overview of P20's time placement pattern for CP2

P21 struggled to work out the format of how a date object would work as an integer, and assumed the object would need to have a string-like format (e.g., dd/mm/yyyy) to be processed by the

compiler. They needed to spent time thinking about how to split the String into a readable comparable format so that it could be used in their condition. There was a clear issue with their naming of variables, and during the audio transcripts the participant even acknowledges that the names were bad and that they had confused themselves during the feedback session. The names 'one', 'two', 'date', 'currentDate', and 'expiryDate' meant they did not realise that a logical error would occur when they ran their code due to date and currentDate representing the same idea (see Figure 164 and Figure 165).

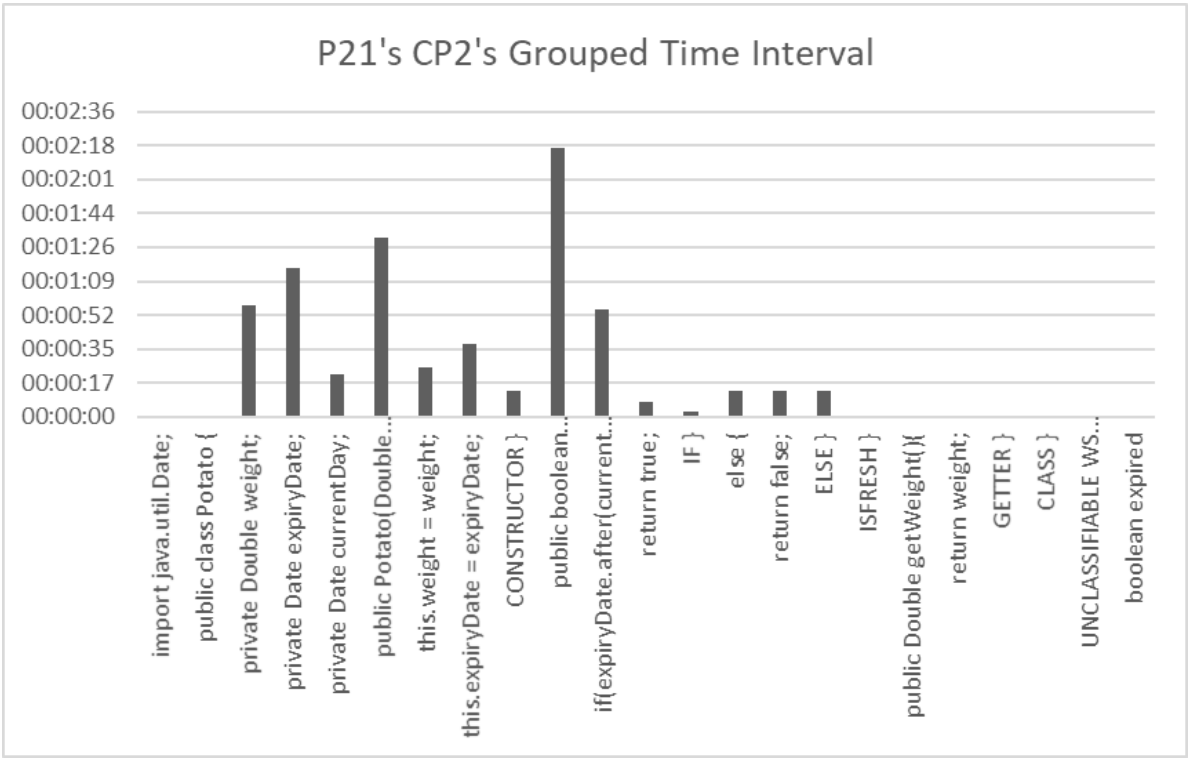


Figure 164: A bar chart of P21's Grouped Time Intervals

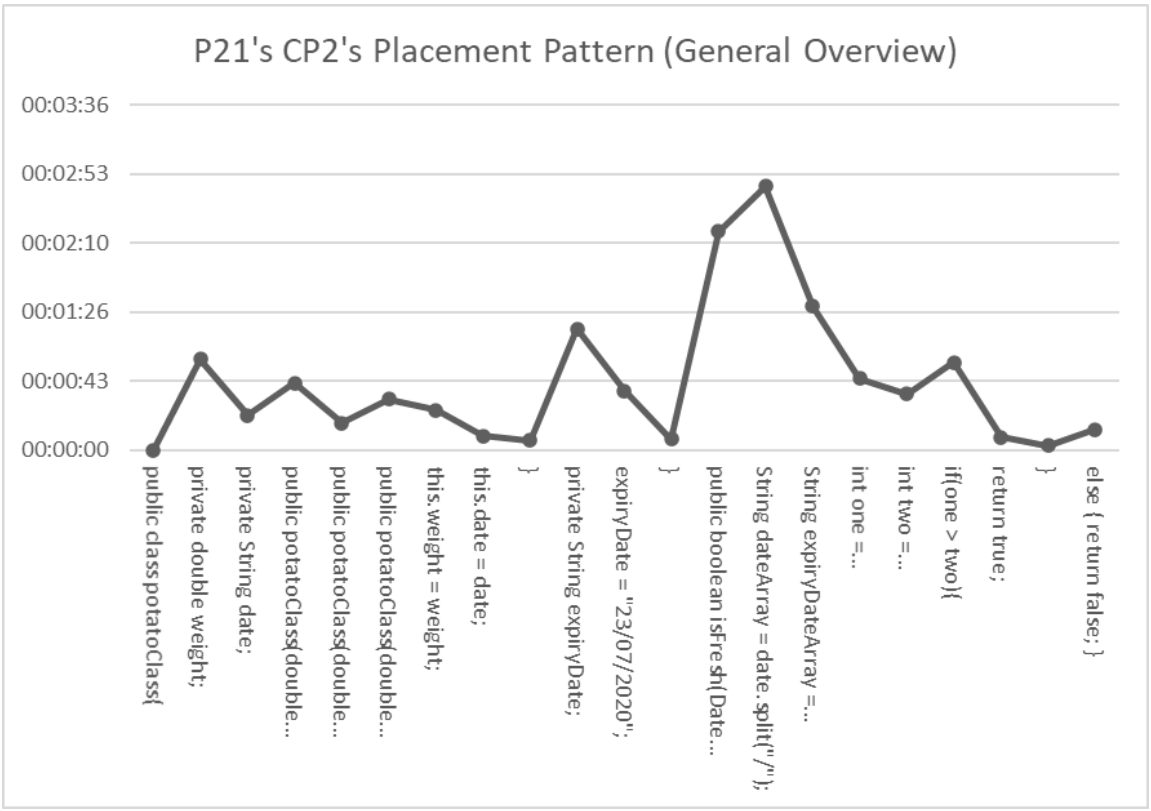


Figure 165: A line graph of P21's time placement pattern for CP2

All participants had a spike in the center of their pattern placement where they were considering the isFresh method, which matches the findings of the pilot study (see Figure 166).

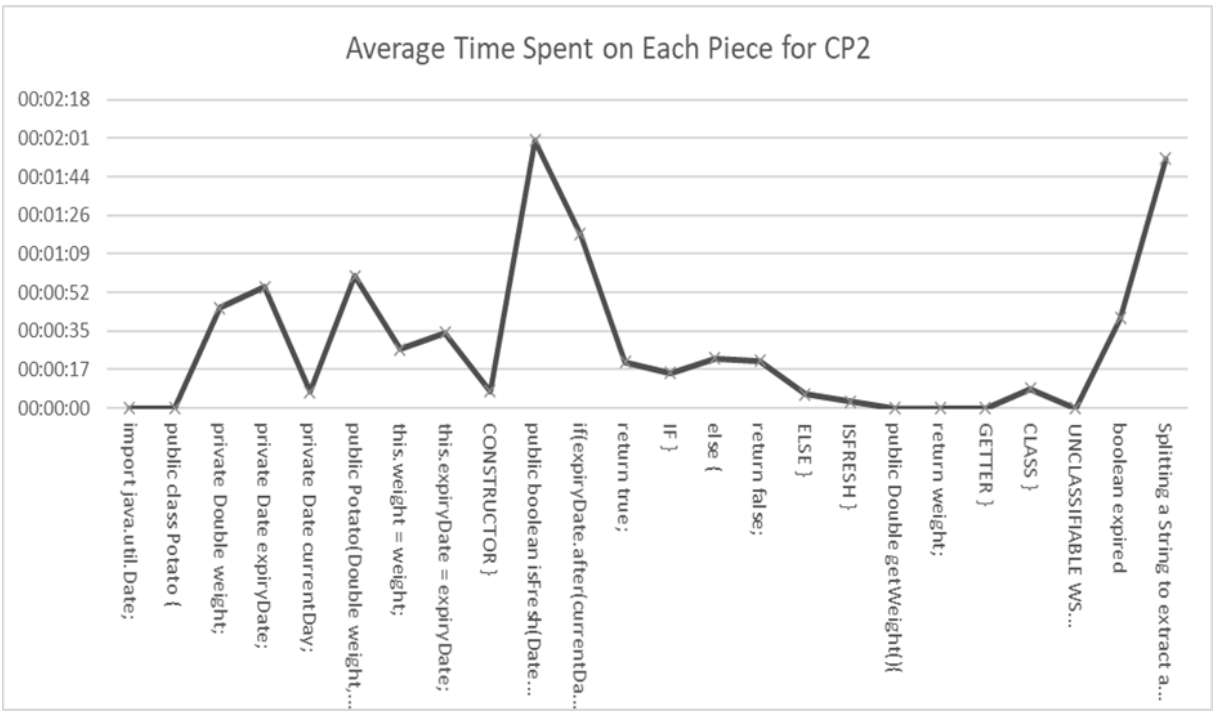


Figure 166: A line chart which demonstrates the average time each participant spent on each part of CP2.

7.2.2 Movement Observations

7.2.2.1 Frequency of Movements

The movement observations for the tertiary study were limited by the allowances of the Blackboard collaborate interface – movements were notably less in CP2 than in CP1 (see Figure 167).

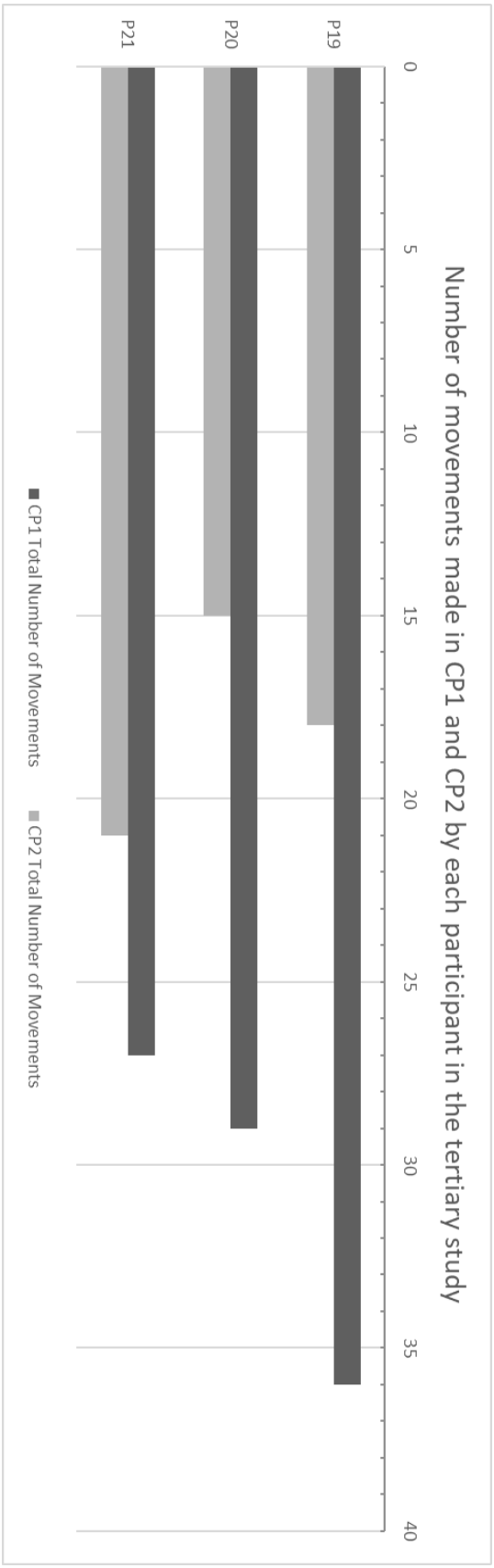


Figure 167: Clustered bar chart for the number of movements made by participants for Code Puzzle 1 (CP1) and Code Puzzle 2 (CP2).

The number of movements per piece reflects the participants getting acquainted with how the Blackboard Collaborate Ultra interface works – therefore the pieces at the beginning of CP1 (the fields and constructor) were moved the most as the length of the constructor meant that that line of code went off screen and needed to be shifted onto the screen. However, there was some confusion which matched the pilot study findings regarding the return statements for sale and sellPotatoes method – there was a participant who used the same piece multiple times due to confusion about the difference between the two methods (see Figure 168, Figure 169, Figure 170 and Figure 171). Indentation was also seen with all participants, who were particular about where they placed the pieces.

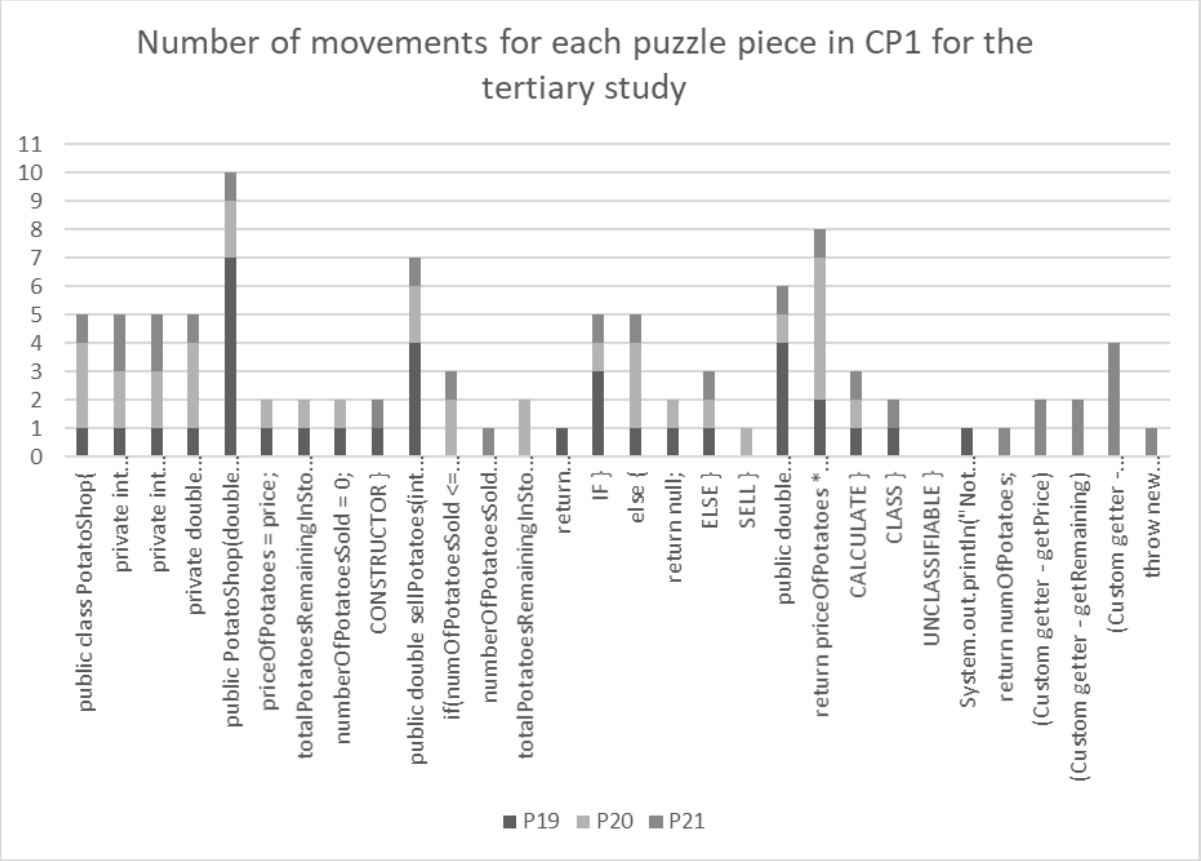


Figure 168: Stacked bar chart illustrating the number of movements made per puzzle piece per participant for CP1.

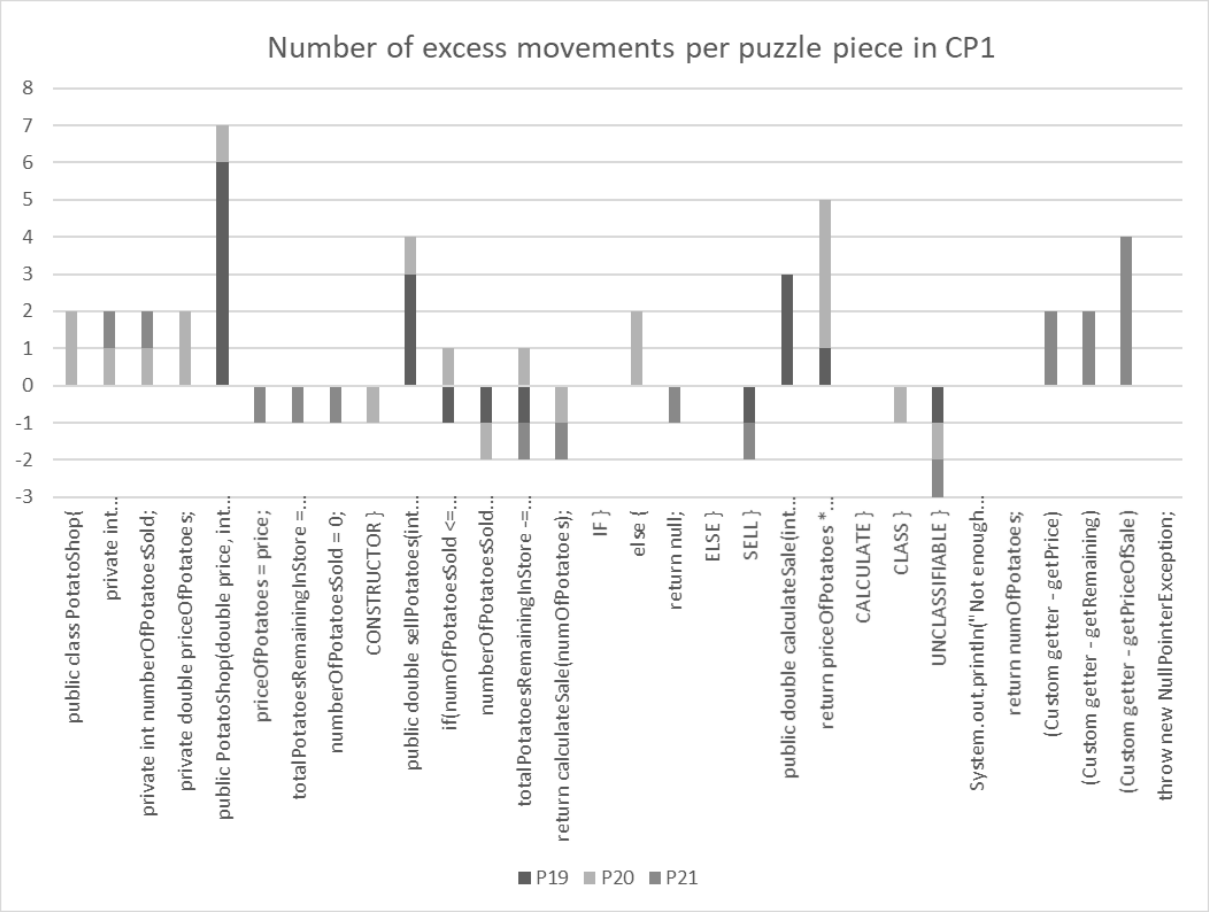


Figure 169: Stacked bar chart that illustrates the number of 'excess' movements –movements that exceed the anticipated number required to construct a line of – recorded for CP1.

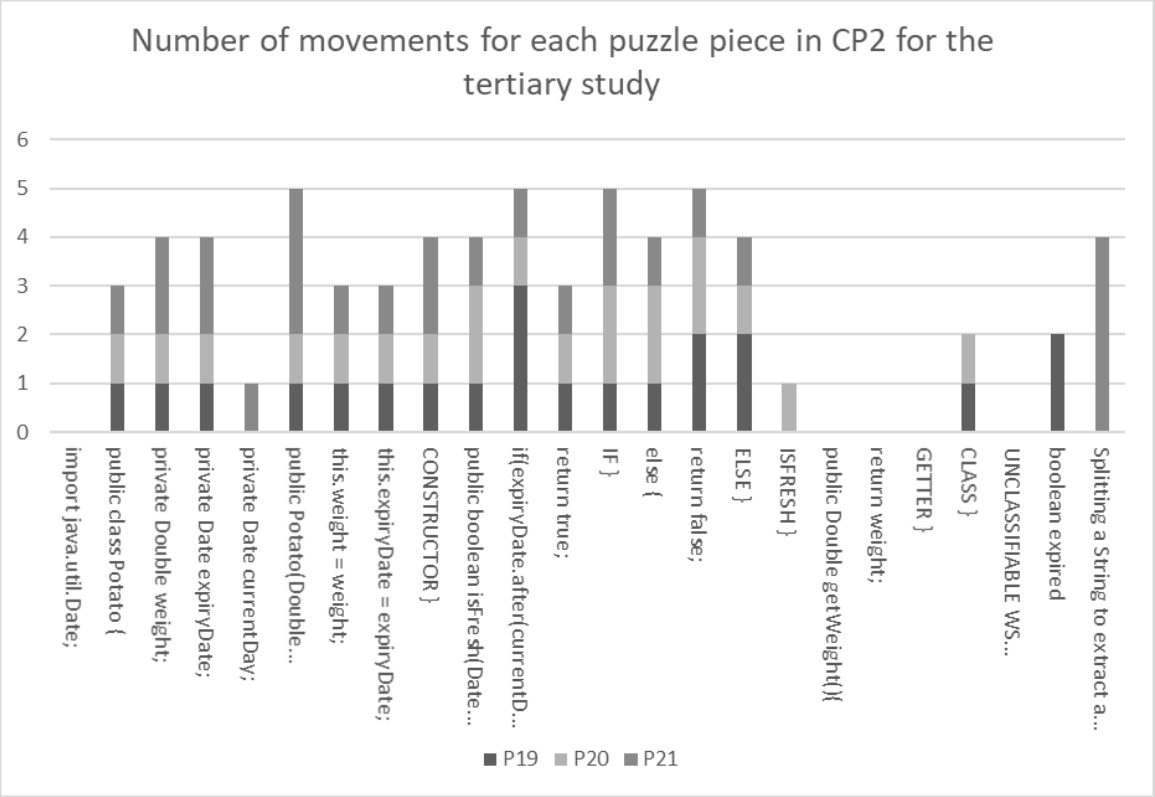


Figure 170: Stacked bar chart illustrating the number of movements made to create each line of code per participant for CP2.

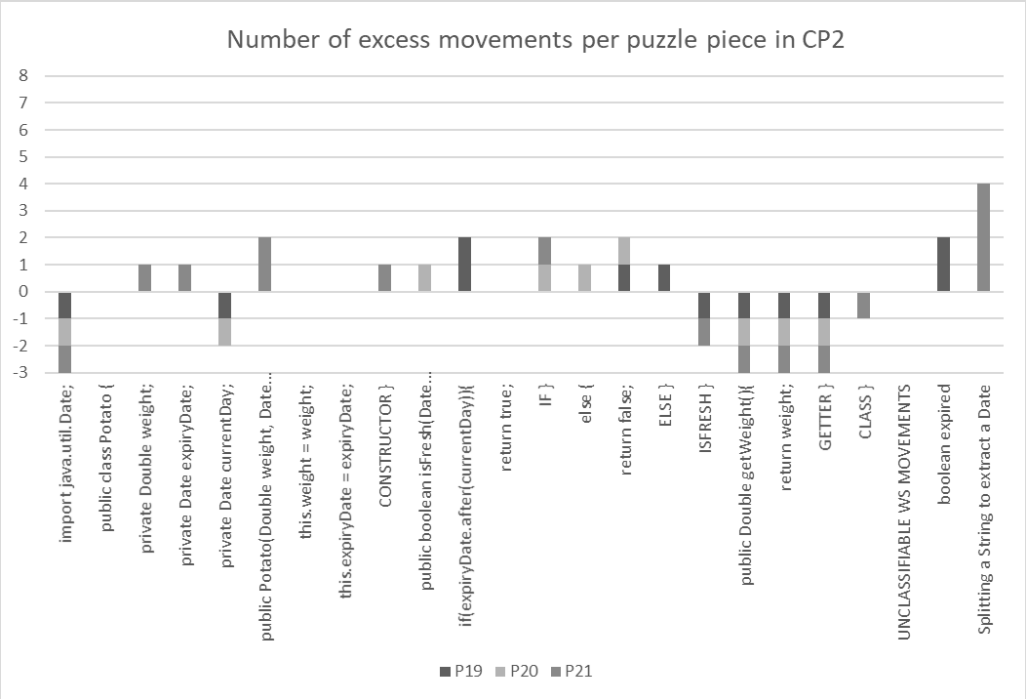


Figure 171: Stacked bar chart that illustrates the number of 'excess' movements –movements that exceed the anticipated number required to construct a line of – recorded for CP1. P1's 63 unclassifiable movements are omitted.

7.2.2.2 Types of Movements Observed

The types of movements observed were limited – not only were there few movements, but these movements were virtually 'add' and 'shift' types only, with one swap type (see Figure 172). This suggests that the workspace is a hidden area and is influenced by the interface design – this would likely be different if pieces were movable.

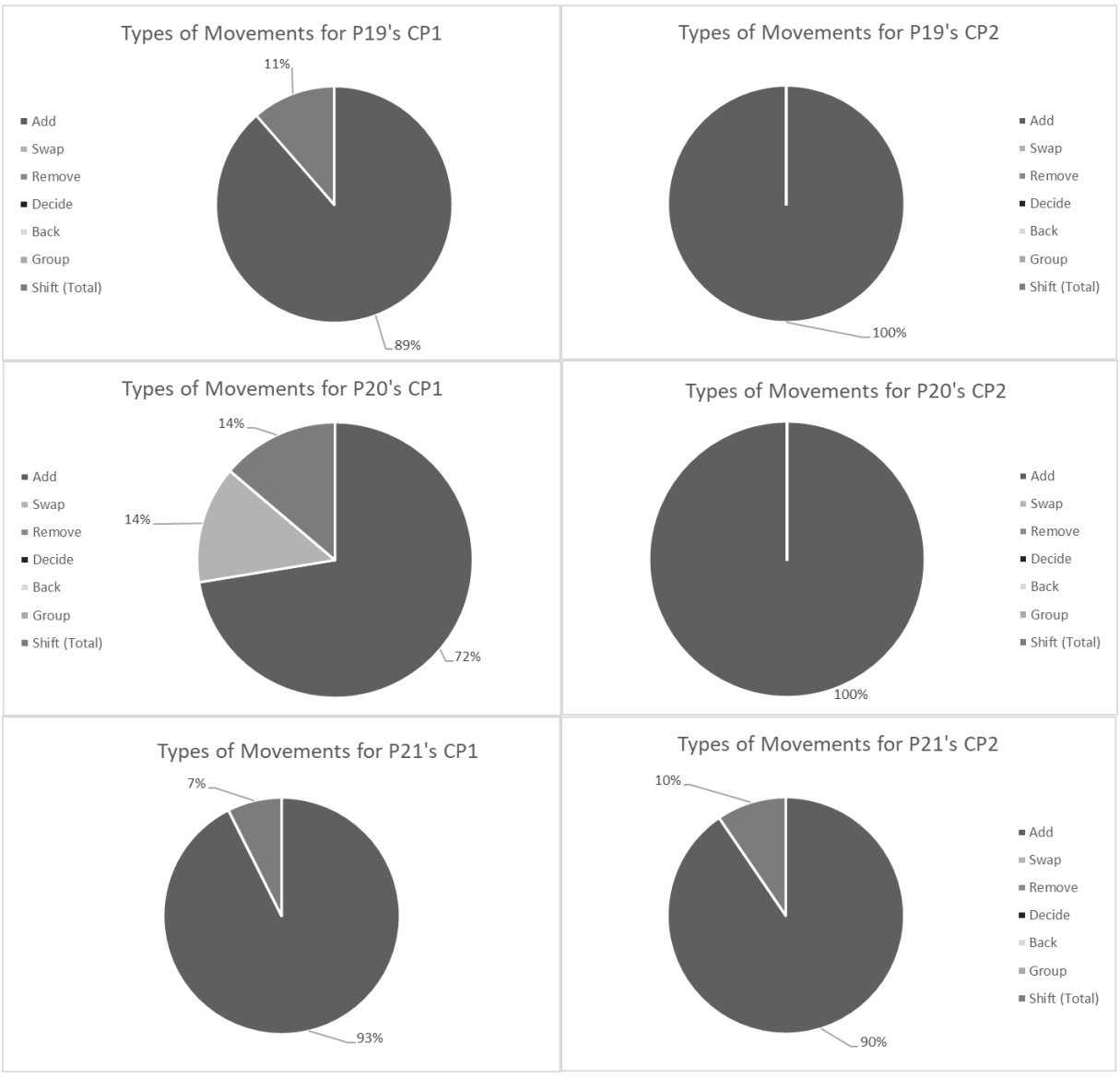


Figure 172: The classification of movement type for each participants' movements in CP1 and CP2.

7.2.3 Analysis of the Submitted Solutions

In comparison to pilot and secondary studies, participants submitted more solutions that had customised pieces inside of them. This is not surprising, as participants needed to type the pieces

from scratch into the text box and naturally shows the names, they would choose for aspects of the class that differ from the standard pieces. All the solutions submitted were of poorer quality than the previous two studies' solutions – this is likely because the extra freedom of choosing how to type each piece meant that participants needed to determine the underpinning logic of the class and were not guided by the number of pieces remaining as such a factor did not exist on the Blackboard Collaborate Ultra interface. The solutions produced can be viewed in: Figure 173, Figure 174 and Figure 175.

Figure 173: Representations of P19's submitted solutions for CP1 (left) and CP2 (right)

```
public class PotatoShop {
    private double priceOfPotatoes;
    private int numberOfPotatoesSold;
    private int totalPotatoesRemainingInStore;

    public PotatoShop(double price, int totalPotatoesInStore) {
        numberOfPotatoesSold = 0;
        priceOfPotatoes = price;
        totalPotatoesRemainingInStore = totalPotatoesInStore;
    }

    public double sellPotatoes(int numOfPotatoes) {
        if(numOfPotatoes <= totalPotatoesRemainingInStore){
            totalPotatoesRemainingInStore -= numOfPotatoes;
            return calculateSale(numOfPotatoes);
        }
        else {
            return null;
        }
    }

    public double calculateSale(int numOfPotatoesSold){
        return priceOfPotatoes*numOfPotatoesSold;
    }
}
```

```
public class Potato {
    int weight, expDate;

    public Potato(int w, int exDate) {
        weight = w;
        expDate = exDate;
    }

    public isFresh(int currDate) {
        if(currDate <= expDate) {
            return true;
        }
        else {
            return false;
        }
    }
}
```

Key:

= Ill-advised

= Incorrect

= Deviation from anticipated Solution

= Identical to real solution

Figure 174: Representations of P20's submitted solutions for CP1 (left) and CP2 (right)

Figure 175: Representations of P21's submitted solutions for CP1 (left) and CP2 (right)

7.2.4 Questionnaires

Participants were asked to complete: one background, one CP1 pre-puzzle, one CP1 post-puzzle, one CP2 pre-puzzle, one CP2 post-puzzle and one-post study questionnaire.

7.2.4.1 Background Questionnaire

On average, participants were 'fairly confident' in their capabilities as a programmer (see Figure 176) in comparison to the 'slightly confident' participants in the secondary study.

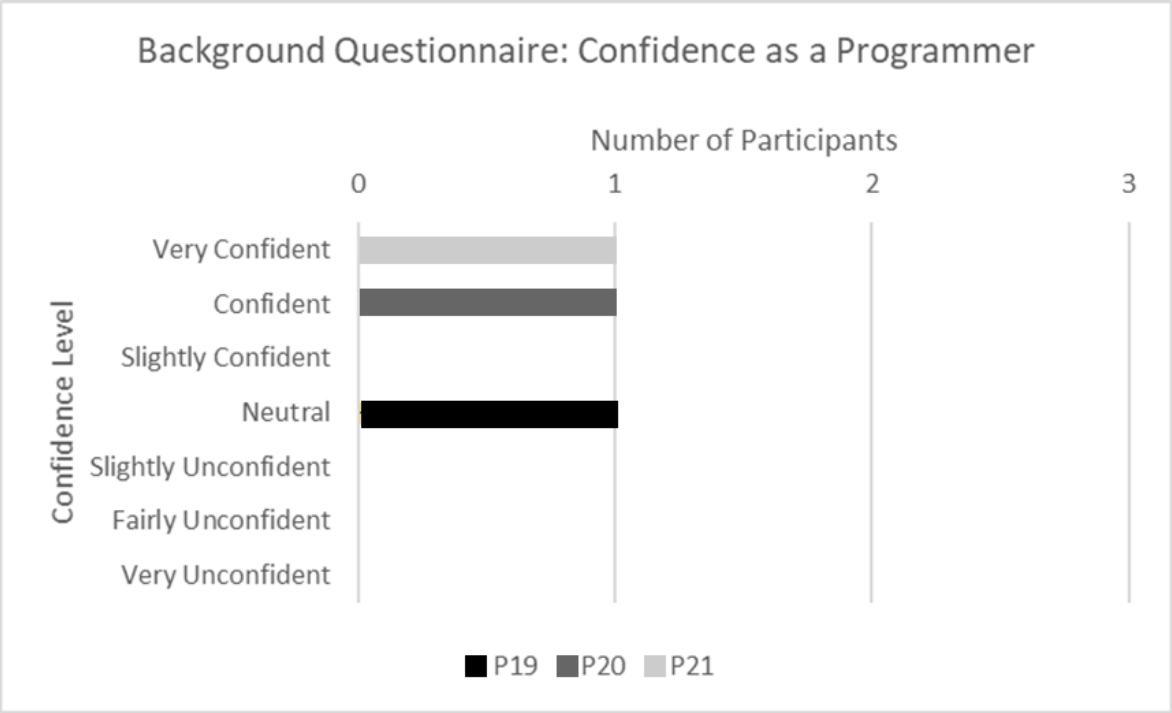


Figure 176: Participants answers to the background questionnaire question: 'How confident are you as a programmer?' in the tertiary study (M = "Fairly Confident")

Participants knew multiple languages, with P20 and P21 classifying themselves as fluent in one. If Figure 176 and Figure 177 are compared, there is the possibility of a link to the level of confident and the number of languages an NP is considered fluent or proficient in.

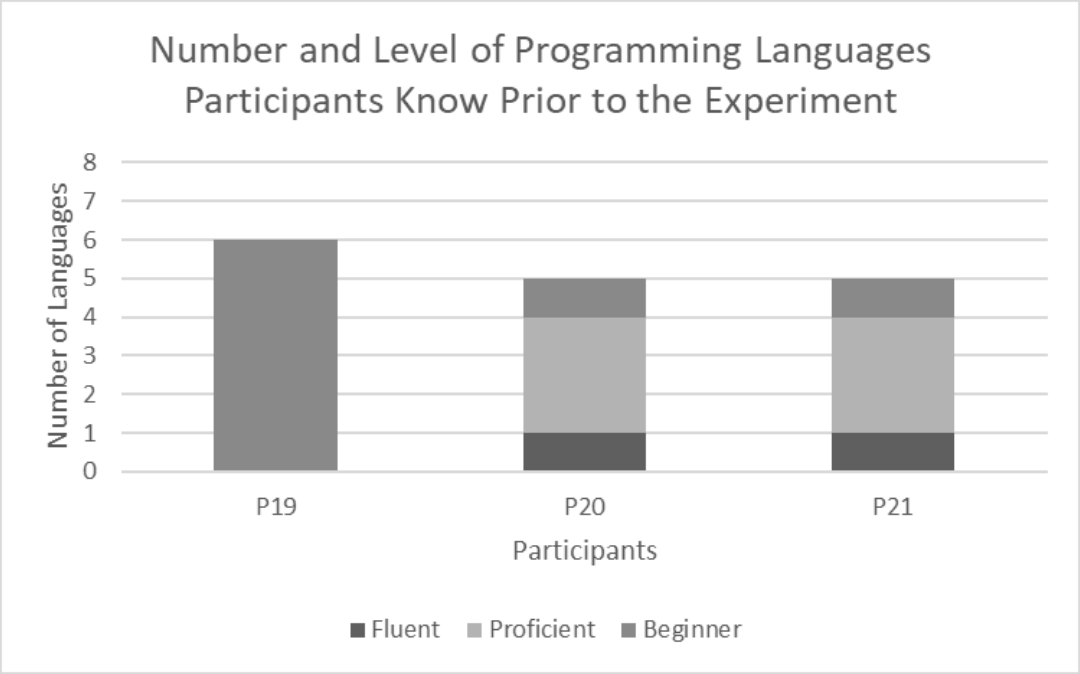
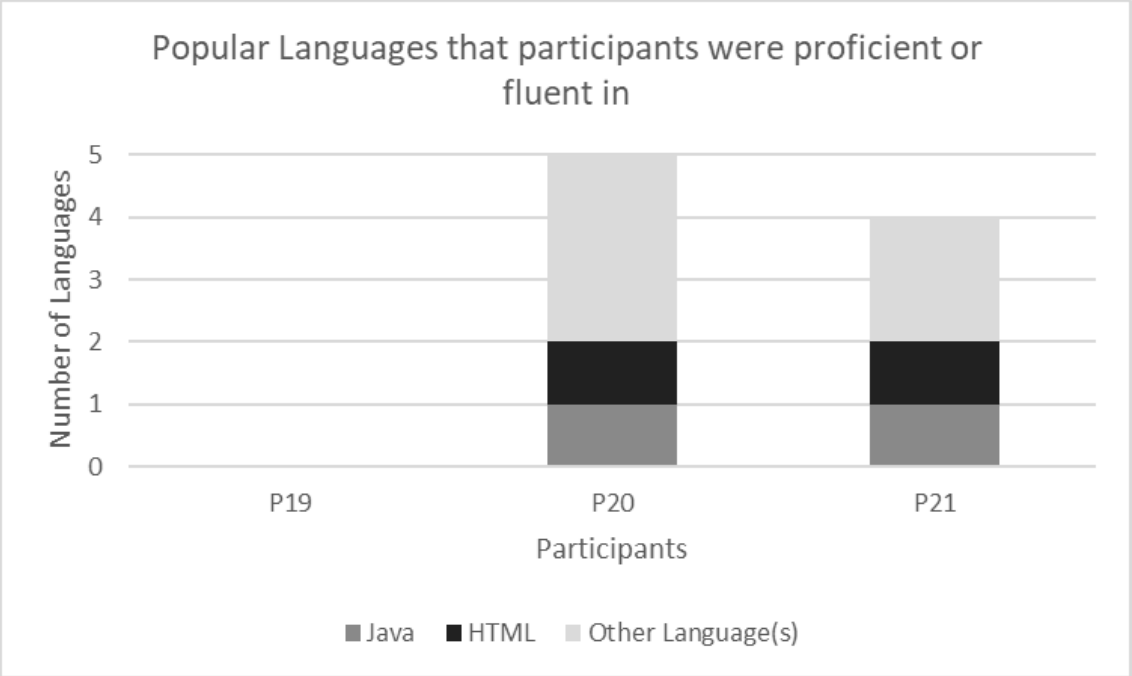


Figure 177: Participants answers to the background questionnaire question: 'How many programming languages are you: fluent, proficient and beginner in?' for the tertiary study. 'Other Languages' are for languages selected by only that participant and could compromise their identity.

Despite the number of languages, most participants did not have any language in common that they noted down for the next question aside from HTML and Java (see Figure 177).



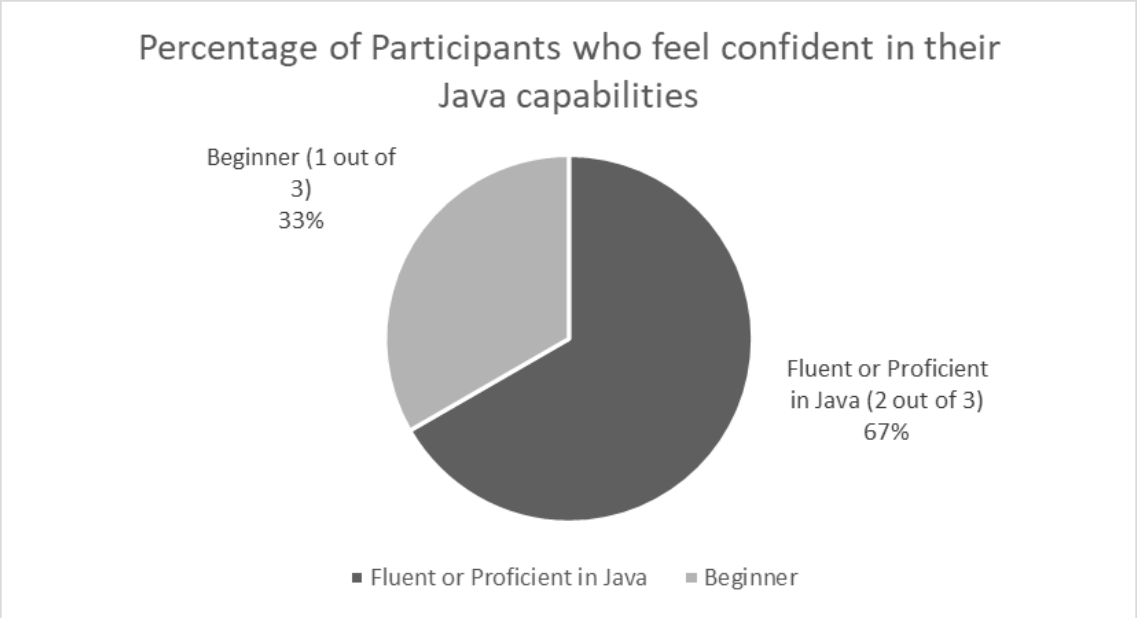
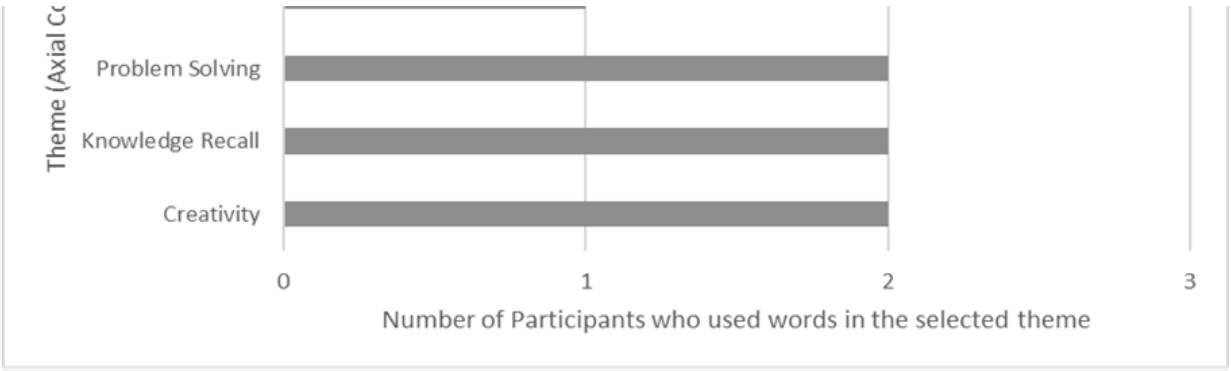
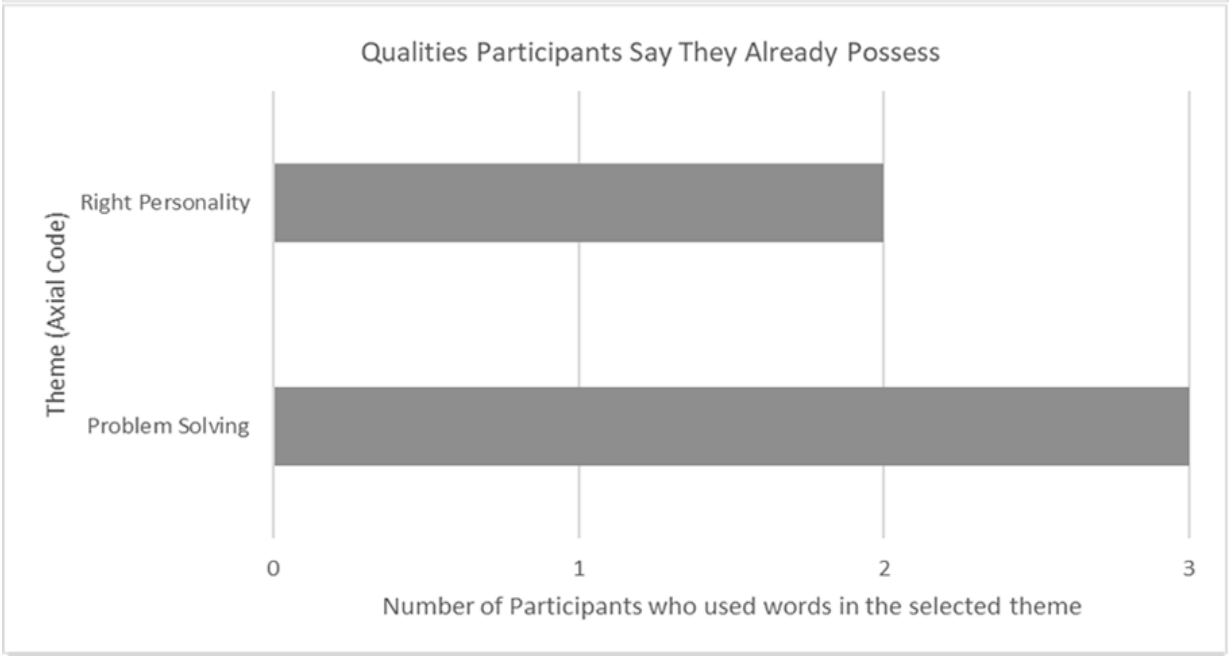


Figure 178: What programming languages are you fluent or proficient in? 'Other Languages' are for languages selected by only that participant and could compromise their identity. (Java: total = 2 participants || HTML: total = 2 participants)

Tertiary participants felt that problem solving was a core requirement for a programmer, which coincides with the findings of the secondary study. Knowledge recall (such as knowing the language), the right personality traits (such as patience and perseverance) were also documented in answers. All participants felt that they needed to improve their problem-solving skills, and this matches the poorly developed approaches highlighted in Figure 181.





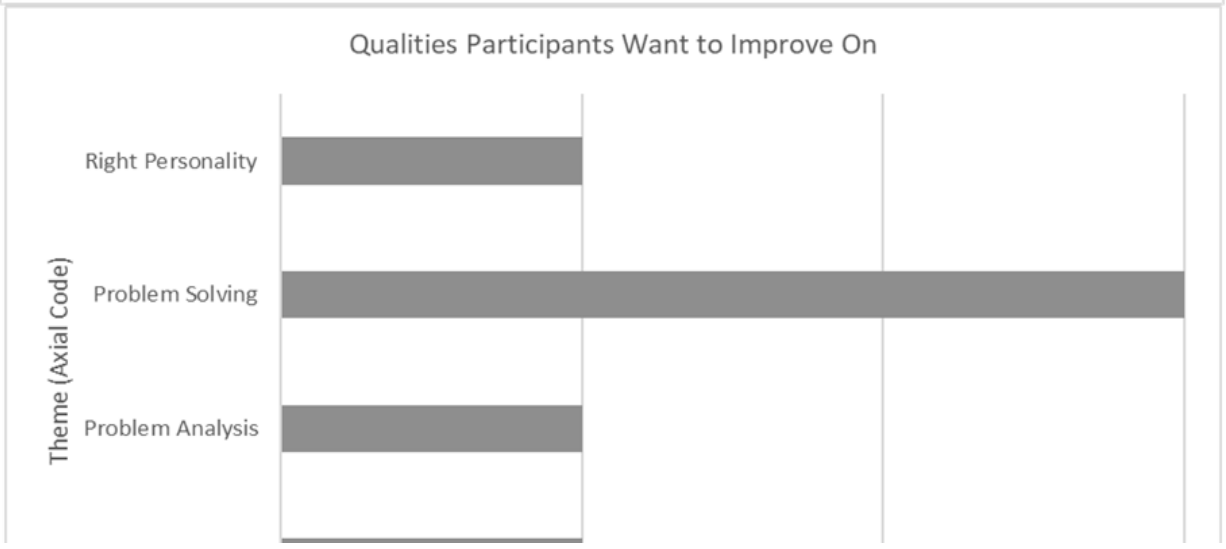


Figure 179: Three stacked bar charts that illustrate the tertiary participants' answers to three related questions in the background questionnaire: 'What qualities does a programmer require (in your opinion)?' (top); 'Which qualities of a programmer do you feel you have?' (middle); and 'Which qualities of a programmer do you feel you need to improve on?'. These were codes created through applying Thematic analysis to the open-ended question answers given.

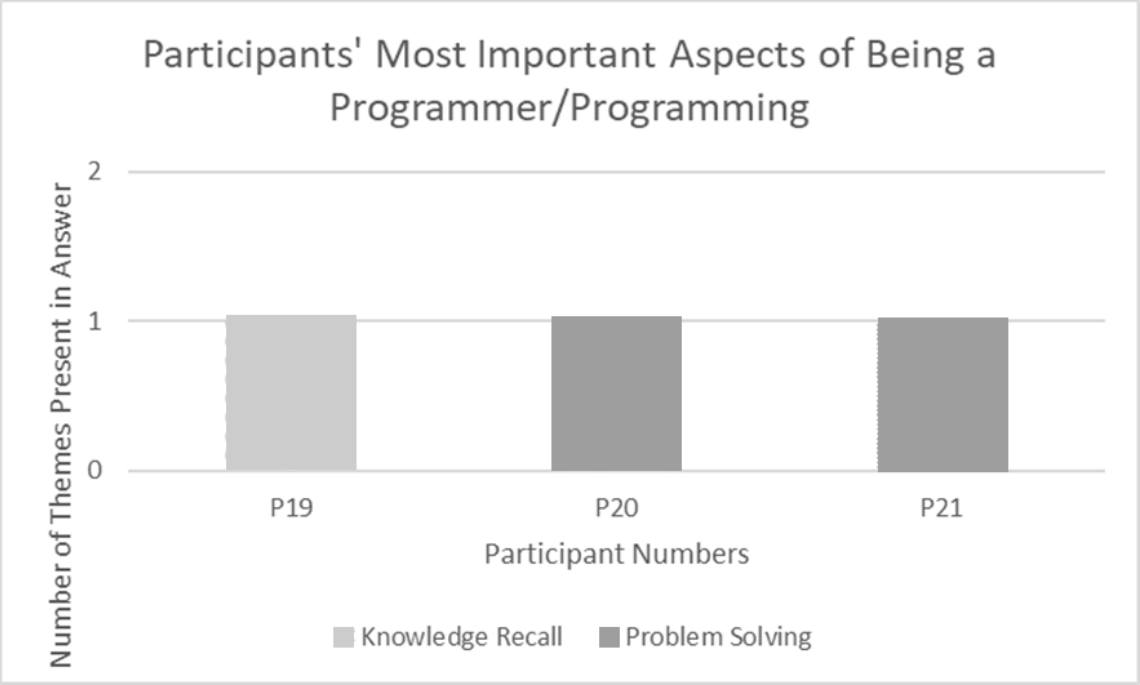


Figure 180: What is the most important aspect of understanding programming (in your experience)?

Each of the participants exhibited a different approach to programming; P20 had the closest approach to the secondary study participants but P19's process was new. This suggests that P19 was correct in their post-study analysis, because their approach could not be identified from looking at the movements of Code Puzzle pieces alone – as their process entirely revolves around copying other people's code and tweaking it. P21 showed little development in terms of their approach to programming – they had a very vague step of 'working out how to do the task' which suggests they do not have a standard logical process for working out how to do that task (see Figure 181).

Participants did not seem to anticipate the tasks themselves as being drastically more difficult than each other and tended to judge the difficulty in the same way with the exception of P19 who was greatly disheartened by the use of a date object and the Java documentation which added to the complexity of the task (see Figure 182).

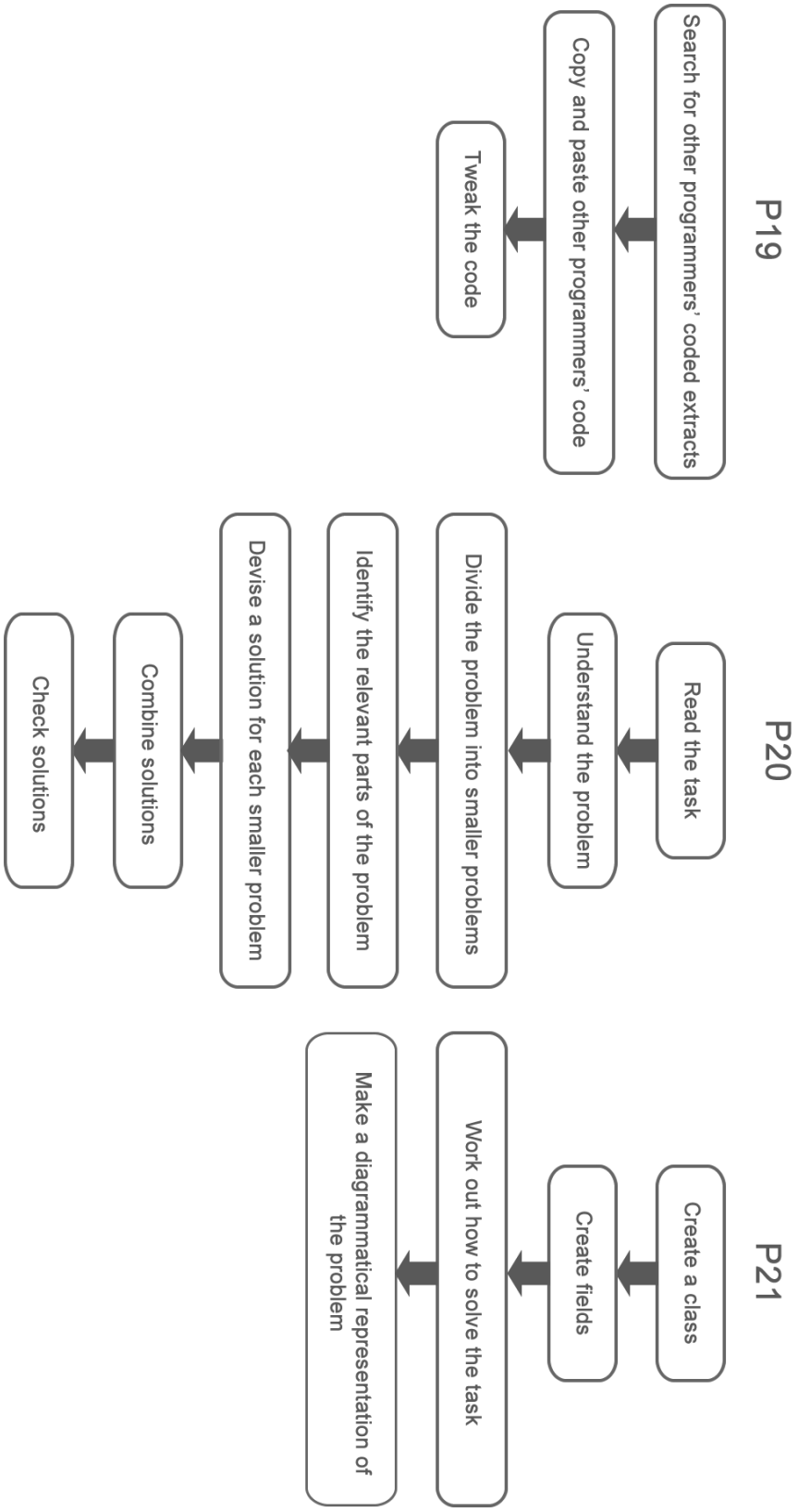


Figure 181: How participants answered 'Can you describe the steps you take to solve a programming task?' for the tertiary study; the process could not be generalised as each participant gave a unique answer.

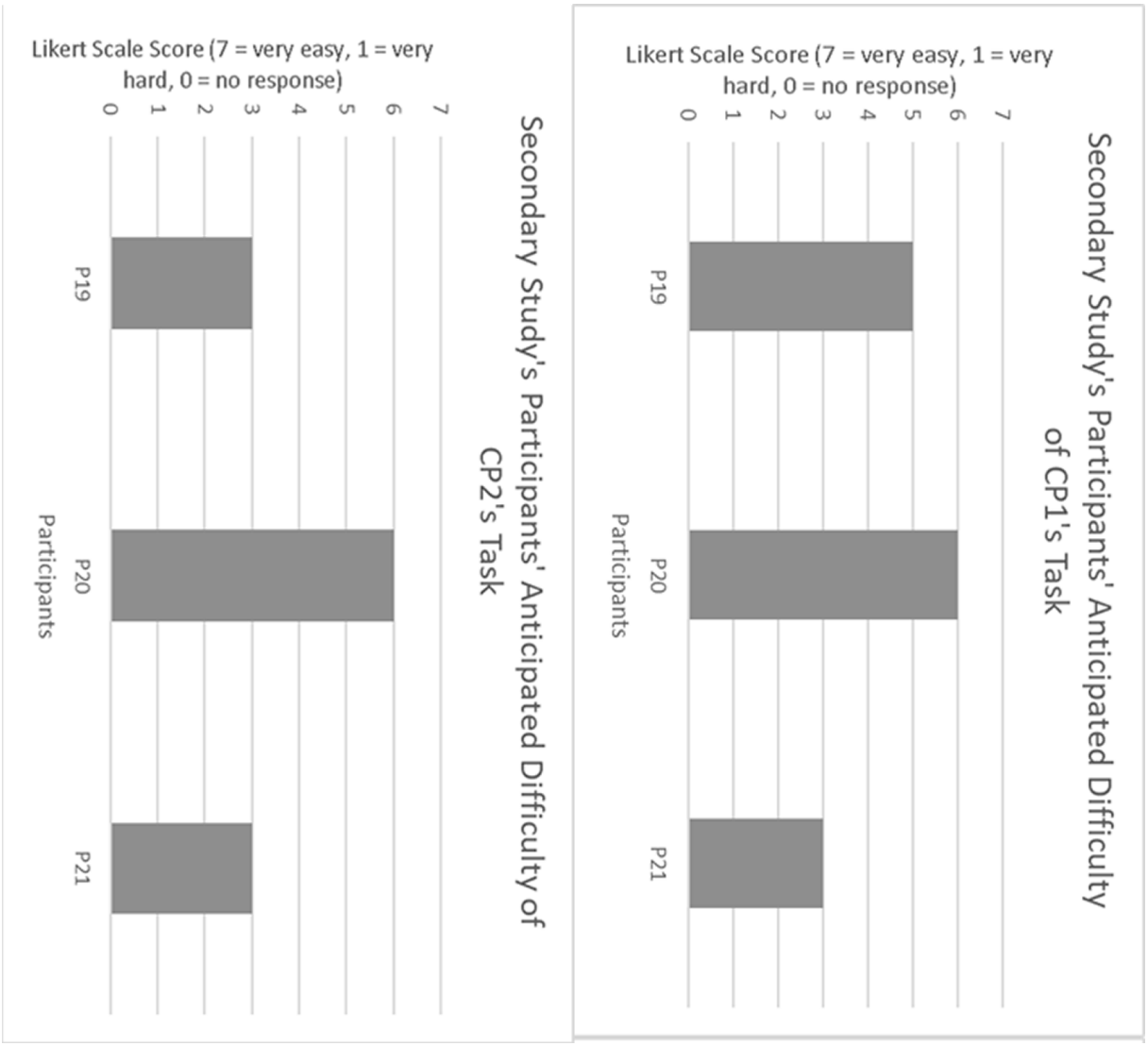


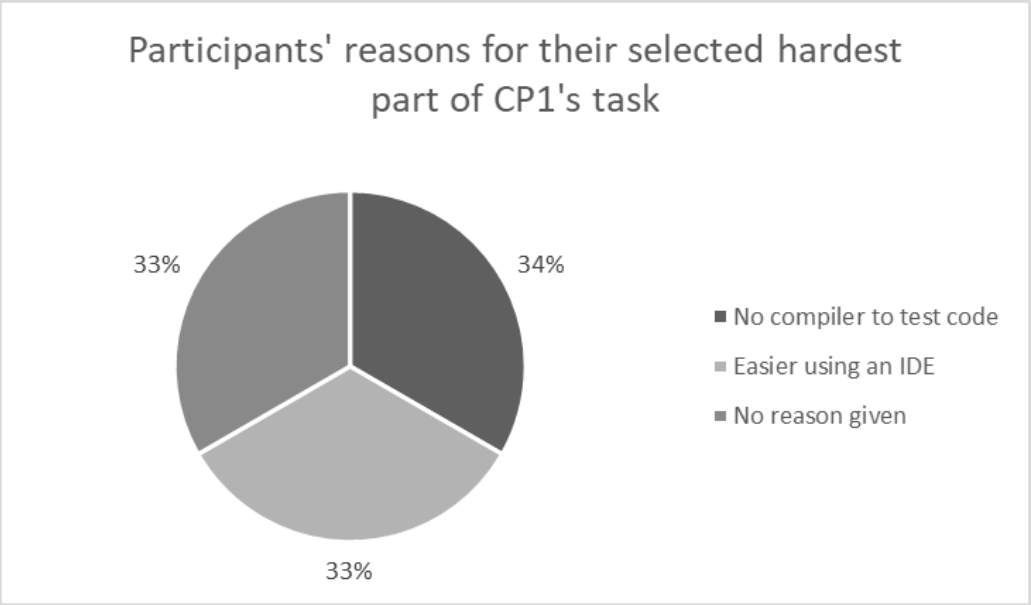
Figure 182: Bar charts for the pre-CP1's (left) and pre-CP2's (right) questionnaire's closed questions on task difficulty for the tertiary study. (Puzzle 1: M = 'Slightly easy' || Puzzle 2: M = 'Neither easy nor difficult')

Participants tended to find different aspects easier and harder – but these do co-align with the aspects found in the secondary study. Participants in the tertiary study tended to be vaguer in terms of the exact part of the task they would find difficult – for example figuring out how things work is not a specific part of the task, but of the process in identifying what to do in the task and would implicate that the process of knowing what is required for the task is a daunting aspect. Secondary study participants were far more focused on the task itself, rather than the whole process which does indicate that despite the levels of confidence exhibited in the background questionnaire answers for the tertiary study, that the participants were of a lower novice level than in the previous two studies. Making a class and declaring fields and methods were all considered easy, which shows that participants were thinking quite broadly in terms of the tasks.

Figure 183: Bar charts for the pre-CP1's questionnaire's thematic analysis on open-ended questions on task difficulty for the tertiary study.

Figure 184: Bar charts for the pre-CP2's questionnaire's thematic analysis on open-ended questions on task difficulty for the tertiary study.

Participants did give reasons for the hardest part – primarily that they had concerns about the complexity of using the puzzles' interface rather than the task itself. This was only proven to be different in CP2 when the focus shifted from the puzzle's interface to the concentration on the Date object and lack of familiarity with such objects.



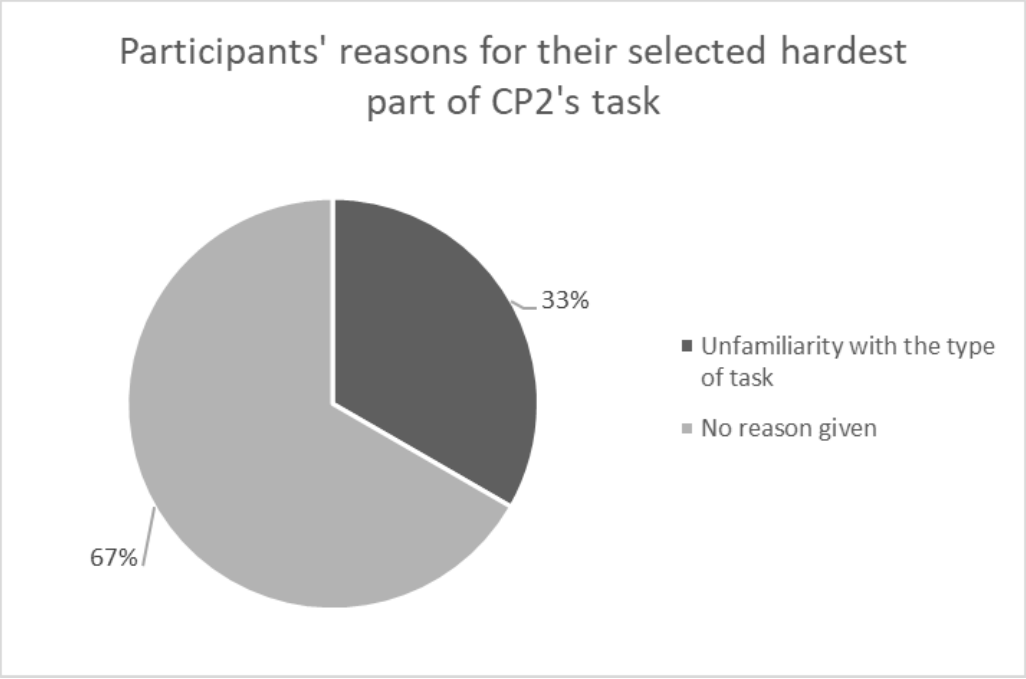


Figure 185: Pie chart the pre-CP1's questionnaire's thematic analysis of why they found the selected item hard on open-ended questions on task difficulty for the tertiary study.

Participants tended to not give reasons for why the part selected from pre-CP1 and pre-CP2 was the easiest part – only one suggested that familiarity with the part of that exercise caused them to believe it was easiest.

Why was it the easiest part?	**Pre-CP1 reasoning**	**Pre-CP2 reasoning**
Familiarity with type of task	1	1
No reason given	2	2

Table 24: Reasoning for why the participant considered the tasks easy.

Unlike the pilot and secondary studies, the tertiary study participants found CP2 easier than CP1 – this can likely be explained by the use of mental load. The way that the interface needed to be displayed on Blackboard gave very little whitespace between pieces in CP1 in comparison to CP2, and despite there being more CP1 pieces, there was less information on each one.

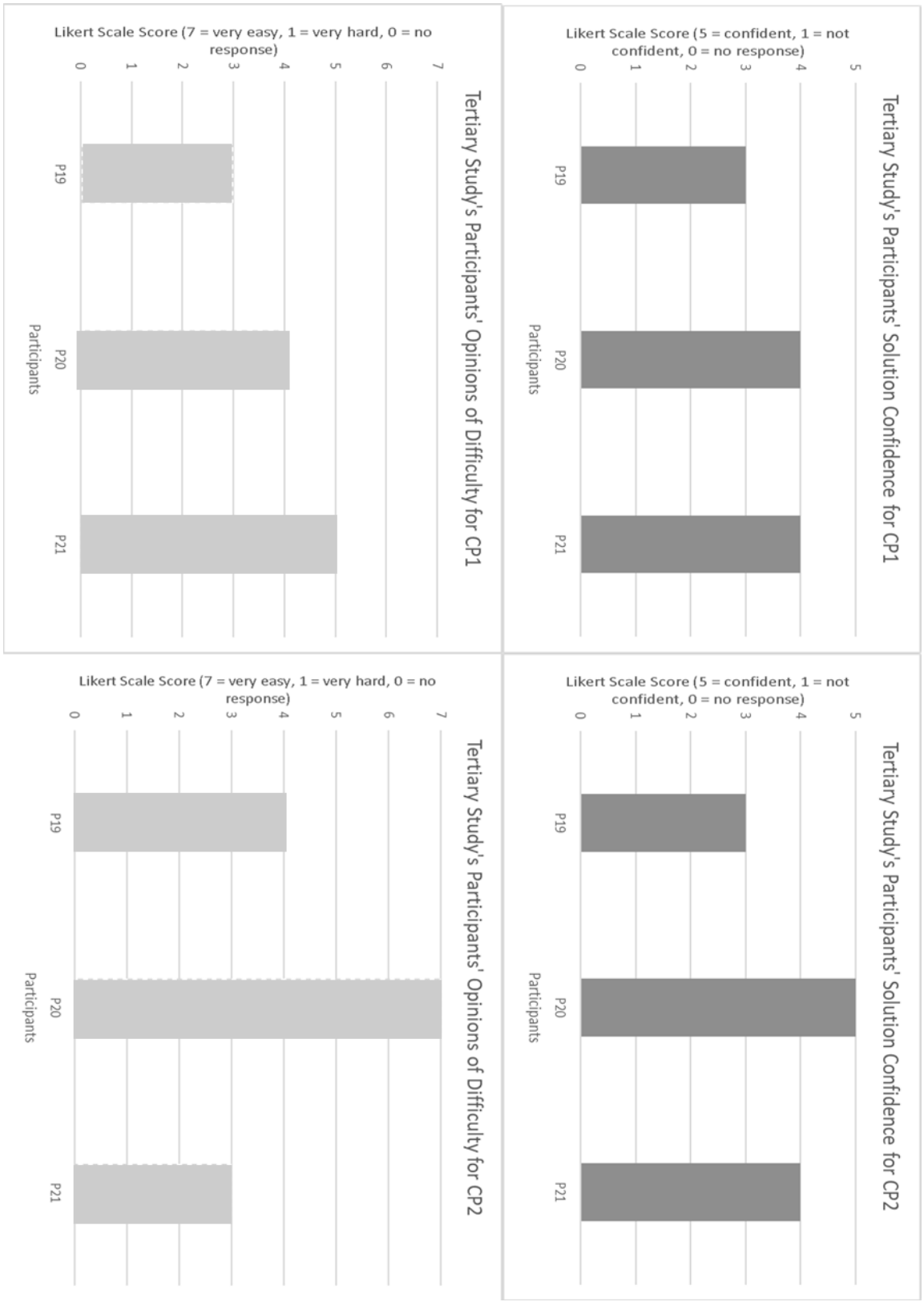


Figure 186: Bar charts of the post-code puzzle question answers for CP1 (left) and CP2 (right).

participants who believed Code Puzzles would be useful to them.

One participant did not return their post-study survey, and therefore only two participants completed the post-study questionnaire. Both participants who answered the post-study survey felt that code puzzles could be used a 'secondary' revision technique, but 1 participant believed it was not suitable as a primary revision tool for the reasoning that they needed to be able to experiment with how the contents of the task would work in a true IDE setting and that they had a hard time visualising the outcome of various pieces in CP1 and CP2. Both participants felt that Code Puzzles would be useful to them, and one believed their approach and understanding of programming had been identified correctly by the observer. One participant felt that their current understanding of the programming concepts was seen, but that the observer failed to fully understand their approach to solving a programming problem because they did not see the way the participant naturally interacted with an IDE – for example, the participant explained that a core part of their approach was in the ability to use the internet to find documentation and tutorials online on how to utilise the code needed, they also explained that they needed to then take that code from the internet and experiment with it before they could fully be confident in their ability to use the code. The observer also believes that part of the issue was that participants relied on running the code as a form of testing, and the ability to 'predict' what would happen when the code was run can be an issue for NPs. However, detecting that an NP needs to do this in order to create a solution can help inform the practitioner on how this NP learns so this tool could still be argued as useful despite it presenting an artificial environment to the NP.

7.3 Discussion

Interestingly, CP1 took longer to complete than CP2 – contradictory to the findings of pilot and secondary studies; this can be explained, in part, by the participants becoming accustomed to the Blackboard Collaborate interface which is evidenced by two participants accidentally wiping their solutions by mistaking the eraser icon to be a tool to remove a single text box rather than an entire board wipe. Participants also were confused about whether the observer could see what they were doing – and sometimes, due to internet connection, this became a problem as their dialogue did not always match to the timestamp the line of code they typed would appear. Similarly, one participant experienced a power cut during CP1 which contributed to the increased time taken to complete the puzzle as they needed to reacquaint themselves with their progress prior to disconnecting. The CP1 experienced, therefore, teething problems that were not equitable to the pilot or secondary study but shed light on the fact that with online environments there are inherent issues with technology whereas in-person observation with paper-based pieces mitigate this issue. Realistically it is not feasible for a lecturer to sit with over 100 CS students and individually perform paper-based puzzles

while observing their every movement, however, for a one-to-one session with a struggling CS student a paper-based option is still feasible. Arguably, if paper-based puzzles were used in lecture halls the participant the observer would be so overwhelmed by trying to keep track of everyone's movements that they would likely gain more information from transferring to an online system such as this.

Participants tended to explain more narrative (reasoning) behind their movements rather than just describing their process like identified in pilot and secondary studies. This shift in type of dialogue may indicate that participants are experiencing less mental load, as they are not needing to perform a major physical gesture to complete the puzzle, they are instead typing – an action they are likely more accustomed to. One participant suggested that the thought of using the puzzle interface was off-putting in their pre-CP1 questionnaire and considered it the hardest part, and another anticipated that it would be easier to code it themselves rather than use the interface. This could be explained by multiple participants – in secondary and tertiary studies – suggesting that the hardest part of solving a programming problem was the process of creating a solution, and that the lack of testing tools (e.g. IDEs) made it difficult implying that NPs do rely heavily on their development environment to point out issues to them. All participants in the tertiary study performed 'mental' testing – where they finally read through their code and tested it in their minds before submitting – and one participant suggested that the lack of compiler was an issue as they would not feel confident in running their code. One participant, similar to the secondary study, also criticised that the puzzles do not demonstrate a realistic mental process for creating a program by suggesting that, as part of their process, they would need to research the materials and libraries connected to the task and test them on an IDE to deduce how the libraries work. By eliminating the use of online tools completely, the puzzles create an isolated environment that is testing their memory of their programming knowledge, rather than their full approach to programming – this is an important limitation to consider, both for the implications of written closed-book examinations in programming and for the use of offline Code Puzzles.

CP2 times were shorter, and participants found CP2 easier than CP1 – and this can also be explained by the way the interface was designed; CP2 incorporated more whitespace between pieces, which participants may or may not have chosen to look at. Participants also created solutions that were not of the same quality of the secondary study for CP1 and CP2 – implying that previous works that have found that NPs can become distracted, go on a tangent, become confused and have not refined their process to design programming are supported by these findings. While in the secondary study participants could create custom pieces, they were often reluctant as a new piece indicated that they were doing something 'wrong' – in the tertiary study, due to the lack of movement of pieces,

participants were creating unique pieces to the degree that the solutions barely resembled the anticipated standard solution. The creation of so many custom pieces indicates that the standard solution, written by an experienced programmer, is not the way NPs view the solution to the problems – in the secondary study, it was implicated by two participants that the way the pieces were written would be alien to them and that if they were writing the class from scratch the participants would have coded the class in a different way. The fact that all the participants used very customised pieces suggests that a wider variety of pieces needs to be presented to the NPs to get a more authentic version of what they would really produce if given an IDE, and could even implicate that the findings of the secondary study, where participants were in large agreement that the study accurately detected their approach and understanding, was influenced by biases such as outcome bias, observational selection bias and social desirability bias. Participants using so many customised pieces, in retrospect, indicates that a study prior to the pilot study, where the participants are asked to complete a task on an IDE and the answers of those tasks formulate the way in which the pieces are written for the future studies, would be, in hindsight, better. However, the secondary and tertiary study findings showed that participants can become overwhelmed by the number of pieces they see and that minimising the selection of these pieces is paramount to gauging the correct level of difficulty – in reality, programmers often have to work with code they have not personally written and these Code Puzzles, therefore, evaluate how an NP can deduce the meaning behind someone else's code rather than their own. The risk of creating a truly authentic task – where a programmer has access to the internet – is also whether the code they produce would be of their own or whether they would copy and paste without understanding the meaning behind every part of the code. Consequently, while Code Puzzles cannot claim to be truly authentic tasks, they are useful tools in deducing how well participants understand the meaning behind code extracts.

Similar to the pilot and secondary studies, participants audibly spent time deducing the meaning behind CP1's puzzle pieces, whereas CP2 did allow participants to apply their own meaning to the piece thus suggesting that the context in which CP1 and CP2 are used is important.

In CP1, the findings highlighted that NPs like to produce methods and lines of code that they think are important, or perhaps when combined with the questionnaire data, suggests that the lines of code produced correlate to what NPs find familiar. Familiarity was a strong theme in the tertiary study results, which indicated that NPs that are not familiar – or have not yet encountered an exact line of code before – are anxious towards that segment and prefer to deviate back to what they already know, even if it was not asked for in the task. Creating print statements and getter methods gave an insight into how NPs program more naturally and offer ideas for how to create pieces that resonate more with NPs than the pieces created by the experienced programmer.

7.4 Limitations and Evaluation

Unfortunately, the study did not attract many participants which resulted in a small sample size. There is also the issue present that the NPs may believe that the experienced observer must be right, even if they secretly disagree with the observer but do not wish to discredit their research. The Date object in Java is not obsolete and is considered a difficult class with multiple methods that the students were not familiar with – despite reassurances that the Foundations module covered the ability to read Java documentation, this research found that the students preferred to write code with basic operators rather than attempt to use the after method – no students in this study chose to use the Date object, and instead, chose to use integers.

Unlike the secondary study, the tertiary study needed to incorporate the feedback after each puzzle in a similar fashion to the pilot study, only recorded. However, this feedback consequently influenced CP2 – for example, P20 was curious about the way in which 'this' worked, and as the observer had picked up on their silence when the word 'this' had been written and asked if they knew what it was, they responded that they "did not know" it was something that they had learned a while ago and had always just been "there". When the participants asked the observer to explain this, they did technically admit that the puzzle had highlighted an issue that they were unaware of – but this did also mean they used 'this' in the correct way during CP2 whereas if CP1 had occurred prior to CP2 this may not have been the case. Therefore, there is likely a follow on bias where the previous answers influence CP2.

The results of the accuracy of the study are generally inconclusive – one participant did not submit their post-study results, and as there are only two participants, it seems unclear on whether the findings were truly accurate or not. There is some evidence to suggest that the participants found the session useful, as all who returned their questionnaire suggested so, but there is not enough data to show the conclusiveness of this finding.

Participants did highlight what could possibly be the issue with using Code Puzzles as a learning tool through their commentary – for example, the participant who suggested their approach was not captured fully as they were not allowed to look at online resources or test the code prior suggests through this feedback that if students are tasked with solving a puzzle they will not 'see' the outcome of their code and learn from it in the way that writing code in an IDE would achieve. Instead, Code Puzzles only capture how participants think the compiler works – and how they perceive the standards of Java naming conventions and what those implicitly indicate the purpose of pieces should be. Therefore, Code Puzzles are useful to assess the way some NPs view code, but not as a learning tool.

7.5 Conclusion

Ultimately, the tertiary study demonstrated the technical difficulties of using an online interface during observations and explored the realistic feasibility of incorporating Code Puzzles into a pre-existing learning environment such as Blackboard Collaborate Ultra. The study did appear to highlight the mistakes and issues that participants had, but as suggested by the feedback, participants feel that offline Code Puzzles do not encapsulate their natural approach to programming. Participants that are not allowed to look at online sources, or 'test' how the code works prior to using the puzzle, argue that this is a vital step that is missing and needs to be addressed.

7.6 Chapter 7 Summary

This chapter documents the data produced by the tertiary study and compares it to the data collected from previous studies to see whether observing a participant's typing is as effective as observing their interactions with specific puzzle pieces. It was concluded that Code Puzzles were more effective for determining understanding, however, it is acknowledged that having only three participants for this tertiary study is a substantial limitation. That said, this thesis defends the usefulness of this study in the way that it showed how confident novice programmers struggled with creating an effective solution. It also highlights the realistic practical implications of transferring the study to an online learning and teaching tool and therefore allows this thesis to present a series of recommendations were this to be incorporated into a virtually delivered computer science module.

This chapter presented, explored and evaluated the findings of the tertiary study.

This study assessed the feasibility of using Code Puzzles as a diagnostic tool kit via an authentic online learning environment (Blackboard Collaborate Ultra), with the conclusion that it is feasible to obtain some form of insight into the NP's understanding of programming, but that the restrictions that an explicit set of Code Puzzle pieces have contribute to not clouding the interpretation with NP's own custom pieces. The workspace phenomenon was not observed during the study, which suggests that the workspace only naturally occurs in paper-based studies or interfaces that would allow for free movement of puzzle pieces.

Limitations in the study's design became apparent, but were required due to COVID-19 restrictions and at least presented a baseline reading for how well an expert could obtain data from just viewing the way a student types code.

The tertiary study highlighted the importance of online tool's interface design and the current limitations of transferring code puzzles to a current educational tool such as Blackboard Collaborate.

This study offers a point of comparison between the use of code puzzles verses the use of a text-based editors and weighs up the pros and cons of using both.

In conclusion, the tertiary study revealed that a certain amount of understanding can be obtained from the listening to NPs as they type on screen, but that even a full automation of this aspect would be difficult without a large sample investigation into the types of customised pieces that NPs would create based on task descriptions. Therefore, automating analysis is far more realistically done with Code Puzzle pieces but the limitations of current interface designs for 2D Parson's Problems need to be addressed to observe the movements necessary to extract the approach and understanding of NPs.

Chapter 8: Collective Discussion: Workspace and Think Aloud Protocols

8.1 Conception and Implications of the Workspace

A novel discovery showed that 7 participants (33.3% of all participants) naturally chose to organise Code Puzzle pieces into groups prior to placing them in the final solution space we have named this area the 'workspace' and it is a physical space that either replaces the randomised solution space or exists between the randomised puzzle space and the final solution space (see Figure 187).

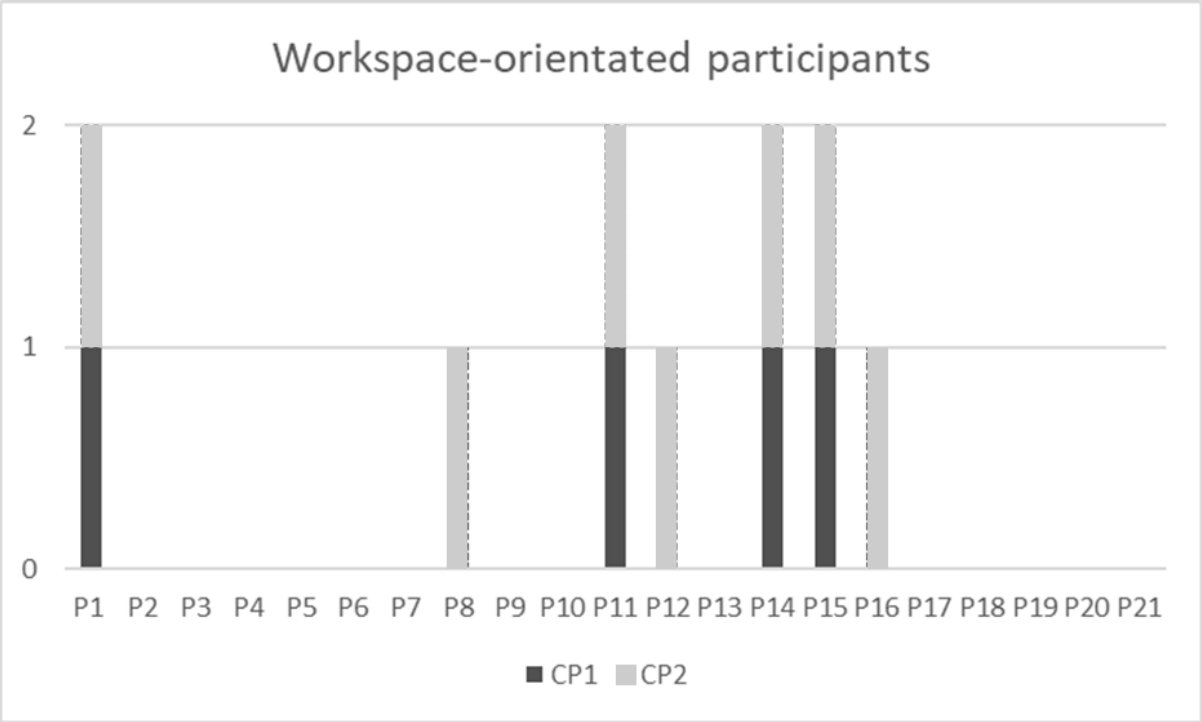


Figure 187: Number and type of participants who were identified as workspace-orientated participants.

The workspace is created when participants group non-identical pieces together (see Figure 188). Both non-workspace-driven and workspace-driven participants grouped identical CP2 pieces together, but this was done to manage the large number of pieces on the table. The reasons for establishing groups with non-identical pieces were based on: perceived similarity between the concepts behind the pieces (the most common reasons); and perceived similarity of context in which the pieces would be used based on the problem description. Figure 188 shows how the design of 2D Parson's Problems might be modified to include a workspace for participants to use to organise their pieces.

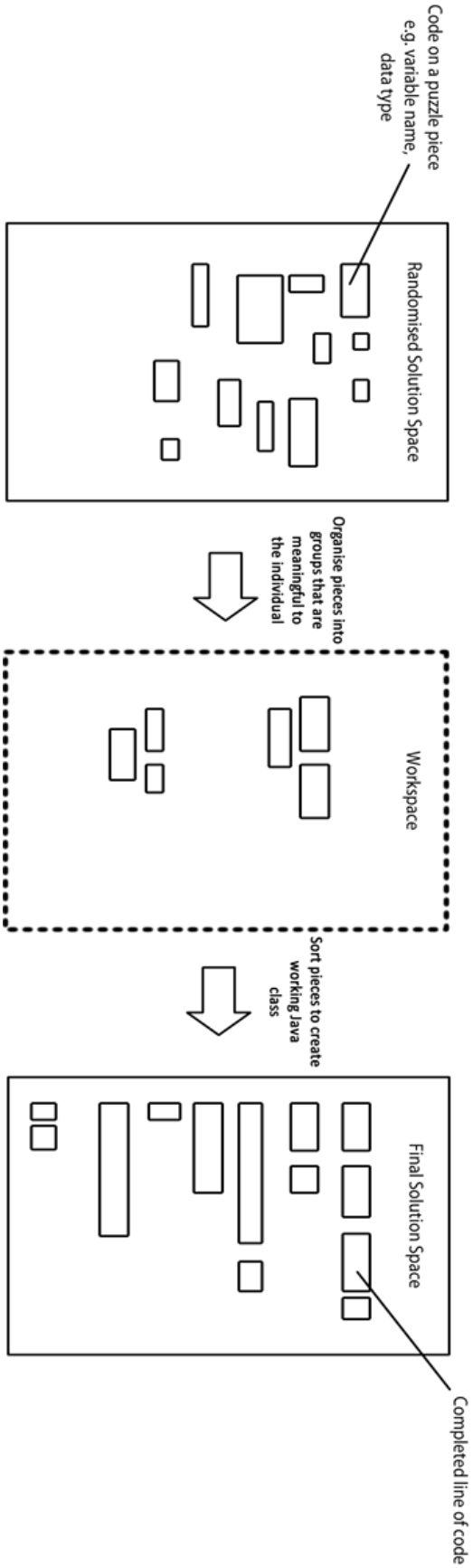


Figure 188: Our proposed workspace design.

We propose the incorporation of the workspace because we found its use to be common and that the participants gave similar reasons for using it. Workspace-driven participants' reasons for utilising the workspace were given as: promoting readability of the pieces; making "sense" of the pieces; making pieces easier to find; and/or reorganising the pieces in a way that they can understand. Despite similar reasons, participants laid out the groups in different ways holistically – although some groups consisted of similar pieces for similar reasons. Therefore, we believe that the workspace is an important area to include in the puzzle design if the expert wishes to understand how the participant relates concepts to one another, and how and why the participants interpret the pieces in the way they do. Workspace-driven participants felt the need to explain their actions as they felt that were performing a weird, unexpected action that the observer would scold them for – with one participant feeling that they felt like they were "wasting time" in grouping the pieces despite it not being a timed exercise. While the observer did sometimes query the reasoning behind participants' movements, participants felt more naturally inclined to explain the reasoning behind grouping pieces together rather than repeatedly explaining the grouping movement itself. The change in tone and style of the dialogue meant that the participant revealed how they viewed the piece more holistically than just the context in which the piece would be incorporated into the class and therefore gave an insight into: a) what the NP believed the piece represented, b) what the NP was noticing in the structure or way that the piece was written, and c) how the NP views and relates programming concepts to one another. In comparison with a computer-based drag-and-drop implementation, information on how the participants view pieces cannot easily be inferred by monitoring a cursor click or drag and drop movements, because such movements do not give any direct information about how a participant perceives the meaning of a particular piece.

While the traditional 2D Parson's problem design expects participants to organise elements inside the final solution space, for some participants, it is clear that there is a missing step that can be explicitly introduced by using a workspace. This is crucial for studies aiming to use Code Puzzles to gain insight into participants levels of understanding, as such studies need to understand the overall process of constructing a piece of program code as well as how the participants view the pieces placed in the code. Our studies demonstrated that participants who did not use the final solution space to organise pieces as they did not tend to move pieces once they had placed them there. Therefore, participants either chose to organise their pieces prior to the final solution space (in the workspace), or chose to place the pieces from top-to-bottom in the final solution space. It was rare for participants to use a workspace for CP1 where pieces contained full lines of code, because there were too few pieces and those pieces contained more contextual information. A participant noted that the CP1 felt like someone else had written the code, and consequently, they did not feel it

resembled their own approach to solving the task – in CP2 where the pieces were smaller and contained less contextual information, participants were more able to claim ownership over their solution and the task in CP2 required participants to extract and apply meaning to each of the pieces to determine how to construct a solution and we believe this is why participants found a workspace to be useful.

Ultimately, it is crucial to amalgamate the results from across the studies and determine the usefulness, impact and effectiveness of the workspace in comparison to traditional 2D Parson's Problems. This led us to propose a new form of Code Puzzle that explicitly asks programmers to group code extracts based on perceived similarity and/or problem context because we believe that process would give us an insight into their understanding of programming constructs.

8.1.1 Analysis of the use of the workspace

Asking participants to explicitly use the workspace to group pieces may provide a way to make it easier to distinguish groups and their members, and also to understand the meaning behind the groupings that participants choose.

To analyse the benefits and issues of using the workspace, we need to consider: how we identify a grouping, identify which pieces tend to be grouped or not, and how participants explain the grouping. The boundaries between groups were often fuzzy due to the nature of paper-based pieces so it is sometimes difficult to infer whether the distance between groups – or even clusters of groups – had intrinsic or accidental meaning. Two participants stated that the relative position of each group had meaning, although most did not audibly acknowledge that the relative position of each group had a meaning behind it. Participants were surprised if the observer asked them to explain their groupings, which indicates that the participants may have assumed that their layout made sense to the observer innately – this phenomenon is explained by Schutz (1967) as the participants assumed their own portrayal of how they link concepts would make sense to an expert. In reality, classifying what a group is or where a group ends and another begins is difficult without the participant's explanation – groups may appear the same between participants, but the reasoning or vocabulary used to explain the programming concepts behind the pieces can be different. This is why it is important to obtain a participant's individual labelling of groups and elements, as it provides insight into how they identify, understand and approach programming constructs.

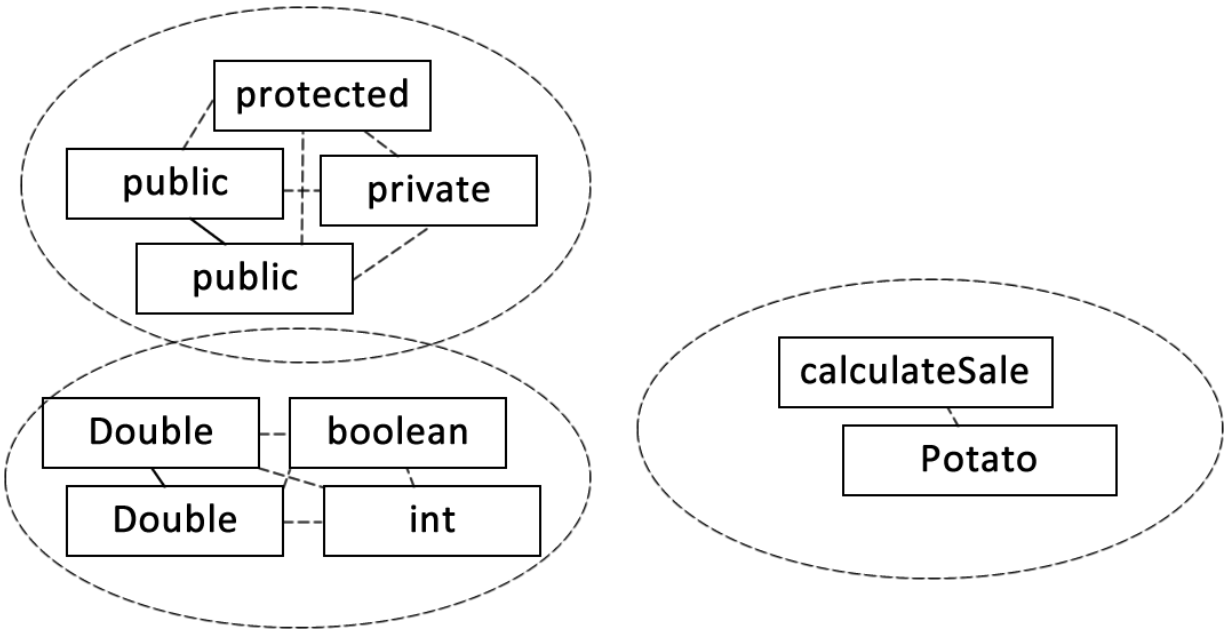


Figure 189: Hypothetical example of 'grouping' of pieces in the workspace and the difficulties between identifying group boundaries.

Figure 189 demonstrates the issue with identifying groups with a software implementation

we could identify groupings based on clustering algorithms (such as using K-means with Euclidean distance) which provide a basis on which to establish groupings, although this does not completely remove the decisions of whether a group of pieces represents one group or more than one.

The centre of a group was difficult to determine – participants used column-based grouping and cluster-based grouping which would require different centres to determine the group boundaries. Whether the centre originates from the centre of a piece, or a centre of a cluster of pieces, is important so that a machine learning algorithm can more accurately determine which group pieces belong to. Therefore, the variation in the ways the pieces were grouped suggests that the user themselves is the expert in determining what is grouped and why, and any interface that incorporates a workspace needs to ensure that the user determines and labels the groups. User-based classification can be achieved through an intuitive user interface, that allows the user to see that placing a piece next to another assimilates the border/shape/colour of the adjacent piece in the same group and allows the user to adjust the clusters to better match their grouping preferences without causing cognitive overload as labelling each piece and assigning it to a group manually would become tedious.

The proximity between groups was also of interest to a subsection of workspace participants – close proximity of groups suggested that the participant viewed the elements as being linked in some way (e.g., similarity of concepts or similarity of context that the concepts are commonly conjoined in),

whereas more distinctly separated groups commonly showed that the concepts were not so closely related to one another. There is also the possibility of groups within groups, but not enough data was obtained to clarify how the boundaries would differ, be detected or frequently at which subgroups were used. Arguably, the number of subgroups identified relates to the number of interlinking concepts available in the puzzle itself and the puzzle content in CP1 and CP2, respectively, was not altered to incorporate a different number of concepts inside the puzzle. One participant explicitly addressed this by suggesting that elements of a similar nature would be easier to find if placed close to one another – but there was no noted, consistent 'group stopping' rule whereby a group would be distinctly separate from another.

Understanding the participants' reasoning behind their actions in the workspace is important to capture their mental models of the underpinning program concepts of the pieces they are grouping, as workspace-driven participants often referred to groups as program concepts – e.g., 'public' meaning 'visibility'. A communication barrier occurs between tutors and NPs when terminology is used – NPs may perceive terminology as 'jargon' and tutors may misunderstand the meaning behind the terminology used by NPs in return. Our proposed workspace acts as a medium whereby the tutor and the NP focus on the groupings without initially worrying about the terminology used to describe the grouping. Using groupings therefore reduces the communication barrier by providing a platform to observe concepts and grouping pieces in the workspace also has the benefit of reducing the cognitive load on the participants.

Understanding the participants' reasoning is key to gaining an insight into their terminology and understanding of the purpose and meaning behind programming concepts. Holistically, NP understanding is comprised of both correct and incorrect interpretations of programming, therefore symptoms of both understanding and misunderstandings are crucial to understanding NPs. There is understanding exhibited in the movements of Code Puzzle pieces and also in the interpretation of the task description for the code puzzles – although little interpretation of the task description was observed in the studies, aside from the tertiary study where P20 chose to highlight key words and values in different colours to symbolise their interpretation of what is relevant, and important, to the task.

.

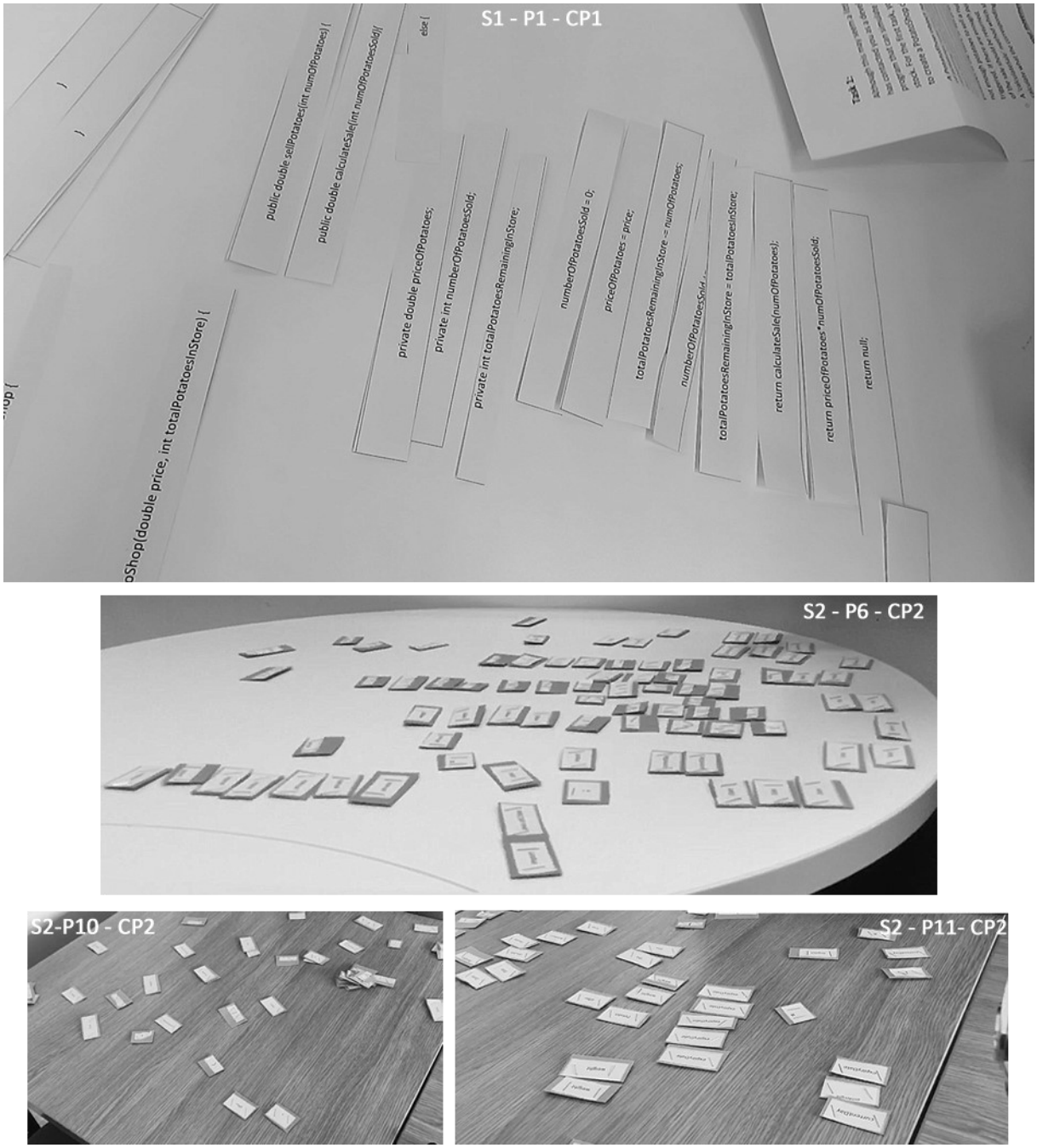


Figure 52: Examples of the different forms of workspace organisation – S1-P1-CP1, P6-CP2 and P11-CP2 show columnized grouping, whereas P10-CP2 shows spaced grouping.

Participants that used workspaces were observed to lay out the pieces in individual ways – as noted by Figure 191, there was a popular columnized form of grouping which meant that related concepts were placed either in a vertical column or horizontal row depending on the room provided on the desk. Columns and/or rows were typically separated by padding (room allowing), and the participant would audibly speak of the pieces as a set group. Individual columns were typically labelled as being able to see how many of the same piece existed for the puzzle so that, presumably, the participant could plan ahead as to how many times the piece needed to be used, or by perceived similarity of

concepts – where the pieces all related to an audible concept, say, the visibility in Java where public and private pieces were placed together or side-by-side. Individual columns varied in length and order, and no one participant placed the exact same pieces in the exact same space on the desk implying that the pieces were ordered in a way that the participant saw fit. That said, certain pieces that related to similar concepts – such as the visibility or data type group – were placed next to other related concepts – such as variable name group. The placement of groups itself is an issue to distinguish mechanically, as the distance between individual pieces is small. If a program was created to incorporate a design that allowed participants to label different groups and ask them to group pieces together, this would be a way in which to clearly differentiate groups with the participant's reasoning made clear. However, automating this in a free-ranged space would be challenging due to the group proximities. Therefore, it is difficult to define the exact distance that a cluster ends and starts without the participant's audible or written words explaining the groups. This thesis argues that the concept of grouping puzzle pieces does give ample information to an expert without causing cognitive overload as the participants feel they are organising the concepts in a way that is meaningful to them inherently implying that if an expert physically sees the participants grouping the pieces that they would gain some insight into the way the participant views programming concepts and the way they understand the standards of the language the code puzzle is written in.

8.1.2 Analysis of how workspace and non-workspace participants spoke about Code Puzzles

For the post-code puzzle questionnaires, the data was combined to indicate whether participants inherently found differences between the two styles of code puzzle (see Figure 190).

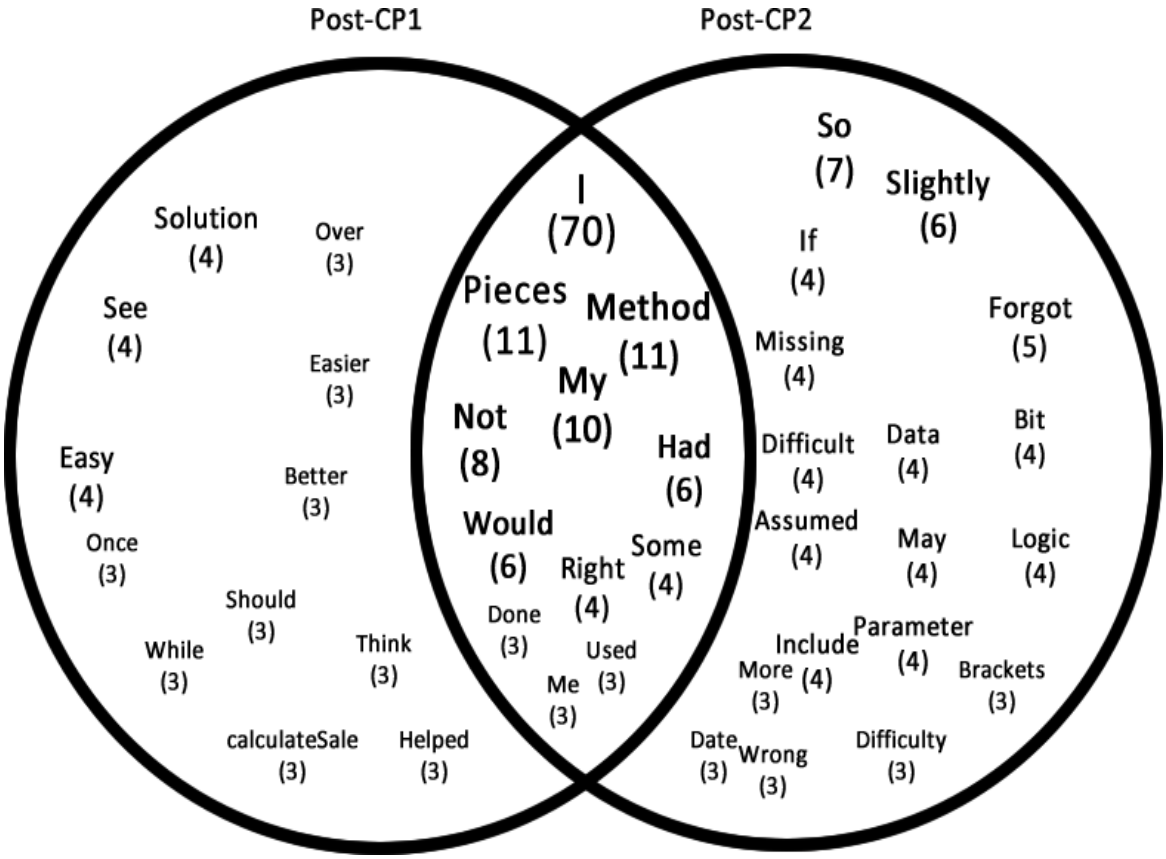


Figure 190: Vocabulary used in the post-CP1 survey (post-CP1) and post-CP2 survey (post-CP2). Only words mentioned over three times were included and prepositions and conjoining words have been omitted alongside punctuation.

As suggested by the diagram, participants spoke in the first person for both puzzles, and referred to pieces and methods when discussing the quality of, and confidence in, their solution. It can be assumed that the intersection of words, common throughout, are not of particular relevance. CP1 was generally considered 'easier' than CP2 which was more 'difficult'. Participants found that they felt they had 'forgotten' and were 'missing' pieces from CP2, whereas this phenomenon was not common in CP1. Forgetting pieces is not unusual for CP2 – there were over 70 pieces for that puzzle in comparison to 24 pieces for CP1, and they were smaller in size. That said, this indicates that CP2 contains more data as more participants were focusing on technical aspects – such as 'bracket', 'parameter', 'data', and 'logic' – which is likely to reveal more than participants who were focusing on the entire 'solution'. One participant (S2-P10) from the secondary study quit CP2 early due to being overwhelmed, though, so this suggests that there needs to be a careful gauge – or incremental policy – towards testing puzzles in the toolkit as if the toolkit is set at too high a level the participant will become overwhelmed and limited data will be obtained other than that the participant failed to comprehend the concepts or is unconfident about applying those concepts. S2-P10 completed CP1 without too much difficulty but constructing the solution from scratch was a problem. This is likely why the common words in CP2 are more focused on small bits of programming logic that caused

issues for individuals and participants are noted to be more hesitant than in CP1 – ‘may’, ‘assumed’ and ‘wrong’.

While the participants who took part in the background questionnaire did highlight other aspects, such as: creativity, testing and debugging, having the right temperament (patience, perseverance etc.), and learning programming from a young age, participants only reflected on the knowledge recall and problem-solving characteristics when assessing how confident they were with the solution. This is expected, considering participants cannot test paper-based puzzle pieces to see if their solution is correct. Most participants who cited personality and temperament traits as important qualities believe that they already possess those characteristics, but that they needed more experience to improve on those characteristics and gain more confidence to enhance those personality and temperament traits. Additionally, the aspect of creativity, is difficult to reflect on and participants that mention they should have done things differently are likely relating to this aspect while also relating, primarily, to their approach to the task.

NPs in the secondary and tertiary studies who explained their answers to the background questionnaires provided the richest insight into their lived experiences of programming in conjunction with their interactions with the code puzzles themselves. Participants often clarified terminology that was shared between candidates, for example, the term ‘patience’ was used by 2 participants in the secondary study and also mentioned by 1 participant in the tertiary study – it was explained fully by 1 participant that “programming requires a lot of patience due to the amount of code you have to have to go through. As code becomes bigger, it becomes more complex, and you need to work a lot harder to fix errors”. Phrases such as these provided a rich insight into the lived experience of the participant regarding programming, it revealed the reasoning behind why the participant perceived such an aspect as important and gives the observer a brief insight into the typical challenges the NP faces when programming. For example, from this participant it is evident that the participant expects to have errors in their code and that, with more code, more errors arise. This can be taken one of two ways – the participant may be struggling to code, and as a result, creates errors that the participant struggles to find the root of, or, that the participant is practicing a test-first approach where they create a series of unit tests that initially runs with errors, and creates the minimum amount of code to fix these errors and make the program work as intended. That said, with this participant, it becomes apparent that the first issue is likely the cause of needing to fix errors as they then explain that their approach to generating code is to “use StackOverflow or open up another project” and attempt to transfer the logic from an unknown user to their own work. This offered a great case-by-case insight into the way participants coded, and what they struggled with, and could be compared as a form of baseline value to the way they interacted with the Code Puzzles

themselves. That said, a criticism of this is that way participants coded on a development environment could not be directly transferred, and thus directly compared, to the way they arrange Code Puzzle pieces as – like multiple participants commented on the post-study and post-puzzle questionnaires – they did not have the ability to "research" or use a "development environment" and it is now common place for programmers to have access to a vast array of online resources meaning that coding in an isolated environment was unrealistic in that regard. On the other hand, it should be argued that if an NP truly understands programming in Java, they should be able to code without the guidance or reminders of internet sources.

The task was purposely designed so that participants drew upon concepts they learned during the course of their Java foundation modules in their CS undergraduate degree, however with CP2 there was a controversial section where the Java object, Java.util.Date, was used and multiple participants reported that this was an issue because they had never encountered a Date object before and did not how to use, or did not feel comfortable using, the Java libraries. In the tertiary study participants were allowed to use an alternative approach, instead only needing to compare two integer values to see whether a potato had expired rather than using the Date object, and all participants chose to use integer or String values rather than use the Date object when given the choice. This, however, does reveal quite a bit of rich data – after all, the task description did provide an extract from the Java libraries that had been taught during the Java foundation module in the undergraduate year, and that extract did include information about the comparative method that was intended to be used, i.e. "date.after(Date otherDate);". Using the after method in the Date class was not foreseen to cause such issues as the participants should have been familiar with the concept of comparing objects (i.e., they were familiar with comparing integers, floats and doubles using the '==', '>' and '<' operators, and familiar with comparing Strings using 'equals') and the idea that these conditions produce a Boolean that can be used in an if statement. Participants were also allegedly taught how to interpret Java documentation and were seemingly familiar with the concept of what a date is in terms of food being fresh or not (i.e., if the use by date of the potato was 7th July 2020, and the current date was 9th July 2020, then the potato would be 'off'). However, multiple participants felt uneasy about utilising any documentation from the Java Oracle Libraries with the tertiary study participants choosing to use their own creations instead due to a lack of confidence in the underpinning concepts and the 'skill' to read Java documentation, which, arguably, is an important skill for an NP to have if they are to use programming approaches where they utilise the information available in programming forums to create their own solutions. While teaching the basics of Data types had been achieved well, the principles of using external libraries or sources that utilised the knowledge of the basic Data types was not achieved and that even a simple comparable function caused difficulty. This may indicate

that there is hierarchy of pre-requisite knowledge and a need to consciously identify, associate and interlink relevant concepts to new, unforeseen concepts that had not been achieved when the participants contributed in the study.

Participants, when interacting with the Code Puzzles, typically voiced action commands when asked to speak their thoughts indicating to the observer that they viewed programming and building a program as a physical activity rather than as purely a mental activity. While it could be argued that Code Puzzles are an activity where actions are required in order to complete a puzzle, the same could be said for the activity of typing code into a development environment as well. It was rare for participants to explain meaning behind the phrases, as, the think-aloud protocol tended to be interpreted as explaining the movements themselves rather than the meaning behind those movements. This suggests it is not natural for the participant to explain the meaning behind pieces, or to identify the concepts of individual pieces, without prompting. Instead, participants typically named or labelled the pieces and said where they should be placed, e.g. “Next we are going to create the constructor” – the dialogue suggests that there is a name for the type of method being created and the participant recognises what it is, but, they have not formally defined the term ‘constructor’ and do so by placing code that they believe is the constructor. This worked as intended, as, participants quite clearly demonstrated whether they knew how to, say, construct a constructor method and typically then went on to “initialise the variables” while placing these within the constructor. Though implicit, and not a burden on the NP’s cognitive load, the puzzles acted as a medium by which the observer could view the way they experience and approach the world of programming without directly interfering with their space of learning. The observer could tell, for example, that participants who declared a constructor and then went on to initialise the variables within that constructor, understood the premise of what a constructor was and what it needed to do in order to be a constructor without the participant physically having to explain this out loud and artificially burden their flow of thought.

A critique of this is that the observer did need to interpret what the participant was doing without interfering or impeding their thought processes, which meant that the observer needed to follow the participant’s movements and reasoning while not querying their movements. In the pilot study, a mistake was made by the observer where they queried the reasoning behind the participant’s movements to try and gain an insight into their lived experience of programming, however, this made the participant feel as if the movement they performed or their reasoning for that movement was unexpected in some way and they became conscious of trying to adhere to the expectations of the observer as a result. Therefore, it was discovered in the secondary and tertiary studies, when the observer was given an observer script (see Appendix) that participants did not realise if they were

performing unusual actions, but it did mean that the observer, during the follow-up after the study, had to remember these actions and ask about the meaning behind them after the puzzles had taken place instead of during. As specified by Marton (2000), it is usual for phenomenography studies to use interviews in order to obtain a realistic interpretation of the lived experience of the participant, these studies collectively used scripted observation of a controlled series of puzzles with unscripted interview elements in the follow up to the study.

The majority of participants believed the observer's analysis of their approach to programming and/or their understanding of programming was accurately identified by the observer after the experiment. However, participants were told in the advertisement that the observer was a 'Java expert' and this may have led to the participants agreeing with the expert to: not invalidate their research as the participants knew the research was conducted partly for a PhD (sponsor bias); not challenge authority by disagreeing with the expert as the participant may want to appear knowledgeable and friendly (social desirability bias); and/or that the analysis was general enough for the participants to feel like they could relate to the observer's analysis despite it perhaps being applicable to other programmers and not specifically them (the Barnum-Forer effect). The Barnum-Forer effect is particularly troubling, as some of the observations that the observer stated could be widely applicable (e.g., "you started with declaring the class, followed by establishing your fields" – which the data from all three studies suggests this is not a unique phenomenon). In retrospect, it may have been better research design to have: not specify that the observer was an expert; ensured there were two interpreters at least to be able to see if the same analysis would have been achieved; and send the observations by e-mail for the participant to have time to reflect on them. Although such design considerations may have caused a poorer uptake of participants and more participants choosing to not submit the final questionnaire as observed in the tertiary study. In this research, the intention was to attract participants by offering them a chance to practice their programming skills in front of a perceived expert in order to obtain tailored feedback which was not promised to be accurate but might be enlightening. Therefore, the intention of the observer was to give tailored feedback, particularly on any noticed problem areas of the participants, so that participants felt they had gained something of value out of the research as well. While a minority of participants did not find the puzzles difficult, and therefore had little feedback from the study which is shown in 2 of the participants' post-study survey data, some participants did require ample feedback. The way the feedback was given at the end of the secondary and tertiary studies, participants were grouped into a set of out-dated categories – such as modular, nested, and linear – which tried to be then individually tailored to how the participant created their solution. Linear participants were very closely related to the Linear Programming Process where they followed the specification to the word,

to the degree that they were never misled by any red herrings, modular participants were very closely related to nested and consequently Structural Programming Process where they typically focused on filling out the methods and may not always start with declaring the variables first, and finally nested participants were closely related to the Structural Programming Process where they viewed methods as being inside a class and typically started by defining the class followed by declaring the variables, the constructor and then filling out the internals of the methods. Due to the simplicity of the sample Code Puzzle solutions, which typically only used one or two methods and had a task description that was closely in line with the type of tasks they had seen in their Java programming modules where the information was laid out in a linear fashion, it became difficult to identify clear-cut categories of description and critical aspects for these elements in retrospect. Despite participants largely agreeing the observer's feedback and the accuracy of identifying their approach, it is possible that there is an element of the Barnum-Forer effect at play with the extremely high accuracy readings. That said, when listening to the audio recordings of the follow-up meetings the participants are getting very specific feedback based on, not only their process, but also the programming concepts that are dealt with in the puzzle. If the puzzle had been excessively long, then, it likely would have deterred participants while also not necessarily yielding extra information about the exact way that they approach programming. It therefore suggests that while the Barnum-Forer effect may be present, it is only attached to the modular, nested and linear classification of participants rather than their reasoning behind the assigned classification or the accuracy of their understanding (and misunderstandings) of the underlying programming concept principles highlighted in the Code Puzzles.

Another reason for why there is an extremely high accuracy reading for detecting their understanding is that participants were aware from the announcement (see Appendix), it is possible that they also perceived the observer to be a form of authority on the matter and trusted their judgement more than their own. In other words, they did not want to disagree with the researcher or observer and make their thesis out to be pointless – there can be factors of social desirability and social acceptance that come into effect in this situation. As there was only one observer present, who was the same as the researcher themselves, this further complicates the issue and may be partly to blame for the high accuracy reading. It was difficult to check or recruit another observer due to the limitations and restrictions associated to the degree.

Unfortunately, when datapoints were collated and analysed a clear theorem was not generated from Straussian Grounded Theory – participants were very vocal about their process of constructing the solution, and the way in which the solution was formed tended to be similar between participants; for example, participants typically started with declaring the class, followed by declaring the fields,

followed by the constructor, followed by initialising the fields and then constructing any methods that were asked to be constructed during the task. This meant that the vocal data given only tended to give insight to the participants' understanding when they vocally wished to make it known. Participants who were unsure of concepts, such as P19, tended to keep quiet and not talk about a specific piece or aspect when explaining their movements to the observer. Therefore, there are several different open codes, and while the datapoints are saturated, the codes do not easily form a coherent theory (see Figure 191).

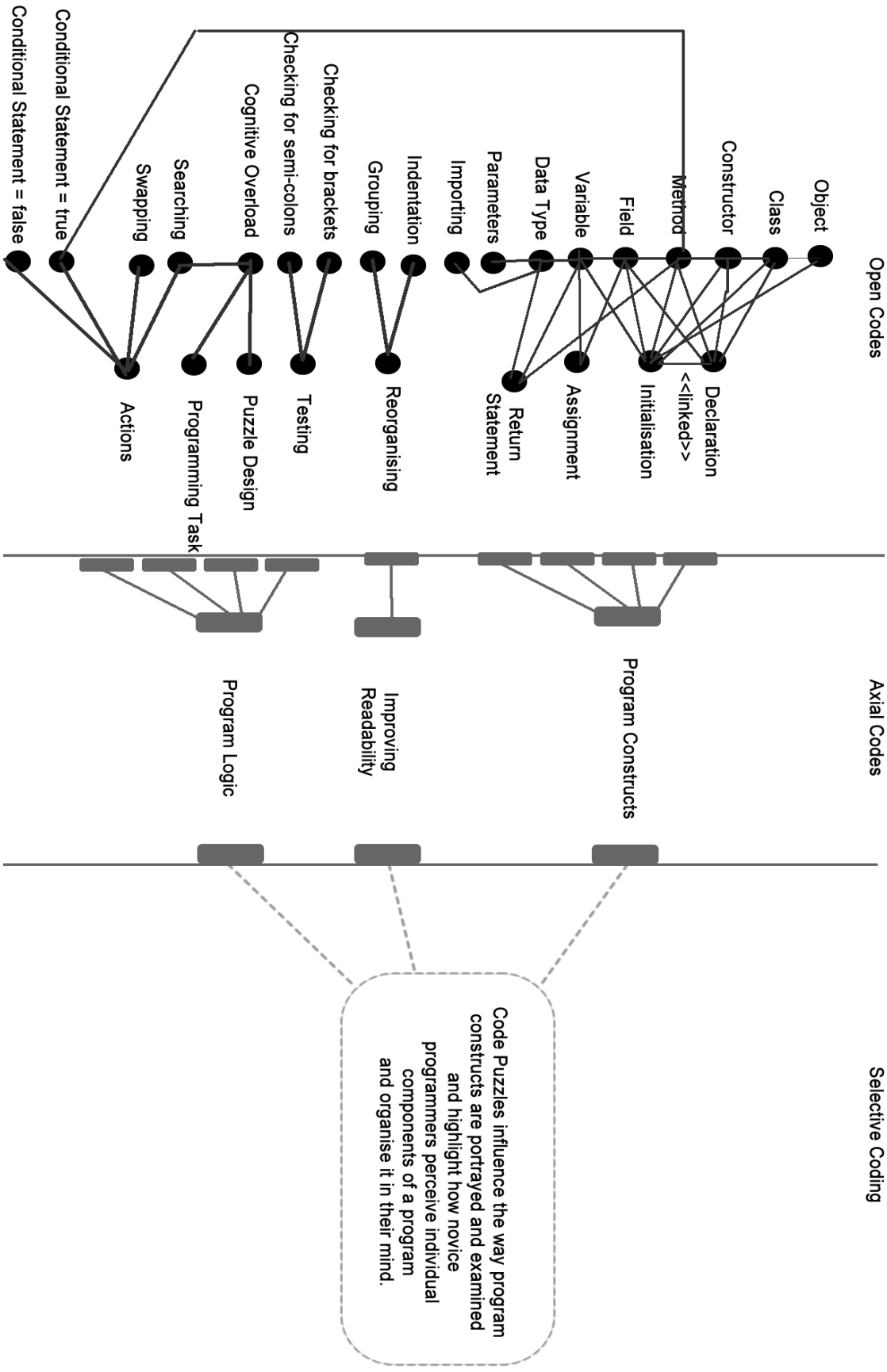


Figure 191: Straussian Grounded Theory applied to the verbal transcripts for Puzzles 1 and 2 for the Pilot, Secondary and Tertiary studies.

8.2 Research Output

Based on the findings of our research, the research proposes the output of a list of requirements and designs for future work in the investigation of Code Puzzle interactions and their insight into the understanding of NPs.

8.2.1 Proposal of a Diagnostic Tool Kit

This research has demonstrated that the use of code puzzles yields reportedly high readings of accuracy, indicating that if a tutor were to use a series of paper-based code puzzles with a CS student, they will likely be able to obtain a useful set of information that can diagnose the CS student's understanding of the underpinning programming concepts of the task in the puzzle. A tutor will discuss groupings with participants and use that as a way to detect misunderstandings, or to highlight that a grouping is related to a sensible programming concept.

In light of this, it is important to firstly consider the difficulty of the topic(s) incorporated and examined in the code puzzle as not all lines of code bear the same level of difficulty. For example, in the tertiary study participants were shown to spend less time on pieces that were associated to either field initialisation, return statements and punctuation and structure of the program in comparison to method signatures and branching pieces. While CP1 and CP2 did not use pieces that relate to all forms of topics in Java, they did yield an insight into the potential difficulty slope in terms of code. First, the tutor needs to consider the amount of information and resources that are available to the NP prior to using the diagnostic tool kit – while it may be useful to use pieces associated to concepts that the NP has yet to interact with, the use of such code puzzles will likely not be as useful as a code puzzle that is gauged more at their level of understanding (see Figure 192).

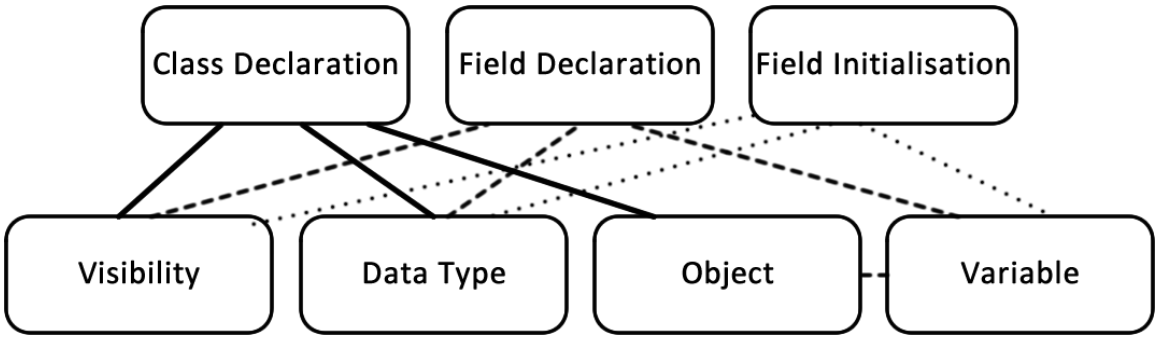


Figure 192: Low difficulty programming concepts linked to parts of a program that participants were observed to quickly complete

The quantity of Code Puzzle pieces needs to also be considered; too many puzzle pieces and options overwhelms the NP – as seen in the secondary study, where one participant was so overwhelmed by

the number of pieces available in front of them that they didn't know how to organise those pieces and did not complete CP2 due to this. Similarly, too little pieces would focus on a small amount of concepts out of context of a realistic problem – that said, if an NP is struggling to pinpoint precisely what part of, say, Java they fail to comprehend then the results of the studies infer that starting with fewer code puzzle pieces orientated around topics they are likely familiar with can help to show precisely which concepts in a smaller puzzle they are struggling with in less time than it would take to complete a puzzle with more pieces. The time taken to complete each puzzle, aside from the tertiary study, showed that the number of pieces available positively correlates to the number of pieces available to the participant. We believe the reason that the tertiary study went against these findings of the previous studies is because the screen was more overwhelmed by longer pieces than shorter pieces and therefore participants found it easy to engage with the smaller pieces. Traditional 2D Parson's problems, in full lines of code, were inferred to be easily to construct by the time taken, and also, by the theory of cognitive load – the participants did not need to think about how to correctly structure a line of code, as the code was already complete, whereas the smaller pieces in CP2 meant that one participant asked to look at their solution for CP1 in order to replicate the structure of the pre-constructed pieces for CP1. Therefore, longer code extracts in puzzle pieces – i.e., CP1 – may be easier for NPs that can trace and read programming, a level that is found to be easier than writing according to Ihantola and Karvita (2011). However, some participants in our study reported that the CP1 style of code puzzle was too impersonal and that they spent more time trying to work out the author's original intention for the pieces than actually the placement of them – if NPs exhibit this trait, it may indicate that they are more confident in their ability to write code and can see faults in the pre-written code or naming of variables used in the puzzle pieces.

8.2.1.1 Factor of Difficulty

The discrepancy in reported difficulty between the pilot and tertiary studies suggests that the difficulty is related to the cognition and experience of individual NPs. However, there is a pattern in what makes a puzzle more difficult than another for NPs – for example, all NPs that commented on the Java library's Date object suggested that this would be the hardest part of the task of CP2 as this is an unknown factor to them. The exact reasoning for an unknown factor differs between participants, for some it was that they were not confident with Java documentation but for others it was that they had not specifically used the Date object before so did not feel confident in working out from the Java documentation alone as to how the Java library works. This suggests that the puzzle's task is paramount to the perception of difficulty and should be carefully crafted based on the experience of the NPs – while the NPs had learned Java documentation in the Java Foundations

module and should have been able to interpret the method's text, it is clear that NPs thrive on being able to see how the object works and experience it for themselves. Therefore, an IDE is an integral part of the NP's formulation of the mental representations of programming constructs, and unless the NP is confident in the way that the compiler (and Java library text) works, they cannot formulate or predict with complete confident how the object will react to situations. The detection of the fact that NPs were hesitant about the Java Date library suggests that the task itself could pick this up, if the observer asked them about how confident they felt about the task. Interestingly, participants in the tertiary studies often did not explain why they felt that the part of the task would be difficult, and one participant even suggested that the whole process of "making it work" suggests that the criteria they use to assess the task difficulty is based on how confident they feel about the concepts or classes used in the task. Ultimately, familiarity with the objects used is paramount to creating an easier puzzle. Surprisingly, most participants were not put off by the way that either task was worded – despite the researcher purposely avoiding using the words 'constructor' and instead describing the purpose behind the method. Unfortunately, due to lack of time and willing participants, there was no possibility to create a control and experimental group where the task was worded differently. The reason the task was worded differently was due to the researcher not wishing to lead the participants by influencing the language that they used to describe the pieces – however the purposeful lack of technical terminology implicitly tested whether the participants could decode the concepts that they had learned in Java with the description of what those concepts are. Arguably, for some participants, the wording did seem to confuse them – and thus they were often highlighting the words This investigation found evidence to suggest that some NPs use the task description as a form of scaffolding to guide their approach with a couple of NPs following the task description like a step-by-step set of instructions, which is in-line with some of the findings purported by Fabic et al. (2019). Therefore, the task description could increase the difficulty of the puzzles by becoming less explicit about what the class requires – some NPs were observed to follow the task description word-for-word, referred to as a 'linear' approach in early research, and others were using the class to scaffold how they interpret the task – for example, thinking about the overall design of the class and often working in a 'nested' approach where they worked on the requires of each individual method. However, the linear vs nested argument was weak in hindsight as the way the task description was worded was too obvious to allow for design decisions to become clearer. After all, most participants did start from the top of the class and work their way down to the bottom of the class with a few step deviations in between. Therefore, difficulty may be increased by only suggesting what the end goal of the class is, without explaining variables, or what methods are required, and seeing whether the NPs do manage to design a coherent class. In the tertiary study it was shown that this advice

should be used with caution as the NPs often derailed and did not have solid understandings of the underpinning logic of the class – however, if the NPs are struggling with designing the class rather than creating the syntactical structure of a class, the ease of use of custom pieces and a vaguer task description would allow for the assessment of how well they can design a class based on the principle of allowing them more freedom. Yet, if the analysis of the NPs movements is to be automatised, this would not be as easy to automate as if the participants had less freedom to choose customisable pieces unless there is a smart compiler built into the design.

Regarding Parson's Problems themselves, evidence from the pilot and secondary studies shows that the extra scaffolding of the pieces allows for participants who do not necessarily know the structure of a line of code to place pieces in the correct area. However, in CP2, it was shown that some of the participants who had an almost perfect CP1 answer did not manage to get a perfect CP2 answer – with the data types of parameters often missing. If we take this example, we can analyse what the potential meanings behind an absence of the data types of the parameters means. In Python, data types do not need to be stated explicitly in parameters – therefore, questioning of the NP needs to take place in order to determine whether data types of parameters are a syntactical issue, or an issue with the concept. After all, the absence of data types in the entire class would be a stronger indicator of poor or no understanding of data types – but if the data types are only missing in certain sections, or the NP forgets the odd one but manages to correctly place them in other similar contexts, then the chances of them misunderstanding the concept of data types significantly lowers. The repetition of mistakes in every instance of a programming concept is a strong indicator of no or poor understanding, the repetition of mistakes in a specific context suggests that there is a poorer understanding with syntactical structure.

The greater number of concepts in a task, the more likelihood that the task is consequently more difficult – however, based on the research findings, this cannot be concluded conclusively because there was no deviation in the type of task presented to the participants. However, there is some anecdotal evidence that would implicate this to be true from the findings of the tertiary study as one participant, who chose to split strings for the data object, did seem to become overwhelmed and needed a significant amount of time to think about how to split the date object whereas other participants who chose to use a simply greater than operator to compare two integers did not struggle as much. The aspect of time – how long it takes to complete a puzzle – does not correlate strongly to the quality of the produced final solution. Sometimes, the participants who took longer to place each puzzle did produce better solutions than those who took a shorter time, but sometimes, this was reversed. Therefore, there is likely a subtle differentiation between participants who took longer because they were carefully contemplating each movement, and participants who took longer

because they performed more moves. In the secondary study, for CP2, participants who performed less shifting and grouping movements did produce better quality solutions than those who were continually shifting their pieces. Shifting is also a sign that a participant is finding a particular part of the puzzle difficult – this could be because the participant feels that the puzzle is 'off' and shifting upwards of pieces indicates that participants were needing to check each piece to see if it makes sense to them. The number of movements taken per piece for the pilot study was a strong indicator of which pieces were causing issues for participants, and this is a more reliable metric than the length of time the participant takes to complete the puzzle. Grouping appeared more commonly when there were more pieces – which indicates that grouping movements should be seen as a sign of the way the participant organises the meaning behind pieces based on the groups, they place the pieces in. Grouping was not strongly correlated to better or worse final solutions, but it did show what participants felt the meaning behind the pieces were prior to placing them into the solution (see Table 25).

Strength of Indicator	**Potential indicator(s) observed over the three studies**
Strong indicator of poor understanding: 'there is a strong likelihood that this NP has an issue with this programming concept'	- High number of mistakes performed on each piece relating to same concept or part of the program (e.g., multiple 'remove' actions, multiple 'back' actions)
Weak indicator of poor understanding: 'there is a weak likelihood that this NP has an issue with this programming concept'	- High number of missing pieces in final solution relating to the same concept (e.g., if all data types of were missing in the entire class, then the student may have forgotten that data types are needed OR may not understand them, especially if a specific data type – say integer – is missing but other data types are used) - High number of 'remove' and/or 'decide' actions performed regardless of if they are done to pieces related to the same concept. - NP falls silent when placing a piece of a different concept to previous pieces where they were talkative (caution: this may indicate a lack of confidence in explaining the piece, or, it may be that the piece is so obvious to the NP that they do not feel the need to explain it)

	- NP identifies incorrect concept(s) related to the piece using the think-aloud procedure - NP uses the incorrect technical terminology related to the piece using the think-aloud procedure.
Neutral Indicator: 'there are instances where this may or may not indicate issues in the NP'	- NP takes a long time to complete the puzzle (e.g., if they are making multiple back-and-forth decisions then there is a chance that they are taking longer than someone who is finding it easier to construct a solution, however some NPs who produced good solutions 'tested' and took their time when creating a solution so length of time to submit is not a strong indicator either way). - An NP's level of confidence in their ability as a programmer (the studies seemed to have fairly confident programmers, all of whom produced drastically different solutions to the same issue) - Order the pieces have been placed - Number of times the NP pauses or falls silent (the NP could be thinking carefully, or they may be overwhelmed – it is difficult for an observer to tell without context) - An NP's type of movement performed (except for 'remove', 'back' – 'swap' was arguably not observed enough to decide upon, but in instances observed in pilot and secondary study 'swap' may just indicate that the NP has found a piece more suited to the context of the task that they had not seen before, however, multiple swapping of the same pieces back and forth may indicate issues between the concepts behind the two pieces – particularly if they are not related conceptually or contextually for there to be confusion) - Speed at which individual pieces are placed in the final solution area related to the same concept (e.g., field-related pieces were placed quickly, on average, across all three studies – however decision-based pieces such as 'if' were placed slowly and this was not seemingly due to a lack of understanding but because more thought is required to place the piece)

Weak Indicator of sound understanding: 'there is a weak likelihood that this NP has no observable issues with this programming concept'	- NP identifies correct concept(s) related to the piece using the think-aloud procedure - NP uses the correct technical terminology using the think-aloud procedure. - NP uses little to no 'remove' or 'back' actions.
Strong indicator of sound understanding: 'there is a strong likelihood that this NP has no observable issues with this programming concept'	- The pieces related to a concept, when submitted in the final solution, are in the correct place (bar one or two 'mistakes' that could be easily forgotten such as braces)

Table 25: Indicators demonstrated in the results of the pilot, secondary and tertiary studies which imply lack of understanding or show understanding compared to understanding of their approach to programming.

8.1.3 Proposal of a Cluster-based Puzzle

In light of the novel concept of the workspace discovered during the pilot and secondary studies, a cluster-based puzzle where NPs are asked to group pieces together based on a programming concept could be a potential avenue for future work. Cluster-based Code Puzzles could identify how participants 'identify', or understand, the concepts behind puzzles by demonstrating to the observer what pieces fit into what categories and how the NP relates those categories together. For example, an NP would need to know about Java naming conventions to correctly distinguish between variable names and object names in Java and could demonstrate this knowledge by placing the relevant pieces into two separate groups. To enhance difficulty, there could also be variable names such as 'double' to see whether participants realise that naming a variable after an object is not good Java naming conventions.

Cluster-based puzzles would also be easier to analyse using K-Means or another clustering algorithm (e.g., GMM) than Parson's Problems as the way a student groups pieces on the screen can be analysed by proximity of pieces – or can include the recommendation of allowing the NPs to 'label' components either by putting them into area labelled the programming concept's name, or by allowing them to label individual pieces. On the basis of simplicity and reducing the mental load the NPs have while interacting with cluster-based puzzles, it would be recommended to test them placing the pieces under the 'correct' programming concept name. The factor of difficulty could be

increased by allowing them to generate the programming concept names themselves, or by allowing programming concepts that are typically confused (e.g., primitive data types and data types) or pieces that do not belong to either group to make the NP consciously evaluate which piece belongs to what group and not allow the pieces to guide them as shown in the pilot and secondary studies.

Cluster-based puzzles in research regarding NPs has been limited based on the findings of a systematic search – Hammoudeh (2016) proposed a fully automated solver for multiple square jigsaw puzzles, which indicates that machine learnings are adapted to the coding equivalent of a jigsaw puzzle. However, cluster-based puzzles will ask the NPs to put the puzzle pieces in the correct area rather than in the order of the way they would be put into a class. The difficulty factor would also be increased by the number of pieces available to the NP, and the number of groups available on-screen. In support of the findings by Fabic et al. (2019) which demonstrated that Parson's 2D problems would work well on a mobile device using their tool PyKinetic if there is one area rather than the final solution and randomised solution spaces, the cluster-based puzzle would also use a single area but have the groups divided up (see Figure 193).

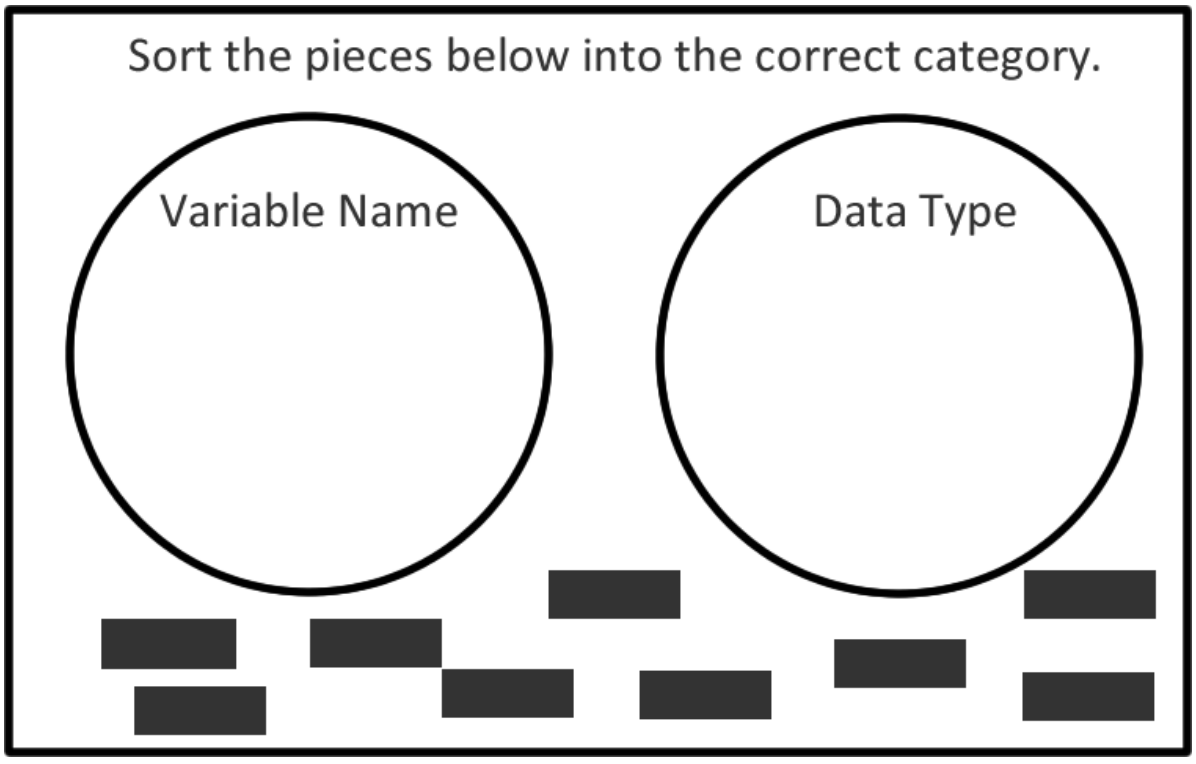


Figure 193: Proposed cluster-based code puzzle in light of the novel finding of the workspace

8.3 Chapter 8 Summary

This chapter presents and discusses the collective findings of the three research studies, alongside highlighting the novel outputs that have been generated as a result of the research undertaken.

Chapter 9: Limitations, Future Work and Concluding Thoughts

This research aimed to discern whether the level of understanding of NPs can be determined through examination of their interactions with Code Puzzles. This thesis has presented the findings of three interpretivist mixed-methods studies that indicate the use of Code Puzzles can reveal the NP's understanding of programming and, from the analysis of the quantitative and qualitative data obtained from these studies, made proposals towards the design of a diagnostic toolkit, based on Code Puzzles that can be used by teachers to gain insight into the levels of understanding of their students.

The results indicate that Code Puzzles are an accurate way of determining understanding of NPs. The symptoms of understanding (i.e., correct terminology used for the selected puzzle piece, confident tone when speaking about the concept, quick labelling of the selected puzzle piece, and correct identification of the place the piece would fit) are detectable in NP interactions with Code Puzzles. Similarly, the symptoms of misunderstanding (i.e., incorrect terminology for the piece, incorrect placement of the piece, and incorrect labelling of the selected puzzle piece) are detectable in NP interactions with Code Puzzles. However, the time taken to complete the puzzle and the length of time NPs spoke does not clearly correlate to their level of understanding. Nonetheless, participants agreed that the accuracy of the expert's analysis of their understanding was high – with a couple of participants even going so far as to suggest that they had learned more about themselves from completing the Code Puzzles. This account of high accuracy of diagnosis of understanding from the participants of the second and tertiary studies could be attributed to authority bias as participants may view the 'expert' as superior in nature to a novice, and therefore attribute more weight to their analysis than is the actuality – however, prior to the studies, the NPs completed background questionnaires which logged their approach and thoughts about programming which correlated to their movements during the Code Puzzles themselves, and, NPs also suggested that the weaknesses highlighted by the expert were genuine weaknesses in their dialogue with the expert during the feedback sessions in the tertiary study. One participant suggested that the puzzles were not difficult enough to capture their perceived issues, and this suggests that for more able students, a graded level of difficulty and style of Code Puzzles is required.

The effectiveness of using Code Puzzles in a realistic classroom-based environment was not possible to conduct, however, based on the 40-to-60-minute length of time required to allow participants sufficient time to complete the puzzles, it is likely that paper-based Code Puzzles are not scalable to large numbers of students in the format that we have used them. One way to resolve the issue of scalability is to propose a software design that can be used by NPs in the classroom to help automate

the process of collecting the data obtained from Code Puzzles; this has proven to be a possibility, as multiple researchers have used software versions of 2D Parson's Problems to investigate cursor clicks (e.g., Helminen et al., 2012) and Code Puzzles have become a popular education tool for teaching programming in recent years (Ito et al., 2020). While unrealistic to obtain the requirements, design, implement, test and deploy a system in the scope of one thesis, this thesis proposes an investigation into how the findings can implicate best practice guidelines (i.e., incorporation of the workspace, ways for students to record their thinking/thought processes about pieces – probably through text, but if natural language processing could be used this may help to partially automate the process and see whether students were referring to the correct terminology when discussing a piece). In our findings, Code Puzzles were found to be an effective medium, or discussion point, to demonstrate understanding of NPs and, consequently, future work could be conducted into using paper-based Code Puzzles to examine the interactions between pair programmers and/or in a collaborative learning environment to investigate whether NP understanding can be more efficiently obtained.

The NP's dialogue accompanying the Code Puzzle movements was found to be important for determining understanding, thus implying that positivist studies investigating Code Puzzles that use pure quantitative data – such as, examining the number of cursor clicks, time taken to place pieces or location of piece placements – may not obtain a full picture of an NP's reasoning behind their movements. NPs' approaches were found to be similar – with participants typically starting from Line 0 and completing the class in chronological line order, thus examining the order in which pieces are placed for 2D Parson's Problems was rated low in terms of understanding the NP's understanding. Likewise, it is difficult from cursor clicks alone, to make inferences about the reasoning behind the NP's movements. We therefore hope to have convinced the reader to realise that it is important to get the NP to verbalise or communicate their thinking for us to fully understand their reasoning or conceptions about the puzzle pieces. For that reason, full automation is a non-trivial project but given the advances in machine learning and text processing, may well be possible. In the meantime, our toolkit design could be used with individual students to diagnose their level of understanding.

This thesis further argues that NPs are demonstrated to be a diverse group as there were a variety of final solutions submitted despite the relatively small sample size. Future work could determine whether similar findings are obtained while varying the type of NP recruited, for example, perhaps specify that participants share the same: starting programming language, set of programming languages, programming paradigm, age, gender, personality, and/or years of experience and vary the Code Puzzles to be in a series of languages that were not necessarily their starting language. These studies could also be repeated using EPs instead of NPs, to see whether the type, form and frequency of communication differs and whether the level of accuracy is also as reportedly high as NPs. Ideally,

these studies would have exerted more control over the sample by incorporating Code Puzzles as an optional part of the syllabus of a module for first-year undergraduate students, however, this was not deemed ethical and therefore the baseline of the studies presumed that all first year CS undergraduates at the same university would have been exposed to the same level of Java. That said, even if the researcher had been allowed to integrate Code Puzzles as optional material it is difficult to control the level of experience or programming exposure that NPs have prior to university. Likewise, studies that could allow for the usage of examination results and data to help compare Code Puzzle performance with written examination performance could be insightful.

Our evidence shows that with modification to their design, Code Puzzles are a useful diagnostic tool for identifying the understanding, misunderstanding and lack of understanding in NPs and that observing NP interactions and listening to NP reasoning illuminates NP's perceptions and assumptions regarding programming. The findings were in line with the study conducted by Ito et al. (2020) and support the notion that code puzzles were a useful tool for diagnosing understanding and rejects the notion suggested by Helminen et al. (2012) that the analysis of movements of 2D Parson's Problems are not sufficient to determine the understanding of the NPs.

9.1 Reflection on Research Aims

To analyse the success of the research, we must consider the aims (see Table 26).

Aim ID	Aim Description
A-1	Discern an approach to identify and represent an NP's level of understanding of programming concepts and the computational thinking strategies they used.
A-2	Evaluate the accuracy of the level of understanding of an NP by comparing the observer's interpretation of the level of understanding to the perceptions of the NP's understanding of their understanding.
A-3	Determine the best practice for representing the level of understanding.
A-4	Compare whether learners of a similar level of understanding share characteristics in the way they interact with Code Puzzles (learner interactions).
A-5	Discover whether a learner's conceptions and misconceptions about a programming concept can be identified purely on their learner interactions.
A-6	Discover whether a learner's level of understanding about a programming concept can be identified purely on their learner interactions.
A-7	Determine whether a learner's perception of their own computational thinking correlates to their actions and thought processes while interacting with Code Puzzle pieces.
A-8	Determine if there is any correlation between the types of interactions performed and the NPs' level of understanding.

Table 26: Evaluating the Research Aims

"Discern[ing] an approach to identify and represent an NP's level of understanding of programming concepts and the computational thinking strategies they used (A-1)" was partially achieved through identifying the characteristics in the NP's interactions to solving the Code Puzzles – although the

representation aspect of the level of understanding has been translated to the difficulties encountered by NPs through the proposal of the advice in the diagnostic tool kit. Quantifying understanding was shown to be problematic, partly due to the nature of paper-based Code Puzzles but also because 'incorrect' submissions or movements did not correlate directly to a lack of understanding, therefore, translating the movements of Code Puzzle pieces to increase or decrease an arbitrary scale was not deemed to be an appropriate representation of the level of understanding exhibited. Instead, listening to the NP's dialogue on how they interpret a piece and whether that type of piece has been used correctly elsewhere in the puzzle (or whether it is a repeated issue) are greater indicators of the level of understanding of the NP. NP dialogue has not been obtained in past research investigating whether understanding can be diagnosed from the cursor movements of 2D Parson's Problems (e.g., Helminen et al., 2012; and Ito et. al, 2020) and this approach of garnering primarily quantitative data associated to the Code Puzzle piece's movements is not enough to accurately gauge the level of understanding of NPs. Likewise, the focus on past research to determine whether pieces were swapped, removed, or added was also shown to be not a perfect indicator as to their understanding from our findings – swapping pieces did not indicate that there was an issue in the underpinning concepts of those pieces, more so, it indicated that the participant either confused the pieces or that the pieces were interchangeable based on their perceived context that the piece should be used in. Removing pieces did show that there is a likelihood that at least one of the underpinning concepts behind the piece may have been misunderstood, but there were other factors such as misunderstanding the problem context and using, say, the wrong variable name for a specific method. The issue discovered with using 2D Parson's Problems to identify and represent understanding is that there is also the understanding of the problem's context – as the process of 2D Parson's problems is linked to the construction of a working piece of code. While problem context makes 2D Parson's problems an authentic task, it means that getting the raw understanding of the participant's applied knowledge of programming constructs is lost to the noise of the participant's understanding of what the problem is asking them to do. We discovered that it is important that the task description does not lead the participant into mimicking the terminology in the text, and that it is also not so vaguely written that the task description becomes unclear. Therefore, the representation of understanding obtained from participants using Code Puzzles is not inherently perfect but does indicate whether the participants can construct a solution and talk through their process and interpretation of individual segments of code to a tutor – which, if the expert listens out for inconsistencies and looks for wrongly applied pieces of code, is a better indicator as to the NP's understanding than if the observer was entirely removed or fully automated. Realistically, tutors do not have the student to tutor ratio to be able to sit with NPs individually and diagnose their

understanding, however, if this tool was used by a programming support officer to diagnose a CS student who was struggling to explain what their issues were with programming it can be indicative of their understanding and troubles to a relatively high degree of accuracy.

"Evaluat[ing] the accuracy of the level of understanding of an NP by comparing the observer's interpretation of the level of understanding to the perceptions of the NP's understanding of their understanding (A-2)" was achieved through methodology change of the secondary and tertiary studies – where the observer gave recorded feedback to the participants either at the end of the study (for secondary studies) or after the end of each puzzle (for tertiary studies) which, in turn, the participants acknowledged whether they felt that the analysis was accurate or not. While it is true that participants may have exhibited authority bias, as the observer was advertised as an expert in the field of CS, these accuracy readings were taken from the post-study questionnaire where it was emphasised that the participant should be honest about the dialogue they had with the observer and whether they did believe it was a correct analysis. Participants did not artificially agree that using Code Puzzles would be a good replacement for revision aids, likewise, some participants did ask for the expert to explain concepts to them that they clarified they did not understand suggesting that watching how NPs interact and discuss Code Puzzle pieces are effective ways to gaining an insight into the NP's understanding of programming and the problem context. The accuracy readings were in line with the findings of Ito et. al (2020), who used surveys alongside the Code Puzzle cursor clicks to clarify whether the findings of Random Forest Tree Selector analysis of cursor clicks match participants' understanding of programming constructs.

"Determin[ing] the best practice for representing the level of understanding (A-3)" was achieved through the recommendation of a series of guidelines for the diagnostic toolkit and the proposal of using a cluster-based puzzle template to better determine how participants relate programming concepts together, as, the 2D Parson's Problems focused the participants' dialogue more so on the approach to creating a solution rather than the participants' thoughts about individual Code Puzzle pieces.

The aim of "compar[ing] whether learners of a similar level of understanding share characteristics in the way they interact with Code Puzzles (learner interactions). (A-4)" was achieved using Straussian Grounded Theory to generate open codes for the participant's dialogue, as, by grouping the dialogue into axial codes we were able to discover the general themes that were being produced between participants. This aim was also achieved through comparing the types of movements between participants in the movement transcripts to notice similarities, and this research did observe that the general process of constructing a class was similar between participants. The terminology that

participants used when talking about the pieces demonstrated similarities, and participants that struggled with the same type of piece showed this through their movements and words.

A-1 was achieved through identifying characteristics in the NPs' approaches to solving the Code Puzzles, and through an appreciation of all of their characteristics combined, a representation of their understanding of program concepts and computational thinking patterns could be achieved. A-2 and A-5 was also achieved, as the accuracy of the level of understanding was purportedly high. A-3, however, was ambiguous in retrospect – due to the limitations of the number of participants we collected for each study we could not feasibly alter the study's methodology to include variations such as: programming language, puzzle type, puzzle task or puzzle content. Therefore the 'best' practice cannot be proven either way, this thesis only present data obtained from using paper-based 2D Parson's problems in Java and only recruited participants from in their first year undergraduate CS degree course. A-4 was also difficult to evaluate, as, it is true that participants did share commonalities and characteristics – such as, for example, approaching the problem domain from a structural programming perspective but unless we could perform a baseline examination on them it is difficult to tell if, truly, they have the same perceptions and understandings of program concepts. For A-4, with this in mind, if we take the idea that the majority of participants felt that the analysis of their understanding was correct, we can determine that, based from the participants' self-evaluation, they at least support that A-4 is achieved as we did note participants making similar movements with purportedly similar answers in their questionnaires after the puzzles. But, with a critical eye, participants may have the same perceptions of the task but not necessarily the same perceptions of the programming concepts themselves. A-6 is debatable; it is deemed that the ideal method of extracting the level of understanding is through a mixture of both verbal and visual feedback to the observer – therefore, this research argues that while you can get some form of representation about obvious movements, such as grouping pieces together, it is unlikely you would receive as accurate a representation as if you spoke to the NP. A-7 was somewhat supported by the data received from the Code Puzzle studies – participants did document in the post-study survey that they felt the approach had been accurately determined, that said, when comparing the approaches, they took to the information they provided in the background questionnaire it became apparent that the approaches did differ – primarily because Code Puzzle pieces are different to writing a program from scratch. Aspects such as testing, or, writing their own variable names were reported by participants as differing from the way they would construct a program if given the task naturally, therefore, it is deemed by the researcher that participants felt it was accurate to the way they approach problems but is not realistic to the way they code on a development environment. A-8 is the aim that was proven untrue by the research; essentially the studies did determine that there could be general

correlations to specific movements – for example, an NP removing a piece from the Code Puzzle's final solution space usually indicates that they have made a mistake and may be a little less sure about related program concepts – but, that these movements without context that the piece was placed in were meaningless with the exception of grouping where the movements were distinct enough that pieces that were closely clustered together could be artificially claimed to have similarity. Overall, the aims of the research seem to have been met, apart from A-8.

9.2 Future Research

Code Puzzles are growing in popularity according to Harms et al. (2015) and education-based gamification is a growing field of research according to Almadia et al. (2021) so there are plenty of potential avenues for future research. To achieve data saturation when using Straussian Grounded Theory with just 21 participants in total is difficult, so a simple future avenue is to repeat the experiment with more participants to see if the results can be replicated or whether new information can be gathered.

9.2.1 Task Description Modifications

In the context of this research, both tasks were written in an almost step-by-step guide – stating what the class was, what the fields were, what the constructor was in that order – and it would be interesting to see how jumbling the order of the task information may affect NPs. There is also a possibility of changing the way the task is presented to an NP – for example, NPs could be given a higher-level task description, describing what the class needs to achieve inside a system, or maybe even a UML diagram and told to implement the functionality. Some participants throughout the course of the three studies suggested the way that the variables or methods were named was not intuitive to them, so giving participants more flexibility to construct custom pieces could be a potential route to take and to see whether just examining the custom pieces alone is enough to determine their understanding of programming or whether more information is needed from them. A participant in the tertiary study also suggested that the task was not authentic, and that they would need access to the internet and their own development environment to properly construct a solution – it would be ideal to conduct a study to provide a more solid baseline on how NPs would construct a solution on a computer to the given tasks as the 'model solution' was written by an 'expert' so may not be what an NP would intuitively write.

If there were enough participants, the wording of the tasks could be altered so that there are different versions of the same task to see which type of structure enables participants to complete the puzzle more quickly. For example, the tasks in this research purposefully avoided using the

technical terminology such as 'constructor', but this was suggested to confuse certain participants who did not understand that a constructor was 'a method that created an object' – it would therefore be interesting to see how the wording affects the participants but also the quality of the observer's assumptions of what the NP understands. It was unclear, for example, whether the observer would have obtained the same information if the participants had not audibly read and deduced what the task description meant.

If the task description was modified in such a way that it asked participants to initially choose the variable, class and method names that they think related to the task based on the task description this would allow for the observer to see how participants determine which pieces are relevant and which ones are irrelevant to the task based on their interpretation of the task's description. If pieces that were alternatives for method names and variable names were arranged then participants could choose the names they prefer before the observer removes the surplus names as it was found, particularly with paper-based, that distractors meant that participants were spending more time trying to find a specific piece because there were so many pieces laid out on the table – which wasn't an indicator for them not understanding something, it was simply because they were unable to find the piece they were looking for.

Another alternative for modified the task description would be to give a very basic sentence of what the class needs to do (i.e., "a potato shop class needs to have the functionality to sell potatoes for £1.00") and see if the NP can write a specification for what they think the class should have as components. This modification, though, would need to be accompanied by either software that allows them to generate custom code segments, or small whiteboard-based cards that the NP can write on when they need to create a card, it would also take longer – it took approximately 50-60 minutes per participant with two tasks with pre-written cards.

9.2.2 Using Cluster-based Puzzles and the Potential for QUI

As suggested by this research, there appears to be value in examining how NPs group code segments as it demonstrates how they relate pieces to programming concepts with more clarity than simply observing how they place the pieces into functioning lines of code. Instead of using 2D Parson's Problems, it would be interesting to see how participants would engage with cluster-based puzzles where the task is modified to specify that the participant needs to group the pieces into two or more groups. The task could be modified so that there may be groups within groups allowed; for example, maybe the task initially says to separate data types from names, but there might be two different types of names – normal and final variable names – and different types of data types – such as numerical and alphabetical and/or primitive data types. The task could also be modified to

intentionally make groups that can overlap; for example, if the task asked participants to place which return statements would be potentially valid to use for two different method signatures into groups there might be a return statement that could be potentially applicable to both methods that could go between the groups.

One of the issues with cluster-based analysis is determining when a group ends and starts, and whether this process could be automated. It would be interesting to see whether observers analysing the same set of groups created by participants would get the same results and interpretations, and this would be necessary for a more formalise procedure on how to interpret grouping of puzzle pieces to be generated. Similarly, attempting to use K-means and GMM clustering algorithms on clearer photo samples of grouping can help to clarify if automation of cluster-based puzzle analysis is possible as the sample size from this research is too small to analyse in that way.

It may be possible that the QUI metric could be refined and more easily incorporated with the cluster-based versions of the puzzles. The QUI metric was a work in progress at the time of thesis submission – it essentially wanted to extrapolate the information obtained from the movement log and quantify the likelihood that a participant had an issue with a particular concept. QUI was designed to use a sliding scale for each programming concept, and each relevant piece to that programming concept would tip the scale based on user interactions. For every decide, remove, back, incorrect missing piece in the final solution and incorrect add placement the scale would decrease to show likelihood of an issue with the concepts related to the piece increasing. For every correct add, the scale would shift upwards to indicate that there is a positive likelihood that the student does understand the concepts related to the piece (see Figure 194).

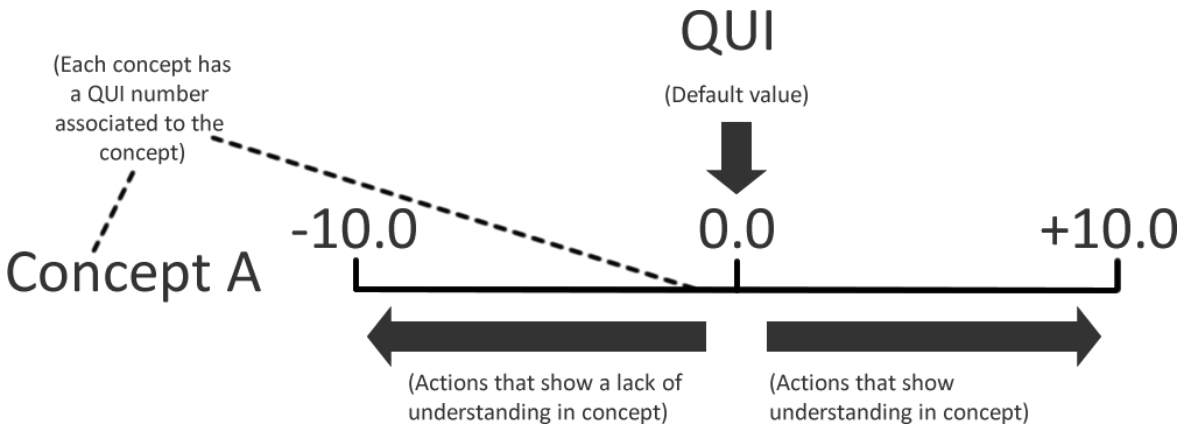


Figure 194: The structure of the Quantitative Understanding Indicator (QUI) for each programming concept.

The scale would operate on a novel concept of 'degree of certainty' as the system could never rule out, as per the findings of all studies, the intention of the movement even if it is incorrect. The length of time to correct a mistake would help to attribute a weighting to the mistake – as, if there are repeated mistakes that are rectified prior to solution submission, it could indicate that the participant forgets such issues and may want to become aware of this. Consequently, from the findings of the research the QUI metric was produced but unfortunately not tested.

There are disadvantages to the QUI even if it had been successfully implemented in time, and these issues may arise within the future work potential of cluster-based puzzle analysis so are worth discussing here. The problem with QUI is that it assumes a lot based purely on movements which, as Helminen et al. (2012) and the findings of this study suggest, is a weakness. QUI does not consider the context of how the piece is placed – for example, if a participant did not understand what a field name represented or what a method was supposed to do it is possible that they could construct something completely incorrect that QUI would assume is down to lack of understanding of the associated programming concepts. As a result, the reasoning and context behind why a piece is placed in the position is crucial to understanding what issues the participant may have. While QUI was designed to consider that participants may make mistakes, and that the penalty for applying multiple pieces related to the same concept incorrectly would scale to try to reduce the damage of mistakes, this is not enough to be able to say with certainty that a quantitative metric can be applied to assess understanding effectively. It is possible that QUI could work if contextual information was considered, but how do you potentially account for all of the variations in interpretations of each element of the task? Perhaps this is feasible for very small sections of code, but not for something as big as a Java class. One of the issues with determining context, especially for CP2, was at what point would a section start and end – for example, when a participant declares fields in a class, they may jump to initialising the fields despite note creating the constructor's signature yet – in this instance, at what point does the automated metric know that the participant does understand the difference between initialising and declaring a field? The answer is when the constructor's signature is put in the right place, but how can the metric tell the difference between a participant that has forgotten what the constructor signature is (or forgotten that it is required) and a participant that has just found pieces related to a field and wants to place all those pieces in the final solution so that they know what to write in the constructor's parameters? The answer to this could be that the metric considers the amount of spacing between lines, but without the participants having a grid it may be difficult to achieve this. Even in the studies shown in this thesis, participants had different indentation and spacing styles – it would be interesting, though, to see whether QUI could be incorporated with any degree of success if participants were given line numbers separated by lines

and that a short task was produced which had both programming concept weightings to pieces and contextual-based information.

The formulae for QUI were envisioned to be influenced by the type of movements discovered in the three studies: as remove, decide and back were stronger indicators than add and swap that there may be uncertainties about the piece from the NP's perspective.

Overall, there is a potential to explore the automation avenue, but this thesis argues that what the NP is saying is more important than the specific moves.

9.2.3 Using Other Languages, Levels of Programmer and/or Parson's Problem GUI Implications

The suggestion that more information may be obtained from a less restrictive version of the Parson's Problems GUI needs to be investigated to substantiate that claim; having a Parson's problems software interface with just one area rather than the traditional two areas may allow us to see whether this is helpful for the electronic versions of Parson's problems or whether this only occurs in paper-based versions of the puzzle.

There is also a future avenue for investigating whether the grouping movement type is observed in more experienced programmers, or whether using a higher-level language than Java (like Python, which has more linguistical brevity than Java) would yield similar results.

9.3 Conclusion and Final Thoughts

This research hopes to contribute valuable ideas to researchers and practitioners in the field of CS Education on the aspect of NP phenomenology. It is hoped that practitioners can find use in the diagnostic toolkit and cluster-based puzzle design to obtain an accurate analysis of the understanding of NP views and experiences, and that the concept of the workspace can shape future research into the way NPs relate and link programming concepts and ideas together.

Chapter 10: Bibliography and References

10.1 List of Bibliography

Allen, I.E., and Seaman, J. (2013). *Changing course: ten years of tracking online education in the United States*. Sloan consortium. [Available from: http://eric.ed.gov/?id=ED541571] [Last Accessed: 2nd July 2020]

Almedia, C.; Kalinowski, M.; and Bruno, F. (2021). *A Systematic Mapping of Negative Effects of Gamification in Education/Learning Systems*. Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp.17-24

Bandiera, O., Larcinese, V., and Rasul, I. (2010). *Heterogeneous Class Size Effects: New Evidence from a Panel of University Students. Economic Journal*. 120, 549, pp.1365-1398

Bettinger, E. P., and Long, B. T. (2016). *Mass Instruction or Higher Learning? The Impact of College Class Size on Student Retention and Graduation*. MIT Press Journals. Education Finance and Policy. 1-36.

Brown, E., Brailsford, T., Fischer, T., Moore, A., and Ashman, H. (2006). *Reappraising cognitive styles in adaptive web applications*. In Proceedings of the 15th international conference on World Wide Web.

Burley, H. (2011). *Cases of Institutional Research Systems*. Information Science Reference.

Chaudhary, Vi., Agrawal, V., and Sureka, A. (2016). *An Experimental Study on the Learning Outcome of Teaching Elementary Level Children using Lego Mindstorms EV3 Robotics Education Kit*. [online] Available from: < http://arxiv.org/abs/1610.09610>

Chen, P. Y., and Popovich, P. M. (2002). *Correlation: Parametric and Nonparametric Measures*. Sage University Papers Series on Quantitative Applications.

Chou, Y. (2019) *Actionable Gamification: Beyond Points*. Badges and Leaderboards

Flowers, T., Carver, C.A., and Jackson, J. (2004). *Empowering students and building confidence in NPs through Gauntlet*. 34th Annual Frontiers in Education, 2004. Piscataway, NJ, USA. 13(1).

Guzdial, M. (2009). *How we teach introductory CS is wrong*. *Blog at* Communications of the ACM. Trusted Insights for Computing's Leading Professionals, URL http://cacm.acm.org/blogs/blog-cacm/45725-how-we-teach-introductory-computer-science-is-wrong/fulltext.

Guzdial, M., and Robertson, J. (2010). *Too much programming too soon?.* Communications of the ACM, 53(3), pp.10-11.

Ivanova, G., Kozov, V. and Zlatarov, P. (2019). *Gamification in Software Engineering Education*. 2nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO).

Kinnunen, P., and Simon, B. (2012). *Phenomenography and grounded theory as research methods in computing education research field*. CS Education. Elsevier B.V. 22, 2, 199-218.

Kinnunen, P. (2006) *Why students drop out CS1 course?.* Proceedings of the second international workshop on Computer education research. ACM Digital Library.

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development*. Englewood Cliffs, N.J.: Prentice-Hall.

Kölling, M. (1999b). *The problem of teaching object-oriented programming, Part 2: Environments*. Journal of Object-Oriented Programming*, 11*(9), 6-12.

Kölling, M., and Henriksen, P. (2005). *Game programming in introductory courses with direct state manipulation*. ACM SIGCSE Bulletin. *37*(3). Pp.59-63.

Lam, H. (2017). *Using phenomenography to investigate the enacted object of learning in teaching activities: the case of teaching Chinese characters in Hong Kong preschools*. Scandinavian journal of educational research. 61(2).

Marton, F. (1981). Phenomenography: Describing conceptions of the world around us. Instructional Science. 10(1)., pp.177-200.

Marton F. (1986). *Phenomenography: A research approach to investigating different understandings of reality*. Journal of Thought. 21(3). Pp.28-49.

Marton F. (1988). *Phenomenography: Exploring different conceptions of reality*. In D. M. Fetterman, Qualitative approaches to evaluation in education: The silent science. Pp.176-205.

Nardelli, E. (2019). *Do we really need computational thinking?* Communications of the ACM.

Ortin, F.; Redondo, J. M.; and Quiroga, J. (2017). Design and evaluation of an alternative programming paradigms course.

Winslow, L. E. (1996). *Programming pedagogy – a psychological overview*. *SIGCSE Bulletin*, 28(3).

10.2 List of References

Ahmed, A., Zeshan, F., Khan, M. S., Marrian, R., Ali, Amjad, and Samreen, A. (2020). *The Impact of Gamification on Learning Outcomes of CS Majors.* ACM Transactions on Computing Education. 20(2). pp.25.

Aiken, J. (2004). *Technical and human perspective on pair programming*. ACM SIGSOFT Software Engineering Notes. 29(5). pp.1-14.

Alardawi, A. S. and Agil, A. M. (2015). *Novice Comprehension of Object-Oriented OO Programs: An Empirical Study*.

Allinson, C. and Hayes, J. (1996). *The cognitive styles index: A measure of intuition analysis for organisational research*. Journal of Management Studies. 33(1). pp.119-135.

Almujally, N., and Joy, M. (2020). *Applying a Gamification Approach to Knowledge Management in Higher Education Institutions*. IEEE 44th Annual Computers, Software and Applications Conference (COMPSAC). pp.455-459.

Alshammari, M. (2016). *Adaption Based Learning Style and Knowledge Level in E-Learning Systems*. School of Computing Science. University of Birmingham. [Online] Available From: <http://etheses.bham.ac.uk/6702/11/Alshammari16PhD.pdf> [Last Accessed: 2nd October 2016].

Aston University. (2020). *BSc CS*. [ONLINE] [Available from: https://www.aston.ac.uk/study/courses/computer-science-bsc] [Last Accessed: 31st August 2020]

AQA. (2020). *A-Level Problem Solving: Top-down design with stepwise design refinement*. [online] Available from: <https://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Top-down_design_and_Step-wise_refinement)>

Bayman, P.; Mayer, R. E.; andSchwartz, M. H. (1983). *A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements*. Communications of the ACM, September 1983) 26(9). pp.677-679.

Beaubouef, T. and Mason, J. (2005). *Why the high attrition rate for CS students*. ACM SIGCSE Bulletin. 37(2). pp. 103.

Becerra, C., Munoz, R., Noel, R., and Barria, M. (2016). *Learning objects recommendation platform based on learning styles for programming fundamentals*. XI Latin American Conference on Learning Objects and Technology (LACLO). Pp.1-6

Begel, A., and Nagappan, N. (2008). *Pair programming: What's in it for me?.* In Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement, Madrid, Spain.

Benda, K., Bruckman, A., and Guzdial, M. (2012). *When Life and Learning Do Not Fit: Challenges of Workload and Communication in Introductory*. CS Online. 12 (4), 1-38.

Bennedsen, J. and Caspersen, M. E. (2007). *Failure Rates in introductory programming*. ACM Digital Library.

Bennedsen, J., and Caspersen, M. E. (2008). *Optimists Have More Fun, but Do They Learn Better? On the Influence of Emotional and Social Factors on Learning Introductory CS*. CS Education: Routledge, Taylor and Francis, Philadelphia, PA.

Bieri, J., Atkins, A. L., Briar, S., Leaman, R. B., Miller, H., and Tripodi, T. (1966). *Clinical and Social Judgement: The Discrimination of Behavioral Information*. New York: Wiley.

Biggs, J. (1995). *Assessing for learning: Some dimensions underlying new approaches to educational assessment*. The Alberta Journal of Educational Research. 41(1). pp1-17.

Biggs, J. B. and Collis, K. F.(1982). *Evaluating the Quality of Learning*. The SOLO Taxonomy. New York: Academic Press.

Biggs, J. B. and Tang, C. (2011). *Teaching for Quality Learning at University*. (4th ed). Maidenhead: McGraw Hill Education and Open University Press. [Adapted sample from CLEAR, CUHK]

Bork, A. M. (1972). *Learning to Program for the Science Student*. [online] Available from: < https://eric.ed.gov/?id=ED060627>

Bosch, N., and D'Mello, D. (2017). *The affective experience of novice computer programmers*. International journal of artificial intelligence in computing.

Bosse, Y, and Gerosa, M.A., (2017). *Difficulties of Programming Learning from the Point of View of Students and Instructors*. IEEE Latin America Transactions IEEE Latin Am. Trans. 15(11). pp.2191-2199.

Bransford, J.D., Brown, A.L., and Cocking, R.R. (2000). *How People Learn*. National Academy Press, Washington D.C.

Bryant, S. (2004). Double trouble: *Mixing qualitative and quantitative methods in the study of extreme programmers*. In IEEE symposium on visual languages and human centric computing. Rome.

Bryant, S., Romero, P., and du Boulay, B. (2006). *The collaborative nature of pair programming*.

Carelli, O. M., M.; Serey, D. and Figueiredo, J. (2017*). Learning styles in programming education: A systematic mapping study.* 017 IEEE Frontiers in Education Conference (FIE). Pp.1-7.

Carmo, E. P. D., Klock, A. C. T., de Oliveira, E. H. T. and Gasparini, I. (2020). *A study on the impact of gamification on students' behavior and performance through learning paths*. IEEE 20th International Conference of Advanced Learning Technologies (ICALT). pp.84-86.

Century, J., Ferris, K. A., and Zuo, H. (2020). *Finding time for CS in the elementary school day: a quasi-experimental study of a transdisciplinary problem-based learning approach*. International Journal of STEM Education. 7(1).

Channel 4 News. (2017). *Which Universities Have the Highest first year dropout rates?* [ONLINE] Available from: <https://www.channel4.com/news/factcheck/which-universities-have-the-highest-first-year-dropout-rates> [Last accessed: 22/05/2021]

Chaparro, E., A., Yuksel, A., Romero, P., and Bryant, S. (2005). *Factors affecting the perceived effectiveness of pair programming in higher education*. In Proceedings of the 17th workshop of the psychology of programming interest group.

Cockburn, A., and Williams, L., (2001). *The costs and benefits of pair programming. Extreme programming examined*. pp.223-243.

Coffield, R., Moseley, D. Hall, E., Ecclestone, K. (2004). *Learning styles and pedagogy in post-16 learning: A systematic and critical review.*

Computer Weekly (2019). *CS undergraduates most likely to drop out*. [online] Available from: <https://www.computerweekly.com/>

Cropley, A. J. (1967). *Creativity: A New Kind of Intellect?*. SAGE Journals. London.

Correia, A.L., da Costa, D., Barbosa, A. and Costa, E. (2015). *Uso de avaliação por pares em disciplinas introdutórias de programação*. Workshop sobre Educação em Computação. pp.1–10.

DeClue, T. H. (2003). *Pair programming and pair trading: Effects on learning and motivation in a CS2 course. Journal of Computing Sciences in Colleges*. 49-56.

de Raadt, M. (2008). *Teaching programming strategies explicitly to NPs.* (Doctoral dissertation), University of Southern Queensland.

Denny, P., Luxton-Reilly A., and Simon B. (2008). *Evaluating a new exam question: Parsons Problems*. University of Auckland: New Zealand.

Desmedt, E. and Valke, M. (2004). *Mapping the learning styles "jungle": An overview of the literature based on citation analysis*. Educational psychology: Taylor and Francis. 24(4)

Deterding, S.; Dixon, D.; Khaled, R. and Nacke, L. (2011). *From game design elements to gamefulness: Defining "gamification"*. Proceedings of the 15th International Academic MindTrek Conference. pp.9-15.

du Boulay, B. (1986). *Some difficulties of learning to program*. Journal of Educational Computing Research*, 2*(1), pp.57-73.

Eckerdal, A., and Berglund, A. (2005). *What does it take to learn 'programming thinking'*?. ICER 2005. ACM Press. pp.135-142.

Ehlert, A. and Schlute, C. (2009). *Empirical Comparison of Objects-First and Objects-Later*. Proceedings of the 5th International Workshop ICER 2009.

Elbardan, H. and Kholeif, A. O. R. (2017). *An Interpretive Approach for Data Collection and Analysis*.

Fabic, G. V. F.; Mitrovic, A.; and Neshatian, K. (2019). *Evaluation of Parsons Problems with Menu-Based Self-Explanation Prompts in a Mobile Python Tutor*. International Journal of Artificial Intelligence in Education. Springer.

Felder, R. M. (1993). *Reaching the Second Tier: Learning and Teaching Styles in College Science Education*. Journal of College Science Teaching. 23(5). pp.286-290

Felder, R. M., and Silverman, L. K. (1988). *Learning and teaching styles in engineering education.* Engineering education. 78(1). pp. 674-681.

Felder, R. M., and Spurlin, J. (2005). *Reliability and Validity of the Index of Learning Styles: A Meta-Analysis.* International Journal of Engineering Education. 21(1). pp.103-112.

Flor, N., V. and Hutchins, E., L., (1991). *A Case Study of Team Programming During Perfective Software Maintenance*. In Proceedings of the fourth annual workshop on empirical studies of programmers. Norwood, NJ.

Garner, S., Haden, P., and Robins, A. (2005). *My program is correct but it doesn't run: A preliminary investigation of NPs' problems*. Conferences in Research and Practice in Information Technology Series. 42(1), pp.173–180.

Giannakos, M. N., Pappas, I. O., Jaccheri, L., and Sampson, D. G. (2017). *Understanding student retention in CS education: The role of environment, gains, barriers and usefulness*. Education and Information Technologies. 22(5). Pp.2365-2382.

Glaser, B. (1992) Basics of grounded theory analysis. Mill Valley, CA: Sociology Press.

Glaser, B. and Strauss, A. (1967). The discovery of grounded theory: Strategies for Qualitative Research. Chicago.

Grasha, A. F. (1972). *Observations on relating teaching goals to student response styles and classroom methods*. American Psychological Journal.

Griffiths, M. (2014). *The guilty secret of the digital skills gap in UK*. New Statesman: Salford Business School. 143(5227). p.20.

Guardian. (2020). *Digital Divide 'isolates and endangers' millions of the UK's poorest in the world*. The Guardian Newspaper: UK. [online] <Available from: https://www.theguardian.com/world/2020/apr/28/digital-divide-isolates-and-endangers-millions-of-uk-poorest > [Last accessed: 19th July 2020]

Guzdial, M., Ericson, B. and Biggers, M. (2005). *A model for improving secondary CS education*. SIGCSE.

Hammedi, W., Leclercq, T., Poncin, I., and Alkire, L. (2021). *Uncovering the dark side of gamification at work: Impacts on engagement and well-being*. Journal of Business Research. 1(12). pp. 256-269.

Hanks, B. (2006). *Student attitudes toward pair programming*. In Proceedings of the 11th annual conference on innovation and technology in CS education. ACM.

Harms, K. J., Balzuweit, E., Chen, J., and Kelleher, C. (2016). *Learning programming from tutorials and code puzzles: Children's perceptions of value*. Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium. pp.59-67.

Harms, K. J., Rowlett, N. and Kelleher, C. (2015). *Enabling independent learning of programming concepts through programming completion puzzles*. in Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on. 271–279.

HESA. (2020). *Non-continuation: UK Performance Indicators 2018/19*. HESA website. [online] Available from: <https://www.hesa.ac.uk/data-and-analysis/performance-indicators/non-continuation-1819>

Helminen, J., Ihantola, P., Karavirta, V., and Malmi, L., (2012). *How Do Students Solve Parsons Programming Problems? – An Analysis of Ineraction Traces*. In Proceedings of the International Computing Education Research Conference (Aukland, New Zealand2012), ACM. pp.119-126.

Hour of Code. (2020). *Hour of Code*. [Online]. Available: https://code.org/

Hnin, W. Y. (2017). *Personalized learning pathways using code puzzles for NPs*. 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 327-328.

Hu, S. and Kuh, G. D. (2000). *A multilevel analysis on student learning in colleges and universities*. Association of Study for Higher Education. ERIC.

Hu, S. and Kuh, G. D. (2002). *Being (Dis)Engaged in Educational Purposefully Activities: The Influences of Student and Institutional Characteristics*. Research in Higher Education. 43(5).

Hu, M. (2015). *Teaching Novices Programming: A Programming Process Using Goals and Plans with a Visual Programming Environment*. University of Otago, Dunedin, New Zealand.

Hulkko, H.; and Abrahamsson, P., (2005*). A multiple case study on the impact of pair programming on product quality*. In Proceedings of the 27th international conference on software engineering. ACM.

Hsu, C., and Wang, T. (2014). *Enhancing concept comprehension in a web-based course using a framework integrating the learning cycle with variation theory*. Asia Pacific Education Review. Netherlands: Springer. 15, 2, 211-222.

Ihantola, P. and Karavirta, V. (2011). *Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations*. Journal of Information Technology Education 10, pp.1-14.

Ito, H., Shimakawa, H., and Harada, F. (2020). *Comprehension analysis considering programming thinking ability using code puzzles*. 15th Conference on Computer Science and Information Systems (FedCSIS). pp609-618.

Janpla, S. and Piriyasurawong, P. (2018). *The Development of Problem-Based Learning and Concept Mapping Using a Block-Based Programming Model to Enhance the Programming Competency of Undergraduate Students in CS.*. TEM Journal. 7(4), pp.708-716.

Kaila, E., Rajala, T., Laakso, M.J. and Salakoski, T. (2008). *Automatic Assessment of Program Visualization Exercises*. Appeared in the 8th Koli Calling International Conference Proceedings.

Kauffmann, C.; Mense, A.; Wahl, H.; and Pucher, R. (2011). *Reducing the drop-out rate of a technical orientated course by introducing Problem based learning – a first concept*.

Kavitha, R., and Ahmed, M. I. (2015). *Knowledge sharing through pair programming in learning environments: An empirical study*. Education and Information Technologies. pp.319-333.

Khazaei, B. and Jackson, M. (2002). *Is there any difference in novice comprehension of a small program written in the event-driven and object-orientated styles?.* Proceedings IEEE 2002: Symposia on Human Centric Computing Languages and Environments. CA:USA. pp.19-26

Kölling, M. (1999a). *The problem of teaching object-oriented programming, Part 1: Languages*. Journal of Object-Oriented Programming*, 11*(8), 8-15.

Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. (2003). *The BlueJ system and its pedagogy*. CS Education. *13*(4). pp.249-268.

Lahtinen, E.; Ala-Mutka, K.; and Järvinen, E. (2005). *A study of the difficulties of NPs*. ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in CS education. 37(3). pp.14–18.

Lane, C. H. (2012). *Cognitive Models of Learning*. Encyclopaedia of the Sciences of Learning. Springer Link.

Leong, F. H. (2015). *Automatic detection of frustration of NPs from contextual and keystroke logs.* 10th International Conference on CS.

Lister, R. (2004). *A multi-national study of reading and tracing skills in NPs*. ACM SIGCSE.

Lister, R.; Simon, B.; Thompson, E.; Whalley J. L.; and Prasad, C. (2006). *Not seeing the forest of trees: NPs and SOLO Taxonomy*. SIGCSE. ACM Digital Libraries. 38(2). pp.118-122.

Looking Glass. (2020). *Looking Glass*. [Online]. Available from: <http://lookingglass.wustl.edu/>

Luciana-Floriana, H., Madalina-Ionela, I., Madalina-Lavinia, T., Illeana, V., and Georgiana-Geanina, B. (2020). *Students' Difficulties Into Understanding First-Year Subjects: A Premise For University Drop-Out. Case Study: The Bucharest University of Economic Studies, The Faculty of Business and Tourism*. Editura ASE Bucuresti. 2(2). pp.14-27.

McDowell, C., Hanks, B., and Werner, L., (2003). *Experimenting with pair programming in the classroom*. ACM SIGCSE Bulletin. pp.60-64.

Marton, F., and Tsui, A. B. M. (2004). *Classroom Discourse and the Space of Learning*. Mahwah, New Jersey: Lawrence Erlbaum Associates.

McDonald, C. (2016). *Digital skills gap costs UK economy £63bn a year*. Computer Weekly [online] <Available from: https://www.computerweekly.com/news/450298249/Digital-skills-gap-costs-UK-economy-63bn-a-year > [Last accessed: 17th July 2020]

Mosemann, R. and Wiedenbeck, S. (2001). *Navigation and comprehension of programs by NPs*. Proceedings 9th International Workshop on Program Comprehension. IEEE Conference. CA: USA. pp.79-88.

Oktay, J. S. (2012). *Grounded Theory*. Oxford University Press: Oxford.

Parson, D. and Haden, P. (2006). *Parsons programming puzzles: a fun and effective learning tool for first programming courses*. In Proceedings of the 8th Australasian Conference on Computing Education. pp.157-163.

Pellini, A. (2020*). Education during the COVID-19 Crisis: Low Income Countries*. EdTechHub. [online] Available from: < https://edtechhub.org/coronavirus/edtech-low-income-countries/>

Pérez-Álvarez, M. (2017). *The Four Causes of ADHD: Aristotle in the Classroom*. Frontiers in Psychology. 1(8).

Pitta-Pantazi, D., Christou, C., and Zachariades, T. (2007). *Secondary school students' levels of understanding in computing exponents*. Journal of Mathematical Behavior. 26. pp. 301-311.

Porter, L., Guzdial, M., McDowell, C., and Simon, B. (2013). *Success in introductory programming*. Communications of the ACM. pp.34-36.

Ragonis, N., and Ben-Ari, M. (2005). *On understanding the statics and dynamics of object-oriented programs*. ACM SIGCSE Bulletin, 37, 226-230.

Sanders, D. (2002). *Student perceptions of the suitability of extreme and pair programming*. Extreme programming perspectives. pp.168-174.

Santos, A. and Gorgônio, A. (2015). *A Importância do Fator Motivacional no Processo Ensino-Aprendizagem de Algoritmos e Lógica de Programação para Alunos Repetentes.* WEI – Workshop sobre Educação em Computação. pp.1–10.

Sawicki, M. (2013). *Body, Text and Science: The Literacy of Investigative Practices and the Phenomenology of Edith Stein*. Springer. ISBN 978-9401139793

Scardamalia, M. and Bereiter, C. (2006). *Knowledge Building Theory, Pedagogy, and Technology*. Cambridge Handbook of Learning Sciences. pp. 97-118.

Schutz, A. (1967). *The Phenomenology of the social world*. Evanston: North-western University Press.

Scott, W. G. (1963). *Communication and Centralisation of Organisation*. Journal of Communication. Wiley Online Library.

Scott, A. (2010). *Using Flowcharts, Code and Animation for Improved Comprehension and Ability in Novice Programming*. University of South Wales: UK.

Scratch. (2020). *Scratch*. [Online]. Available from: <https://scratch.mit.edu/>

Seaborn, K. (2021). *Removing Gamification: A Research Agenda*. CHI EA 2021.

Shadbolt, N. (2016). *Shadbolt Review of CSs Degree Accreditation and Graduate Employability*. UK Government. [Online]. Available from: < https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/518575/ind-16-5-shadbolt-review-computer-science-graduate-employability.pdf >

Shaffer, D.W., and Resnick, M. (1999). *"Thick" Authenticity: New Media and Authentic* Learning. Journal of Interactive Learning Research. 10(2). Pp.195-215.

Sharp, H., and Robinson, H. (2010). *Three 'C's of agile practice: Collaboration, co-ordination and communication*. Agile Software Development. Berlin: Springer.

Shee, D. Y. and Wang, Y. S. (2008). *Multi-criteria evaluation of the web-based e-learning system: A methodology based on learner satisfaction and its application*. ACM Digital Libraries.

Sleeman, D. (1986). *The challenges of teaching computer programming*. Communications of the ACM, New York, USA. 29(9), 840-841.

Smith, J. (2016). *On the Soul, by Aristotle written circa 350BCE*. The Internet Classics Archive. MIT.

Soanes, C. and Stevenson, A. (2005). *Oxford Dictionary of English (Revised Edition)*. Oxford University Press. England: Oxford. ISBN 0-19-861057-2.

Soloway, E. and Spohrer, J. (1989) *Studying the NP*. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Sprenger, M. (2003). *Differentiation through learning style and memory*. SAGE Publications.

Stapel, K., Knauss, E., Schneider, K., and Becker, M. (2010). *Towards understanding communication structure in pair programming*. Agile processes in software engineering and extreme programming. pp.117-131.

Strauss, A. and Corbin, J. (1994). *Grounded Theory Methodology: An Overview*. (1st ed) Handbook of Qualitative Research. pp.273-284

Sweller, J., van Merriënboer, J. J., and Paas, F. G. (1998). *Cognitive architecture and instructional design*. Educational Psychology Review. *10*(3), 251-296.

Thompson, E. (2008). *How do they understand? Practitioner perceptions of an object-oriented program*. Massey University, Palmerston North, New Zealand.

Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., and Robbins, P. (2008). *Bloom's taxonomy for CS assessment*. Proceedings of the Tenth Conference on Australasian Computing Education. pp.155-161.

Thornberg, R. and Charmaz, K. (2012). Grounded Theory. John Wiley: San Francisco, CA. pp.41-67

Utting, I. Tew, A. E., McCracken, M., Thomas, L. Bouvier, D., Fry, R., Paterson, J., Casperen, M. Kolikant, Y. B.D., Sorva, J. and Wilusz, T. (2013). *A fresh look at NPs' performance and their Teachers' Expectations*. University of Kent.

Vahldick, A., Mendes, A. Marcelino, M. J., Hogenn, M. J., and Schoeffel, P. (2015). *Testando a Diversão em um Jogo Sério para o Aprendizado. Introdutório de Programação*. XXIII Workshop sobre Educação em Computação.

Van Maanen, J. (1988). *Tales of the Field: On Writing Ethnography*. (2nd ed). Chicago: University of Chicago Press.

Voit, T., Schneider, A., and Kriegbaum, M. (2020). *Towards an Empirically Based Gamification Pattern Language using Machine Learning Techniques*. IEEE 32nd Conference on Software Engineering Education and Training (CSEEandT). pp.1-4.

Walters, R. (2018). *Solving the United Kingdom technology skill gap*. Available online: < https://www.robertwalters.co.uk/solving-the-uk-skills-shortage/technology-research.html>

Warr, P. B., Bird, M. W. and Rackham, N. (1970). *Evaluation of Management Training: A Practical Framework, with Cases, for Evaluation Training Needs and Results*. Gower Press: London.

Wiedenbeck, S.; Fix, V.; and Schlotz, J. (1993). *Characteristics of the mental representations of novice and expert programmers: an empirical study*. International Journal of Man-Machine Studies. 39(5). pp.793-812

Wiedenbeck, S.; Ramalingam, V.; Sarasama, S. and Corritore, C. (1999). *A comparison of the comprehension of object orientated and procedural programs by NPs. University of Nebraska: USA. Elsevier.*

Williams, L., Kessler, R., Cunningham, W., and Jeffries, R. (2000). *Strengthening the case for pair programming*. IEEE Software, 17, 19–25. doi:10.1109/52.854064

Williams, L., and Kessler, R., (2000). *All I really need to know about pair programming I learned in kindergarten*. Communications of the ACM. pp.108-114.

Williams, L. and Kessler, R. (2002). *Pair programming illuminated*. MA: Addison-Wesley.

Young, J. (1996). *The Coherence Theory of Truth*. Stanford University.

Zahavi, D. (2003). *Husserl's Phenomenology*. Stanford: Stanford University Press.

Záhorec, J., Hašková, A., and Munk, M. (2020). *Results of a Research Evaluating Quality of CS Education*. Informatics in Education. 11(2). pp.283-300. [Online: Available from: https://eric.ed.gov/contentdelivery/servlet/ERICServlet?accno=EJ1064274]

Zhang, Y., Dou, Y., Meng, X., Lai, Y., Lu, Y. and Wang, X. (2020). *Gamification of LR Algorithm: Engaging Students by Playing in Compiler Principle Cours*e. 15th International Conference on CS and Education (ICCSE). pp.396-400.

Zarb, M. and Hughes, J. (2015). *Breaking the communication barrier: guidelines to aid communication within pair programming*. CS education. Taylor and Francis. 25(2). pp.120-151.

Chapter 11. Appendices

The appendices contain relevant documents associated to the thesis, but not the raw data itself. Permission to access all the individual coded transcripts and data can be granted upon request.

11.1 Study Supplements

This section contains the supplements provided for all three studies.

11.1.1 CP1: Task Description

"Task 1:

Although this may seem a little strange, imagine that a shop that sells potatoes has contacted you as a developer and has requested you to set up a Java program that can simulate and effectively record the sales on their potato stock. For the first task, you should arrange the necessary puzzle pieces below to create a PotatoShop class!

A PotatoShop object needs to keep track of its: price for the potatoes, number of potatoes sold over the lifetime of the program, and the number of potatoes remaining in store that can be sold (you can't sell potatoes to customers if you don't have them!); the number of potatoes remaining in the store and the price of a potato are provided to the potato shop object when it is created.

This version of the PotatoShop class should have:

- A method that creates a potato object, setting up the corresponding number of potatoes sold, price of a potato and remaining potato values when called (in other words, initialises the instance of potato)
- A 'sellPotatoes' method which should check whether there are enough potatoes in the store to sell prior to selling the potatoes – if there are not enough potatoes to sell a null pointer exception should be triggered. If there are enough potatoes to sell then the calculated price of the sale should be returned.
- A 'calculateSale' method which should use the number of potatoes and calculate what the corresponding cost would be for that particular sale.

Remember to explain your movements to the observer as you do them".

Workspace:

Here are some pieces that you can use as reference:

19 `private int totalPotatoesRemainingInStore;`

`if(numOfPotatoes <= totalPotatoesRemainingInStore){` 20

1 `private double priceOfPotatoes;` 18 `priceOfPotatoes = price;` 17 `}`

21 `return null;` `return numOfPotatoes;` 22

2 `public PotatoShop(double price, int totalPotatoesInStore) {` 16 `else{` `}` `return priceOfPotatoes*numOfPotatoesSold;` 23

3 `private int numberOfPotatoesSold;` 14 `public class PotatoShop {` 15 `}`

`}` 24

4 `numberOfPotatoesSold = 0;` 13 `totalPotatoesRemainingInStore -= numOfPotatoes;` `numberOfPotatoesSold = 1;` 25

5 `public double sellPotatoes(int numOfPotatoes) {` `public double calculateSale(int numOfPotatoesSold){` 26

6 `numberOfPotatoesSold += numOfPotatoes;` `return calculateSale(numOfPotatoes);` 27

7 `numberOfPotatoesSold == numOfPotatoes;` 12 `}` `totalPotatoesRemainingInStore = totalPotatoesInStore;` 28

8 `numberOfPotatoesSold -= numOfPotatoes;` 11 `return priceOfPotatoes/numOfPotatoesSold;` `}` 29

9 `return priceOfPotatoes+numOfPotatoesSold;` 10 `return priceOfPotatoes-numOfPotatoesSold;`

A PotatoShop object needs to keep track of its: price for the potatoes, number of potatoes sold over the lifetime of the program, and the number of potatoes remaining in store that can be sold (you can't sell potatoes to customers if you don't have them!); the number of potatoes remaining in the store and the price of a potato are provided to the potato shop object when it is created.

This version of the PotatoShop class should have:

A method that creates a potato object, setting up the corresponding number of potatoes sold, price of a potato and remaining potato values when called (in other words, initialises the instance of potato)

A 'sellPotatoes' method which should check whether there are enough potatoes in the store to sell prior to selling the potatoes – if there are not enough potatoes to sell a null pointer exception should be triggered. If there are enough potatoes to sell then the calculated price of the sale should be returned.

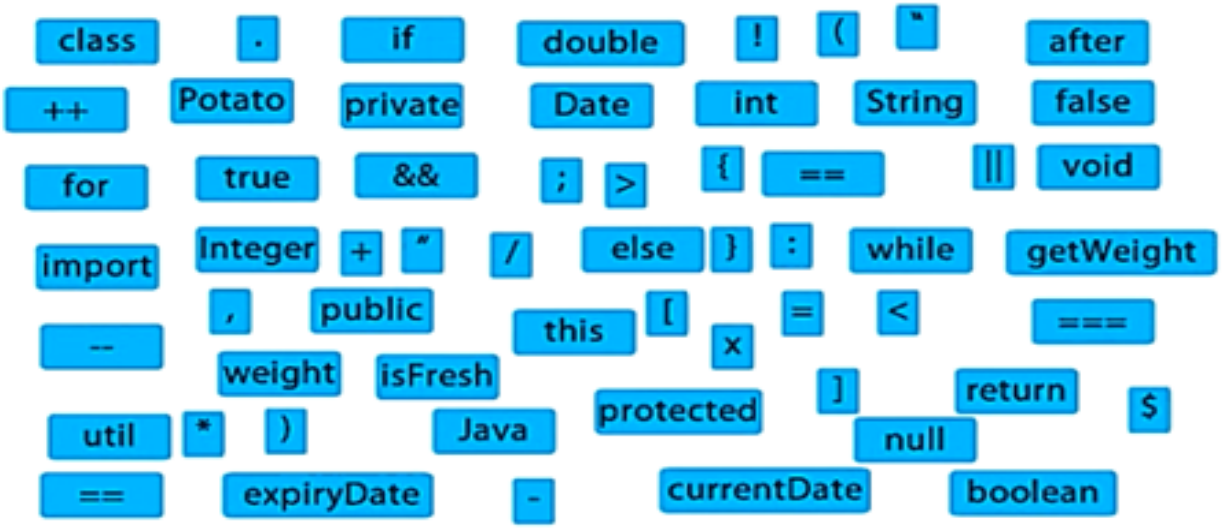
A 'calculateSale' method which should use the number of potatoes and calculate what the corresponding cost would be for that particular sale. ***Remember to explain your movements to the observer as you do them.***

Final Solution Space:

```
public class PotatoShop {
```

Workspace:

Here are some pieces that you can use as reference:



A Potato object needs to keep track of its own weight and expiry date; these values are provided to the potato object when it is created.

This version of the Potato class should have a method that creates a potato object, setting up the corresponding weight and expiry date values when called (in other words, initialises the instance of potato), and an 'isFresh' method which should check whether the potato is spoiled using the current date (provided externally for the method) and expiry date provided by the potato object itself. The 'isFresh' method should return a true or false value depending on whether the potato is in date (fresh) or out of date (spoiled). You can choose whether to use an Integer value for the current day and compare it to another Integer value for the expiry day, and assume the current and expiry day is always in the same month, or, you can make use of the Date object from the Java libraries.

You can create your own pieces; write the solution in full!

Final Solution Space:

```
public class Potato {
```

11.1.3 CP2: Task Description

"**Task 2:**

Create a Potato class using the necessary puzzle pieces below! Remember to explain your movements to the observer as you do them.

A Potato object needs to keep track of its own weight and expiry date; these values are provided to the potato object when it is created.

This version of the Potato class should have a method that creates a potato object, setting up the corresponding weight and expiry date values when called (in other words, initialises the instance of potato), and an 'isFresh' method which should check whether the potato is spoiled using the current date (provided externally for the method) and expiry date provided by the potato object itself. The 'isFresh' method should return a true or false value depending on whether the potato is in date (fresh) or out of date (spoiled).

As reference, the Java.util.Date library is being used to create Date-typed variable(s). The following library method, described by Oracle (2017), has been used in the solution to this puzzle:

> after
> public boolean after(Date when)
>
> Tests if this date is after the specified date.
>
> Parameters:
> when – a date.
>
> Returns:
> true if and only if the instant represented by this Date object is strictly later than the instant represented by when; false otherwise.
>
> Throws:
> NullPointerException – if when is null.
>
> Oracle. (2017). Class Date. [online] Available From: < https://docs.oracle.com/javase/8/docs/api/java/util/Date.html#after-java.util.Date-> [Last Accessed: 4th April 2017]

"

11.1.4 Blackboard Announcements

These were used to advertise the study to potential students.

11.1.4.1 Pilot and Secondary Study Announcement

"[Announcement Title:] Wanted – First Year Computing Students to Participate in Code Puzzle Research"

"[Announcement Body:]

Dear all,

A pilot investigation is being conducted over the coming year into the effectiveness of a prototype research algorithm that is being used to try to estimate your level of understanding about Computing-related concepts based on how you move puzzle pieces that are related to Java code. It would be extremely helpful if there were volunteers who would not mind offering their time for: a 10 minute signing of consent form, and a 30 minute recorded and observed experiment based on arrange puzzle pieces with Java coding on them to form a solution.

Prior to the experiment, you would need to be willing to meet me for a minimum of 10 minutes to sign a consent form in regards to data protection and your rights to the information produced from the study as well as being allocated an anonymous ID number (AIN). Your lecturers will not be informed about whether or not you participate in the study, nor will they be informed about the results of specific students, nor will this experiment impact your grade in any way. I will clarify, your name will not be kept next to your data (only you will have access to the mapping between your name and AIN). If you choose to provide your e-mail, you may also be invited to a follow-up session where the results of the experiment are discussed with you. This follow-up session will vary in terms of length, and will depend largely on the results yielded from the system as well as any discussion on the accuracy of these results. You must bring your AIN to the follow-up session as I will not hold mapping to your name.

This experiment would be particularly beneficial for students as the goal of the system is to try to highlight areas of understanding in regards to Object-orientated design. I would like to emphasize that this can be advantageous for a whole range of abilities, and would encourage all students to consider participating.

Feel free to contact me ([REDACTED]) if you would are potentially interested in participating in the study, or, if you have any questions or require further information about the study.

Kind regards,

Katrina Jones

(Aston University Computing Education PhD Student)"

11.1.4.2 Tertiary Study Announcement

[Announcement Title:] “Wanted – First Year Computing Students to Participate in Code Puzzle Research”

[Announcement Body:] “Hi,

Do you fancy a revision opportunity where you will get to interact with Java in a novel way? Do you like solving puzzles and want to brush up on basic Java concepts? Or do you simply not know where you would start if someone asked you to complete a Java class and feel that having an experienced Java programmer talk to you about your understanding would be greatly appreciated?

My name is Katrina Jones and I am a PhD student here at Aston University. As part of my PhD studies, I am conducting an investigation into the effectiveness of a decision tree algorithm that I would like to use to try to estimate students’ level of understanding about computing-related concepts so that we can find more effective ways to teach programming to students such as yourself.

In order to collect data for my investigation, I like to invite you to complete a series of online Java code puzzles. I will record how you move the puzzle pieces and your reasoning for moving them the

way you do in order to arrive at a working Java class. From this, I hope to identify your understanding of the code, syntax, and computational concepts. Due to the current COVID-19 restrictions, this study is being conducted via Blackboard Collaborate Ultra accessed via your Java Programming Foundations module. Participation would take approximately 1 hour of your time. Following the study session itself, I will provide you with some feedback on your coding approaches/strategy that I hope will be helpful for you as you revise for your assessments.

Your lecturers will not be informed about whether or not you participate in the study, nor will they be informed about the performance of specific students. Recorded data will be coded and thus anonymised so that you will not be identifiable from the retained data.

If you are potentially interested in participating, please contact me ([REDACTED]). I will send you further details regarding your potential participation and be able to answer any questions you might have. I will e-mail you a consent form and participant information sheet so that you can make an informed decision as to whether you'd ultimately like to participate: your participation should be entirely voluntary.

Kind regards,

Katrina Jones

Aston University Computing Education PhD Student"

11.1.5 Questionnaires

11.1.5.1 Pre-Puzzle Questionnaire (Secondary and Tertiary Study Only)

Pre-CP2 Questionnaire

Puzzle ID: [Enter here]

Participant: [Enter here]

YOUR EXPERIENCE

Please indicate with a tick which statement(s) you agree with the most in regards to the puzzle you have just done:

	Statement	**Puzzle (tick)**
1. How difficult do you think this task will be?	Very easy	
	Fairly easy	
	Slightly easy	
	Neither easy nor difficult	
	Slightly difficult	
	Fairly difficult	
	Very difficult	

What do you think will be the easiest part of this task (and why do you think this)?

What do you think will be the hardest part of this task (and why do you think this)?

11.1.5.2 Post-Puzzle Questionnaire

Puzzle ID: [Enter here]

Participant: [Enter here]

YOUR EXPERIENCE (after the task was performed)

Please indicate with a tick which statement(s) you agree with the most in regards to the puzzle you have just done:

	Statement	**Puzzle (tick)**
2. Do you think your solution would run without errors?	Yes, I think I have a fully working, correct solution.	
	Maybe, I think I have most of it correct. Maybe I have some details wrong, though?	
	No, I think some of the code is in the right order but some of it is not.	
	No, I think I have the majority of the solution wrong and I imagine it'd create errors at run time.	
	I'm not sure, it might compile or it might not.	

Any further comments on your solution?

	Statement	**Puzzle (tick)**
3. How challenging was the puzzle for you?	Very easy; I didn't struggle at all.	
	Fairly easy; I almost saw the solution straight away.	
	Slightly easy; I could see how to solve it with some thought.	
	Neither easy nor difficult; I think it was just right for me.	
	Slightly difficult; It took me a while to figure out a solution.	
	Fairly difficult; it took a long time for me to see any kind of solution.	
	Very difficult; I really struggled to find a solution.	

Any further comments on the level of difficulty of this puzzle?

11.1.5.3 Post Study Questionnaire (Secondary and Tertiary Study Only)

Post-Experiment Questionnaire

Participant: [Enter here]

YOUR EXPERIENCE

Please indicate which statement(s) you agree with in regards to the study you have just partaken in:

Do you believe the Code Puzzles would be of use to you?	Yes/No
Do you prefer Code Puzzles over other revision techniques?	Yes/No
Would you use Code Puzzles in conjunction with other revision techniques?	Yes/No

Do you feel that the study accurately portrayed your approach? Why do you feel this way?

Do you think the analysis did reflect on your understanding or were the findings inaccurate? (Please be honest).

11.1.5.4 Background Questionnaire (Secondary and Tertiary Study Only)

Participant ID: [Enter here]

YOUR EXPERIENCE

Please indicate with a tick which statement(s) you agree with the most in regards to your past experience with programming:

	Statement	**Mark with an 'X'**
1. How confident are you in your ability as a programmer?	Very confident	
	Fairly confident	
	Slightly confident	
	Neutral	
	Slightly unconfident	
	Fairly unconfident	
	Very unconfident	

How many programming languages would you say you are fluent in?	
How many programming languages would you say you are proficient in?	
How many programming languages would you say you are a beginner in?	

Please list the languages you are proficient or fluent in:

What qualities does a programmer require (in your opinion)?

Which qualities of a programmer do you feel you have?

Which qualities of a programmer do you feel you need to improve on?

What is the most important aspect of understanding programming (in your experience)?

Can you describe the steps you take to solve a programming task?

11.1.6 Consent Forms and Participant Information Sheet

11.1.6.1 Pilot Study Consent Form and Participant Information Sheet

INFORMED CONSENT STATEMENT

You are formally invited to participate in study which aims to assess the accuracy and effectiveness of analysing puzzle-based interactions for estimating a learner's level of understanding about a puzzle's core topic.

EXPERIMENT INFORMATION

To help us we will ask you to participate in a 15 to 30 minute test session where we will ask you to arrange Java code puzzle pieces into a working solution while explaining your movements and reasoning to the observer as you do so. If you complete a puzzle, you will be asked if you would like to complete another if there is enough time remaining and another puzzle is available to be completed during the session. Your speech, movement time, and type of interaction will be recorded both using a recorder and by the observer themselves. The information retrieved from your session will be then analysed using a custom algorithm and we will use your speech in an attempt to estimate the accuracy of the algorithm result. Your information will be grouped with information from other participants to help us assess the accuracy and effectiveness of the algorithm created by us.

FOLLOW-UP FEEDBACK INFORMATION

If you choose to provide an e-mail address, once the information gathered from the research has been processed, we will contact you to try and arrange a final meeting that is convenient for you where we will provide feedback based on our algorithm. This will be very beneficial for us as we may gain a better idea of the accuracy of our algorithm's result based on your feedback, and it may be beneficial to yourself for a reflection on your strengths and weaknesses when dealing with object-orientated programming. Please note that our study is aiming to assess the accuracy of this algorithm, so, do not be afraid to say that our algorithm is correct or incorrect. We ask you to be honest!

YOUR RIGHTS

This section reminds you of your rights as a participant, please read through this carefully as we regard your signature as an agreement and acknowledgement of these rights.

RISKS

We do not perceived any foreseeable risks associated with this particular study.

CONFIDENTIALITY AND WITHDRAWING

The data obtained through this study will be treated as confidential information; it will be stored securely on the Aston University system.

After signing this consent form you will be assigned an Anonymised Identification Number (AIN), your AIN will be associated to the data only, and a mapping between your name and your AIN will not be kept. **It is your responsibility to keep your AIN if you wish to withdraw your data.** Your name will never be used in reports or publications. The lecturers at Aston University will not be able to map your AIN with your name, and as such, you will not be treated differently whether you choose to participate or not.

Your participation in this study is voluntary; you can decline to participate without any penalty or loss of benefits to which you are otherwise entitled to. If you decide to participate you may discontinue the participation at any time without penalty or loss of benefits to which you are otherwise entitled to. If you discontinue prior to your study ending and have your AIN, we will guarantee that any data related to you will be destroyed. If you wish to withdraw after the study has ended and you have your AIN, we will do our best to ensure that local copies of your data are destroyed. This is primarily due to Aston University having an open policy in regards to data in publications, which means that if you do not withdraw prior to the submission of a research paper it will be impossible to ensure that your data is destroyed.

COMPENSATION

Unfortunately, we cannot offer any other incentive other than contributing to computing education research.

CONTACT INFORMATION

If you have any queries at any time about the study, you may contact the principal investigator, Katrina Jones, at [REDACTED].

If you wish to register a complaint as you have not been treated in the way that this form suggests, or you feel as if your rights have been violated, please contact the Ethics Committee at Aston University using the following URL: https://www.ethics.aston.ac.uk/contact

CONSENT

I have read and understood the above information. I have received a copy of this form, and a personalised AIN which I will keep safe and bring to the experiment study. I agree to participate in this study and have my data contribute to publications.

Participant's Signature: _____ Date: _____

Investigator's Signature: _____ Date: _____

I wish to be contacted by the Principal Investigator for a Follow-Up Feedback Session, and agree to give them my e-mail in order to contact me for this reason: ☐

Participant's E-mail: _____

11.1.6.2 Secondary Study Consent Form and Participant Information Sheet

INFORMED CONSENT STATEMENT

You are formally invited to participate in study which aims to explore the ways in which students build their programs, and how this may reflect a student's level of understanding about their code and related computational concepts to that code.

EXPERIMENT INFORMATION

To help us we will ask you to participate in an hour test session where we will ask you to arrange Java-based Code Puzzle pieces into a working solution while explaining your movements to the observer. The observer will hand you two pieces of coloured card:

- Red card: raise this card when you wish to ***stop the puzzle*** (this [is] the equivalent to submitting a solution or stopping the experiment);
- Yellow card: raise this card if you wish to ***ask a question*** to the observer (this is the equivalent to pausing the puzzle).

There will be two Code Puzzles; not all of the pieces necessarily need to be used in order to complete the puzzle. There will be six mini questionnaires that we will ask you to complete; one prior to starting the experiment (based on your experiences with coding), one before each puzzle (asking you about your perceptions of the task based on the specification), one after each puzzle (to assess the difficulty of the task in retrospect and your confidence in the solution working), and one at the end (assessing the accuracy of your results). Your speech and movement will be recorded both using a recorder and by the observer themselves. The information will then be discussed at the end o the session for roughly 10 minutes where any issues that are noticed will be talked about. Your information will be grouped with the information from other participants to help us assess whether the way a student builds their programs does reflect on their understanding of the code.

YOUR RIGHTS

This section reminds you of your rights as a participant, please read through this carefully as we regard your signature as an agreement and acknowledgement of these rights.

RISKS

We do not perceived any foreseeable risks associated with this particular study.

CONFIDENTIALITY AND WITHDRAWING

The data obtained through this study will be treated as confidential information; it will be stored securely on the Aston University system.

After signing this consent form you will be assigned an Anonymised Identification Number (AIN), your AIN will be associated to the data only, and a mapping between your name and your AIN will not be kept. **It is your responsibility to keep your AIN if you wish to withdraw your data.** Your name will never be used in reports or publications. The lecturers at Aston University will not be able to map your AIN with your name, and as such, you will not be treated differently whether you choose to participate or not.

Your participation in this study is voluntary; you can decline to participate without any penalty or loss of benefits to which you are otherwise entitled to. If you decide to participate you may discontinue the participation at any time without penalty or loss of benefits to which you are otherwise entitled to. If you discontinue prior to your study ending and have your AIN, we will guarantee that any data related to you will be destroyed. If you wish to withdraw after the study has ended and you have your AIN, we will do our best to ensure that local copies of your data are destroyed. This is primarily due to Aston University having an open policy in regards to data in publications, which means that if you do not withdraw prior to the submission of a research paper it will be impossible to ensure that your data is destroyed.

COMPENSATION

You will receive two £5 Amazon vouchers at the end of the study, regardless of whether or not you complete working solutions.

CONTACT INFORMATION

If you have any queries at any time about the study, you may contact the principal investigator, Katrina Jones, at [REDACTED].

If you wish to register a complaint as you have not been treated in the way that this form suggests, or you feel as if your rights have been violated, please contact the Ethics Committee at Aston University using the following URL: https://www.ethics.aston.ac.uk/contact

CONSENT

I have read and understood the above information. I have received a copy of this form, and a personalised AIN which I will keep safe and bring to the experiment study. I agree to participate in this study and have my data contribute to publications.

Participant's Signature: _____ Date: _____

Investigator's Signature: _____ Date: _____

I wish to be contacted by the Principal Investigator for a Follow-Up Feedback Session, and agree to give them my e-mail in order to contact me for this reason: ☐

Participant's E-mail: _____

11.1.6.3 Tertiary Study Consent Form

Investigation into the effectiveness of categorising the interactions that students have with Computing-related puzzle pieces to identify their level of understanding.

Consent Form

Name of Chief Investigator: Katrina Jones

Please initial boxes

1.	I confirm that I have read and understand the Participant Information Sheet 3.0 20/04/2020 for the above study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.	
2.	I understand that my participation is voluntary and that I am free to withdraw at any time, without giving any reason and without my legal rights being affected.	
3.	I agree to my personal data and data relating to me collected during the study being processed as described in the Participant Information Sheet.	
4.	I agree to my session being audio/video recorded and to anonymised direct quotes from me being used in publications resulting from the study.	
5.	I agree to my anonymised data being used by research teams for future research.	
6.	I agree to take part in this study.	

_____ _____ _____

Name of participant	Date	Signature
_____	_____	_____
Name of Person receiving consent.	Date	Signature

11.1.6.4 Tertiary Study Participant Information Sheet

How useful are Code Puzzles for determining learner understanding?

Participant Information Sheet

Invitation

We would like to invite you to take part in a research study.

Before you decide if you would like to participate, take time to read the following information carefully and, if you wish, discuss it with others such as your family, friends or colleagues.

Please ask a member of the research team, whose contact details can be found at the end of this information sheet, if there is anything that is not clear or if you would like more information before you make your decision.

What is the purpose of the study?

The aim of this study is to investigate the usefulness, effectiveness, reliability, accuracy and feasibility of using code puzzles to inform students' understanding of programming concepts and paradigms.

Why have I been chosen?

You are being invited to take part in this study because we believe you are a current undergraduate student in a Computing-related discipline (e.g., CS, CS with Business, Multimedia student, Computing with Mathematics) at Aston University.

Unfortunately, you will not be able to participate if any of the following applies to you:

- You are not a current Aston University Undergraduate student who is studying a computing-related discipline;
- You are not at least 18 years of age;
- You have failed your first term of your first year of your undergraduate degree;
- You are not able to access Blackboard Collaborate Ultra; and
- You are clinically blind as there is currently unfortunately no audio substitute for Code Puzzles.

What will happen to me if I take part?

You will be required to participate in one 40-60 minute online session, via Blackboard Collaborate Ultra which you can access via the Java Programming Foundations module on Blackboard (i.e., you will not be required to install any additional software on your computer).

You will be asked to complete a series of code puzzles. For each, you will be presented with a series of pieces of code and you will be required to arrange those pieces into a working Java class to solve a problem. You do not have to use all the pieces required; you will also be able to create pieces of your own to use if you prefer a different approach to the solution.
As you complete the puzzles, you will be asked to verbally explain your actions and reasoning. Your actions and your narrative will be automatically audio/video recorded within the software for analysis; the researcher will also be taken written notes as you work.

You will be also be required to:

- Fill in a short background questionnaire that focuses on your general experience with programming and thoughts on coding;
- Fill in a very short first impressions questionnaire before each code puzzle;
- Fill in a very short final impressions questionnaire after each code puzzle; and
- Finally, after a short debriefing during which you will receive feedback from the researcher on your coding strategy, fill in a very short questionnaire about your impressions of code puzzles and their usefulness.

How will my narrative and any conversation during the session be recorded and the information I provide managed?

With your permission we will audio record your narrative/discussion and take notes. The recording will be typed into a document (transcribed) by the researcher. This process will involve removing any information which could be used to identify individuals e.g. names, locations etc.

Audio recordings will be destroyed as soon as the transcripts have been checked for accuracy.

We will ensure that anything you have told us that is included in the reporting of the study will be anonymous.

You of course are free not to say anything you don't want to or not to answer any questions that you are asked without giving a reason.

How will the video recordings made during the study be managed?

The video recordings will be destroyed as soon as the research team have analysed the information in them to answer the research question.

We will ensure that anything from the analysis of the videos that is included in the reporting of the study will be anonymous.

Do I have to take part?

No. It is up to you to decide whether or not you wish to take part.

If you do decide to participate, you will be asked to sign and date a consent form. You would still be free to withdraw from the study at any time without giving a reason.

Will my taking part in this study be kept confidential?

Yes. A code will be attached to all the data you provide to maintain confidentiality.

Your personal data (name and contact details) will only be used if the researchers need to contact you. Analysis of your data will be undertaken using coded data.

The data we collect will be stored in a secure document store (paper records) or electronically on a secure encrypted mobile device, password protected computer server or secure cloud storage device.

To ensure the quality of the research, Aston University may need to access your data to check that the data has been recorded accurately. If this is required, your personal data will be treated as confidential by the individuals accessing your data.

Staff will not be informed of you participating or not participating in this study, nor will they view the recordings themselves or hear your voice. This will not affect your course or mark in any way, and no credit will be given to you for participating in this study.

What are the possible benefits of taking part?

The data collected during this study will be used as part of the researcher's PhD thesis to further knowledge in the area of using code puzzles as an analytical tool for understanding and teaching programming. It is hoped, however, that the feedback delivered to you personally during the debriefing session will be of direct benefit to you: it should hopefully help you to understand any potential weaknesses in your understanding of coding.

What are the possible risks and burdens of taking part?

We do not perceive any foreseeable risks associated with this particular study beyond those associated with normal study
We do acknowledge, however, that if you struggle to complete the puzzles it might cause you to lose some confidence in your coding abilities. If that is the case, we would encourage you to contact the module tutor for additional advice or the programming support officer for additional programming support.

What will happen to the results of the study?

The results of this study may be published in scientific journals and/or presented at conferences. If the results of the study are published, your identity will remain confidential.

A lay summary of the results of the study will be available for participants when the study has been completed and the researchers will ask if you would like to receive a copy.

Expenses and payments

You should not incur any expense in order to participate in this study. We are, unfortunately, unable to offer financial incentive.

Who is funding the research?

The study is being funded by Aston University School of Engineering and Applied Science (EAS).

Who is organising this study and acting as data controller for the study?

Aston University is organising this study and acting as data controller for the study. You can find out more about how we use your information in Appendix A.

Who has reviewed the study?

This study was given a favorable ethical opinion by the EAS Research Ethics Committee.

What if I have a concern about my participation in the study?

If you have any concerns about your participation in this study, please speak to the research team and they will do their best to answer your questions. Contact details can be found at the end of this information sheet.

If the research team are unable to address your concerns or you wish to make a complaint about how the study is being conducted you should contact the Aston University Research Integrity Office at research_governance@aston.ac.uk or telephone 0121 204 3000.

Research Team

Principal Investigator/Researcher: Miss Katrina Jones (jonesk6@aston.ac.uk)
Primary Supervisor: Dr. Tony Beaumont (a.j.beaumont@aston.ac.uk)

Thank you for taking time to read this information sheet. If you have any questions regarding the study please don't hesitate to ask one of the research team.

Aston University takes its obligations under data and privacy law seriously and complies with the General Data Protection Regulation ("GDPR") and the Data Protection Act 2018 ("DPA").

Aston University is the sponsor for this study based in the United Kingdom. We will be using information from you in order to undertake this study. Aston University will process your personal data in order to register you as a participant and to manage your participation in the study. It will process your personal data on the grounds that it is necessary for the performance of a task carried out in the public interest (GDPR Article 6(1)(e). Aston University may process special categories of data about you which includes details about your health. Aston University will process these datapoints on the grounds that it is necessary for statistical or research purposes (GDPR Article 9(2)(j)). . Aston University will keep identifiable information about you for 6 years after the study has finished.

Your rights to access, change or move your information are limited, as we need to manage your information in specific ways in order for the research to be reliable and accurate. If you withdraw from the study, we will keep the information about you that we have already obtained. To safeguard your rights, we will use the minimum personally identifiable information possible.

You can find out more about how we use your information at www.aston.ac.uk/dataprotection or by contacting our Data Protection Officer at dp_officer@aston.ac.uk.

If you wish to raise a complaint on how we have handled your personal data, you can contact our Data Protection Officer who will investigate the matter. If you are not satisfied with our response or believe we are processing your personal data in a way that is not lawful you can complain to the Information Commissioner's Office (ICO).

11.1.7 Ethics Submission: Amendment Documentation

NOTICE OF MINOR OR SUBSTANTIAL AMENDMENT

To be completed in typescript by the Chief Investigator in language comprehensible to a lay person and submitted to the both the Secretary and Chair of the School of Engineering and Applied Science Ethics Committee via email j.leigh@aston.ac.uk/j.lumsden@aston.ac.uk.

Details of Chief Investigator: Name: Katrina Jones Telephone: [REDACTED] Email: [REDACTED]

Full title of study:	Investigation into the effectiveness of categorising the interactions that students have with Computing-related puzzle pieces to identify their level of understanding.

REC reference number:	Ethics Submission 1115
Date study commenced:	20th February 2017
Amendment number and date:	Amendment 2.0. An initial amendment was filed and approved on 24.07.2017 to permit payment to participants in order to address poor recruitment rates. This second amendment is dated 24.06.2020.

Type of amendment (indicate all that apply in bold)

(a) Amendment to information previously given on the Ethics Application Form

Yes / ~~No~~

If yes, please refer to relevant sections of the REC application in the "summary of changes" below.

(b) Amendment to the protocol

Yes (see changes) / ~~No~~

If yes, please submit either the revised protocol with a new version number and date, highlighting changes in bold, or a document listing the changes and giving both the previous and revised text.

(c) Amendment to the information sheet(s) and consent form(s) for participants, or to any other supporting documentation for the study

Yes (see changes) / ~~No~~

If yes, please submit all revised documents with new version numbers and dates, highlighting new text in bold.

Is this a modified version of an amendment previously notified to the REC and given an unfavourable opinion?

No, the original version of this application (#1115) was first submitted on 24th February 2017, and was approved on 24th March 2017. Due to poor participant recruitment, a modification was added on 23rd July 2017 to grant permission to use £5-£10 Amazon Vouchers as an incentive. This amendment was approved on 24th July 2017.

(a) Amendments to information previously given on the Ethics Application Form

- **Change of end date (from 31st December 2019 to 7th September 2020):** due to my six months leave of absence from July 2019 to January 2020 my research was temporarily paused and I have been unable to complete my data collection as a result. Ideally, I require a larger sample size than has been collected to date for my PhD research. As such, I would like to request an extension of the study period to 7th September 2020.
- **Change of supervisor (from Dr. Errol Thompson to Dr. Tony Beaumont**): I had a change in primary supervisors due to the retirement of Dr. Errol Thompson. My new primary supervisor is Dr. Tony Beaumont: a.j.beaumont@aston.ac.uk / 0121 204 3447.

(b) Amendment to the Protocol

- **Change to recruitment and consenting process**: Similar to the pilot study, the participants will be contacted through an announcement via BB on the Java Programming Foundations module. If they reply to the announcement, the participant information sheet (PIS) and consent form will be emailed to them for review. If, after reviewing the PIS and consent form, they still wish to participate, they will be required to complete, sign and email the consent form to the researcher before study participation commences. Given the entirely online process amid COVID-19 restrictions, consent forms will need to be signed one of two ways: (1) the student can digitally sign the PDF document (using an Adobe signature or hand written digital signature); or (2) the student can print, physically sign, and photograph/scan the signed consent form. Once the student's consent form has been received, (s)he will be invited to an individual Blackboard Collaborate Ultra study session where the consent form and participant information sheet will be read through again to confirm understanding before commencing the study activities.
- **Change of face-to-face format, to online format:** COVID-19 restrictions now mean it is impossible to conduct in-person, paper-based Code Puzzles as per the approved procotol. Instead, participants will be asked to complete the same task but using electronic task code puzzle sheets via Blackboard Collaborate Ultra. Their activities and narrative will be recorded using the tools embedded within Collaborate Ultra and these recordings processed in the same way as originally articulated for video recordings of in-person sessions.
- **Anonymity of student:** Blackboard Collaborate Ultra sessions will be individually set up for participants at a time that suits them. These will only be accessible to the individual student and module tutors. Module tutors will be advised that the sessions are running and asked to avoid entering the sessions; there is no way to prevent the module tutor from entering as (s)he has all-access permission on BB but we will rely on the module tutor's professional integrity not to breach this request. Furthermore, to reinforce the private nature of the session and avoid student identity being disclosed, they will be entitled 'Experiment In Progress: Do not Enter' and will only be visible to those permitted to enter and the module tutor. Individual participants will be asked to enter a Collaborate session using a 'guest access' link that will be provided to them. On entering, Collaborate asks them to self-identify and they will be instructed to enter 'Study Participant' instead of their name. Their guest role will be set to 'Presenter' so that they can then anonymously share their desktop. Unfortunately, there is no way to guarantee participant attendance isn't recorded in the backend of BB but if the session is deleted after the recording is downloaded, this should minimise the likelihood of the participant's anonymity being breached. We feel these steps – the maximum possible within the enabling technology – should allow us to protect student anonymity as far as possible and that the remaining risk is acceptable given the nature of the study activities and data collection.

- **No monetary incentive:** amendment #1, as approved, allowed for participants to receive Amazon vouchers for their participation. It is no longer feasible to (a) access the vouchers for distribution nor (b) to safely distribute them to participants and so no incentive will now be provided.

Additional Changes:

- **Change in length of time of experiment (from 30-40 minutes to 40-60 minutes):** via practical experience, it was discovered that most participants could not complete the study within the originally estimated 30-40 minutes. To accommodate this as well as to factor for the new online deployment we would like to extend the estimated/stated participation length to 40-60 minutes.
- **Number of questionnaires given during the study:** participants, according to the 2017 ethics application, were originally given two questionnaires to complete after the code puzzles were completed. These questionnaires gauged the difficulty and confidence of the participants in regards to their submission. It was felt that, after piloting this, that more information needed to be gathered and so four other questionnaires were introduced (see appendix for all of these): two are associated to the two code puzzles, and ask about the perceived difficulty of the task itself before participants begin the puzzles, one is completed at the end after the follow-up meeting where participants are asked about the perceived usefulness of the code puzzle to them personally, and one solicits necessary background information on:
    - The number of programming languages participants know and at what level;
    - The names of the programming languages participants know;
    - How confident participants feel with programming in general;
    - Participants' thoughts on what qualities a programmer should possess;
    - Participants' thoughts on what qualities they, themselves, possess; and
    - What approach participants believe they take when creating a coded solution.
- **Follow up meeting protocol:** this was originally optional but was changed to immediate: participants are given feedback immediately during the debriefing portion of the 40-60 minute session.
- **Making the second Code Puzzle easier:** the content of the Java foundations and programming courses in the first year undergraduate CS program changed after the study commenced and it was determined that the second code puzzle may be perceived as more difficult now than it was previously. As a result, the original second code puzzle remains in place for participants who complete the primary two puzzles confidently and wish to be given a harder puzzle; another, easier, puzzle was developed to replace it as one of the two core puzzles.

Points to Note *(the following points reflect behind-the-scenes changes to the analysis of the study):*

- **Study Focus:** the focus has changed slightly from testing an algorithm to testing whether code puzzles are an effective way to gauge understanding (final study). This focus in no way impacted how the study was run and the participation experience.
- **Data Analysis:** the way in which the data is analysed is different as a mathematical formula was not shown to be effective due to the discovery of the 'workspace' phenomenon (where participants used a self-created central area to reorganise their thoughts instead of the standard interface restricted two column approach); instead, the same data is collected but it is now analysed via the use of axial coding and Grounded Theory in an attempt to unbiasedly form a theory about the person's understanding. The consent form has been adjusted to reflect this albeit it is of no direct consequence to the participant (see Appendix).

Any other relevant information

Applicants may indicate any specific ethical issues relating to the amendment, on which the opinion of the REC is sought.

Change from five week data withdraw policy to two weeks: participants will have less time to withdraw their data – from 5 weeks to 2 weeks. This is due to the practicalities of completing and writing up the PhD thesis.

List of enclosed documents (see Appendix)

Appendix Number	***Document***	***Version***	***Date***
A-1	Background Questionnaire	3.0	03/03/2018
A-2	Pre-Code Puzzle Questionnaire	1.0	02/10/2017
A-3	Post-Code Puzzle Questionnaire	2.0	12/02/2017
A-4	Post-Experiment Questionnaire	2.0	02/10/2017
A-5	Consent Form	6.5	04/07/2019
A-6	Participant Information Sheet	2.0	04/07/2019
A-7	CP1	2.0	24/03/2020
A-8	CP2	2.0	24/03/2020
A-9	Debriefing Document	2.0	04/05/2018
A-10	Original Ethics Submission	1.0	23/02/2017
A-11	Modified Ethics Submission	7.0	12/02/2020
A-12	Blackboard Recruitment Announcement	2.0	04/03/2020

Declaration

- I confirm that the information in this form is accurate to the best of my knowledge and I take full responsibility for it.
- I consider that it would be reasonable for the proposed amendment to be implemented.

Signature of Chief Investigator:

[REDACTED]

Print name: [REDACTED]

Date of submission: [REDACTED]