FLEXSIG -

FLEXIBLE SOFTWARE INSPECTION GROUPWARE

SHAMSUL BIN SAHIBUDDIN

Doctor of Philosophy

ASTON UNIVERSITY

February 1999

ASTON UNIVERSITY

FLEXSIG -
FLEXIBLE SOFTWARE INSPECTION GROUPWARE

SHAMSUL BIN SAHIBUDDIN

Doctor of Philosophy

1999

SUMMARY

The objective of this research is to design and build a groupware system which will allow members of a distributed group more flexibility in performing software inspection.

Software inspection, which is part of non-execution based testing in software development, is a group activity. The groupware system aims to provide a system that will improve acceptability of groupware and improve software quality by providing a software inspection tool that is flexible and adaptable.

The groupware system provide a flexible structure for software inspection meetings. The groupware system will extend the structure of the software inspection meeting itself, allowing software inspection meetings to use all four quadrant of the space-time matrix: face-to-face, distributed synchronous, distributed asynchronous, and same place-different time. This will open up new working possibilities. The flexibility and adaptability of the system allows work to switch rapidly between synchronous and asynchronous interaction.

A model for a flexible groupware system was developed. The model was developed based on review of the literature and questionnaires. A prototype based on the model was built using java and WWW technology. To test the effectiveness of the system, an evaluation was conducted. Questionnaires was used to gather response from the users.

The evaluations ascertained that the model developed is flexible and adaptable to the different working modes, and the system is capable of supporting several different models of the software inspection process.

Keywords : Software Engineering, Computer Supported Cooperative Work, World Wide Web, Java, CASE

To my Parents,

Hajjah Alimah Idros &

Haji Sahibuddin bin Haji Muhammad

To my Wife,

Sarimah bt Shamsudin

To my Children,

Khairul 'Abid bin Shamsul &

Muhammad 'Izzuddin bin Shamsul

... and in fond memory of my late ....

'Nenek' &

Shaiful Rizal

ACKNOWLEDGEMENTS

Alhamdulil-laahirab-bil 'alamiin

I would like to thank my supervisor Mr Bernard S. Doherty for his invaluable advice and assistance throughout the course of this research.

My thanks also goes to the folks in Room 268, Jai, Vasilis, and Zul, and to the support staff, Dr. Tony, Patel, and Neil, for their support.

I would also wish to acknowledge the University Technology of Malaysia and the Public Service Department of Malaysia for providing the financial support for this research.

And lastly, I would like to acknowledge Mr. Jeff Breidenbach, for the code on which Chat and the Pop-up Message is based.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.0 INTRODUCTION

This thesis describes the development and evaluation of a model and a prototype implementation for a flexible distributed software inspection groupware system. The model both combines and extends previous software inspection models. The prototype system is based on the model. The prototype is built from existing software and communication tools and technology to provide a vehicle for testing the model and its implementation. The system was evaluated and the results analysed.

In this first chapter, the research and areas related to it are introduced.

- **Section 1.1** will briefly outline the overall structure and organisation of this thesis.
- **Section 1.2** is an introduction to software engineering (SE) and computer supported cooperative work (CSCW), which span the main areas of this research.
- **Section 1.3** introduces the justification of this research
- **Section 1.4** gives an introduction to four areas of technology used in this research. They are: the world wide web (WWW), java technology (Java), the object oriented paradigm (OOP) and hypertext.

1.1 ABOUT THIS THESIS

This thesis is organised into nine chapters. This first chapter covers the overall introduction to the research and areas related to it. The rest of the thesis is organised as follows:

Chapter 2 gives a detailed review of the current state of research in software engineering especially in software quality. A detailed discussion of issues in software inspection is given.

Chapter 3 provides a review of reported work in computer supported co-operative work. The issue of software engineering, in general, and software quality, in particular, in relation with computer supported cooperative work is discussed.

Chapter 4 focuses on the problems this research addresses and describes the methodology used to solve the problems. Detailed research design and procedures are presented. This includes the purpose and the importance of the study, research methodology, and methods to be used in testing the model and the prototype.

Chapter 5 discusses the model of the process. The model developed is based on an extension and an integration of several existing models and systems, and also based from a questionnaire. Models from other works within this research area are also described.

Chapter 6 details the implementation of the prototype of the model. It includes a review on the supporting technologies: the world wide web, java technology, the object oriented paradigm, and hypertext. The architecture and functions of various modules of the prototype systems are described.

Chapter 7 describes the evaluation of the prototype. Methods used in the evaluation phase are given here.

Chapter 8 discusses the research finding. The results of the evaluation are presented here. Data gathered before and after the evaluation of the prototype are compared. The chapter also presents the system's limitations and the critique on the model and the prototype.

Chapter 9 concludes the works of this thesis. Suggestions are given for future development of this thesis.

1.2 BACKGROUND OF THE RESEARCH

As computers have become smaller, cheaper, and numerous, people have become more and more interested in connecting them together to form networks and distributed systems. The merging of computers and communications has had profound influences on the way computer systems are organised. Tanenbaum (1981) said that:

> *"The concept of the 'computer centre' as a room with a large computer to which users bring their work for processing is rapidly becoming obsolete. This model has not one, but two flaws in it: the concept of a single large computer doing all the work, and the idea of users bringing work to the computer instead of bringing the computer to the users."*

This old model is now replaced by one in which a large number of separate interconnected computers do the job.

In the eighties, the massive rise in the number of personal computers was followed by the trend to network these machine together. This has opened

the possibilities of individuals working together as groups via such networks. To allow this group working, computer communication networks should have the following goals:

- To make all programs, data, and other resources available to anyone on the network without regard to the physical location of the resource and the user.
- To provide interprocess communication, such as among users and processors.
- To provide distribution of processing functions.

For group working to be supported, suitable software is necessary. Most software systems only support interaction between a user and the system. Even systems designed for multiple users provide minimal support for interaction between users (Ellis *et al.*, 1991). Additional software is needed to support interaction between group members. Since a significant portion of a person's activities occur in a group, co-operative working support is clearly needed (Ellis *et al.*, 1991). To support group interaction, attention must be paid to three areas: communication, collaboration and co-ordination. Early software systems tended to support part of these group functions, but not all.

1.2.1 Software Engineering

Jackson (1985) describes software development as engineering because it is concerned to make useful physical devices to serve practical purposes in the real world. Software is a description of a machine. Jackson (1985) further describes the relationship between the world and the machine by dividing it into four facets. The four facets are: the modelling facet, where the machine simulates the world; the interface facet, where the world touches the machine physi-

cally; the engineering facet, where the machine acts as an engine of control over the behaviour of the world; and the problem facet, where the shape of the world and of the problem influences the shape of the machine and of the solution. Brooks (1986) said that the software project has something of the werewolf's character, usually innocent and straightforward, but capable of becoming a monster of missed schedules, exceeded budget, and flawed products. Ohno (1989) also stated that it is not an overstatement that the bottleneck of future society is software technology.

Brooks (1979) wrote that in programming, one must perform perfectly. Human beings are not accustomed to being perfect, and few areas of human activity demand it. Brooks (1979) concluded his book by saying:

> *"One can expect the human race to continue attempting systems just within or just beyond our reach; and software systems are perhaps the most intricate and complex of man's handiwork. The management of this complex craft will demand our best use of new languages and systems, our best adaptation of proven engineering management methods, liberal doses of common sense, and God-given humility to recognise our fallibility and limitations."*

The key for the problem that was laid down before is an engineering approach to the development of software, coupled with continuing improvement of techniques and tools (Pressman, 1992). By combining comprehensive methods for all phases in software development, better tools for automating these methods, more powerful building blocks for software implementation, better techniques for software quality assurance, and an overriding philosophy for coordinations, control, and management, we can achieve a discipline for software development - a discipline called software engineering (Pressman, 1992).

Definition of Software Engineering

Software engineering was defined by Fritz Bauer in 1968 at a conference sponsored by the Science Committee of the North Atlantic Treaty Organization (NATO) (Bauer, 1976). Bauer defined the term as the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines (Bauer, 1976; Pressman, 1992). The IEEE (1983) defines software engineering as the systematic approach to the development, operation, maintenance, and retirement of software.

Boehm (1976) defines software engineering as the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them. Fairley (1985) defines software engineering as the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates. Fairley (1985) further stated that the primary goals of software engineering are to improve the quality of software products and to increase the productivity and job satisfaction of software engineers.

Software development processes consist of four components: the study process, design process, implementation process, and maintenance process (Sodhi, 1991). Sodhi characterised the components of software development by differences in their purpose and in the nature of constituent activities to achieve these purposes. Meanwhile, Schah (1993) divides the software process as he calls it, into eight phases: requirements, specification, planning, design, implementation, integration, maintenance, and retirement. There are a number of software life-cycle models in use nowadays. Some of the popular models are: the waterfall model, the rapid prototyping model, the incremental model, and the spiral model.

There are also a number of software process models available. The principal approaches for software development are: structured approach, object-oriented approach, entity related approach, event-oriented approach, and step-wise refined approach.

Introduction to Software Quality

The primary goals of software engineering are to improve the quality of software products and to increase the productivity and job satisfaction of software engineers (Fairley, 1985). Quality can be defined as the adherence to some agreed specified performance. Pressman (1992) defines quality as conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software. Yasuda (1989) defines quality as the degree of user satisfaction. Definition of quality as meeting a national or an in-house standard alone is not sufficient. User satisfaction depends on two conditions: the design specifications must meet the user's needs; and the program must function as described in the design specifications (Yasuda, 1989). Analysing and removing identified errors in the software area provide the basis for improving software quality (Schulmeyer, 1990).

Every software product should posses the following quality attributes: usefulness, clarity, reliability, efficiency, and cost-effectiveness (Fairley, 1985). To achieve quality in a software product, there is a need to have a proper management system. Daily (1992) proposed a supplier Quality System which includes the element of Quality Control, Quality Assurance, and Quality Management. Quality Control (QC) checks for errors and problems in the work. Quality Assurance (QA) assesses independently the work done and the way it is done. Quality Management (QM) is the organisational framework and asso-

ciated responsibilities for quality. In QC, techniques such as inspection, review and testing are part of the process.

1.2.2 Computer Supported Cooperative Work

Possibilities of computer support for groups in face-to-face meetings or through a network have been investigated by researchers working in decision support systems, co-ordination systems and office automation procedures. Researchers in a range of disciplines, for example, sociology, anthropology, psychology and linguistics had been trying to relate their area to work in the computer field oriented towards working in groups. All these developments had occurred independently.

In 1986, all these works from various disciplines were brought together under an international conference called CSCW '86 in Texas, USA, with the aim of discussing human group working preferences and characteristics, and exploring how computers can support them. CSCW has expanded rapidly since the first conference. The field has attracted different kinds of organisations for various reasons. Computer supplier organisations were attracted because of the new market. It attracted academics because of the opportunity to build usable systems based on a thorough understanding of human individuals and groups. It also attracted user organisations because operating effectiveness depends on good interworking between employees. Some of the products based on CSCW that are presently available are electronic mail systems, computer conferencing systems, co-authoring systems, group decision support systems, and team development and management tools.

Definition of CSCW

CSCW was first coined by Irene Greif of Massachusetts Institute of Technology and Paul Cashman of Digital Equipment Corporation in 1984 (Greif, 1988). They did not intend any special emphasis to the meaning of the individual words within it but simply wanted a shorthand way of referring to a set of concerns about supporting multiple individuals working together with computer systems (Bannon & Schmidt, 1989). Wilson (1990, 1991) defines CSCW as a generic term that combines the understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services and techniques. Wilson's definition embraced other term including Groupware and Workgroup Computing.

Classification of CSCW

Wilson (1990, 1991) divides CSCW into two major area of concerns: the group working process and the enabling technology to support it. The group working process is further subdivided into four areas: individual aspects, organisations aspects, group working design aspects and group dynamics aspects. The enabling technologies area is subdivided into four areas. These four areas are communication systems, shared work space systems, shared information systems and group activity support systems.

Group Working Process

According to Wilson (1990, 1991), the group working process areas are individual aspects, organisations aspects, group working design aspects and group dynamics aspects.

The *individual aspects* of the group need to be understood because the characteristics, skills, knowledge and artefacts that individuals bring will have a crucial impact on the group effectiveness. Research on human communication characteristics, individual work patterns and interface design for group support systems is being done with respect to the individual aspects of the group. Human communication research investigates how humans "talk" with each other. Individual work patterns need to be understood because individual habits show through, and need to be taken into consideration in designing group support tools. Interface design for group support systems depends on knowledge of human perception and cognitive psychology.

The *organisational aspects* concern the following areas: representation of organisational knowledge, organisation design, and management issues. In the area of representation of organisational knowledge, the problem of too much organisational information and its frequent changes have been investigated. Organisation design deals with the problem of creating tools to support organisational changes. Meanwhile, in management issues, new support tools will create new management requirements on activities, people and resources.

In the *group work design aspects*, research is being done in user involvement, prototyping and usability testing and group work design procedures. Investigation into user involvement in the work group design process is needed because it will create more positive attitudes and make the system work. The designs then need to be tested because human interaction within groups is complex and unpredictable. The requirement for clear guidelines for the analysis and design of CSCW systems resulted in research being done in group work design procedures.

Research in *group dynamics aspects* deals with the way individuals behave within groups and the way groups perform. Work in this area covers the collaborative process, group performance and group behaviour. The collaborative process needs to be understood because it contributes to the design of group work tools. How CSCW systems affect group performance is investigated because this will indicate its effectiveness. Research on group behaviour is related to group performance and is done in real-world conditions and in studies of groups using prototypes.

Enabling Technologies

The enabling technologies areas are communication systems, shared work space systems, shared information systems and group activity support systems (Wilson, 1990, 1991).

Examples of the *communications systems* area that are being investigated are advanced electronic mail systems, X.500 electronic mail directories of group and organisational information, real-time desktop video conferencing and large video wall screen systems. Research on how to provide support so that informal communication can continue is the main aim of this area.

Research in the *shared work space systems* area looks into the role of shared work spaces in group work and builds supporting tools for the process. Shared work spaces and the material created on them is often used as the basis for further discussion. Example of development in this area is remote screen sharing facilities.

Shared information systems manage common and shared information. People working together start from a base of common informa-

tion. As their work progresses, people generate shared information. In this area, facilities which can support input, storage, navigation and retrieval of that information by all members of the group are being investigated. Examples of shared information systems are multi-user hypertext and multi-user databases.

Work in *group activity support systems* includes procedure processing or work flow systems; activity processing which allows a more general form of work flow processing; methodologies and support tools for groups to analyse, define and prototype the organisations; procedures and equipment to carry out a group activity; specialised coordination, procedure processing and activity processing systems to support systems development process; co-authoring tools to support joint writing of documents and many more. Group activity support systems must be able to meet the needs of groups to establish goals, procedures and making it visible and carrying it out.

1.2.3 Software Development as Group Activity

The software development process is a group activity. It is a complex system which requires collaboration of many different types of specialists working together over substantial periods of time to create information-intensive products (Vessey & Sravanapudi, 1995). Projects ranging from two to fifty people are fairly typical in software development.

A more detailed view of the software life-cycle suggests that there are twelve activities, namely, system initiation, requirement analysis and specification, functional specification, prototyping and simulation, partition and selection, architectural configuration specification, detailed component design specification, component implementation and debugging, software integration and

testing, documentation revision and system delivery, training and use, software maintenance, and software retirement (Vessey & Sravanapudi, 1995).

Each activity outlined above will require a different type of requirement from a system that is providing it with group support. Document revision will require document co-authoring support. Training and use may require desktop video conferencing. Requirement analysis may require on-line meeting with document co-authoring support. However, Computer Aided Software Engineering (CASE) tools currently on the market are essentially for the single user (Vessey & Sravanapudi, 1995). There are only a handful of products that support group activities for certain activities in the software life-cycle (Vessey & Sravanapudi, 1995).

1.3 JUSTIFICATION OF THE RESEARCH

This research is interested in the software inspection process aspect of software engineering. The aim was to develop and evaluate a model and a prototype implementation for a flexible distributed software inspection groupware system. The model is both a combination and an extension of previous software inspection models. The prototype system was based on the model. The prototype utilised existing tools and technology to achieve the flexibility outlined in the model developed. The system was evaluated and the results analysed.

The aim of inspection is to analyse a product or document to detect defects (Fagan, 1976). Inspection processes also impose discipline, provide measurable feedback, and provide a way to educate a team (Grady, 1993). The potential saving from finding errors early in the life cycle are considerable (Daily, 1992). However, a manual software inspection process is difficult and time consuming. A formal meeting must be arranged where everybody must be in

one room to discuss and inspect the document. Normally pen and paper are used to record the errors detected. If a groupware tool can be provided to support this activity, the difficulties in the software inspection process can be reduced. Among the tools that exist, ICICLE (Brothers *et al.*, 1990) is meant to be used in the same room face-to-face; InspeQ (Knight & Myers, 1991, 1993) only supports single inspection or the individual phases of multiple inspector inspection; CSRS (Johnson & Tjahjono, 1993) supports distributed asynchronous inspection; CSI (Mashayekhi *et al.*, 1993) only supports distributed synchronous process; CAIS (Mashayekhi *et al.*, 1994), like CSRS, supports distributed asynchronous; AISA (Stein *et al.*, 1997) supports distributed asynchronous inspection on the Web; and finally, Scrutiny (Gintel *et al.*, 1993) supports distributed synchronous inspection. None of the tools support both the distributed synchronous and the distributed asynchronous mode at the same time.

In designing a software inspection model and groupware system, there are a number of points that need to be considered. The groupware must be flexible in order to support the asynchronous and the synchronous mode of the distributed meeting. The tool to support group working in software inspection must also be flexible in supporting different implementations of the software inspection process based on different model. In order to be flexible across computer networks, the system must be able to be executed on different computing platforms and provide a consistent user interface.

Under the classification proposed by Wilson (1990, 1991), this research falls into the category of the group work design aspects of a group working process and also touches on the aspect of shared information systems and shared work space systems under the area of enabling technologies.

1.4 IMPLEMENTATION ISSUES

To implement the flexible software inspection groupware prototype, a number of technologies are needed. Four other areas that contribute to this research are the world wide web (WWW), Java technology, object oriented paradigm (OOP) and hypertext. All four areas will provide technology to enable this research to accomplish its goal. The following sections present a brief introduction to all four areas. A more detail discussion on these topics is in Chapter Four.

1.4.1 The World Wide Web

In this era of information technology or IT, the world wide web (WWW, Web, W3) has become one of the most essential tools for the dissemination of global information and for global communication. It was introduced by CERN (Centre European pour la Recherche Nucleaire) in January, 1992 (Berners-Lee *et al.*, 1992; Tatters, 1996). More and more people are being connected to the Internet and are using it to communicate. In addition, they are also utilising the technology for searching various information by 'surfing' the relevant 'Web site' and then locating the desired information on the 'Web pages' of the appropriate 'home page address'.

Tim Berners-Lee refers to the WWW as a distributed heterogeneous collaborative multimedia information system (Berners-Lee *et al.*, 1992). However, nowadays, it is defined as a graphical hypertext-based information system that provides access to a multitude of Internet resources (Tatters, 1996). In Horton *et al.* (1996) the WWW is considered as the collective name for all the computer files in the world that are accessible through the Internet, electronically linked together, usually by 'tags' expressed in HTML and viewed, ex-

pressed or retrieved through a 'browser' program running on the computer. Thus, the WWW is an easy way to access the interconnected web of computers and information that can be accessed by just about anyone with an Internet connection (Tatters, 1996) and a 'Web browser'.

1.4.2 Java Technology

The second technology that is important to this research is Java. Java was introduced in 1995. It started as a research project to developed advanced software for a wide variety of network devices and embedded system (Sun, 1995). Java was based on C++ but without its negative features, and new principles and structure were inherited from other object-oriented language, such as Eiffel, SmallTalk, Objective C, and Cedar/Mesa (Sun, 1995). Among the features that are of interest to this research are its security aspects and the fact that it is architecture neutral and portable.

1.4.3 Object-Oriented Paradigm

Object-orientation is one of the methods used in computer software system development. It is a system development abstraction which addresses some of the problems the software industries have with today's system development methods. Object-orientation encourages indivisible packaging or binding of data, process and state into problem-related, reusable, maintainable, and Integrated-Circuit-like items called objects (Cox, 1986). This is known as encapsulation in OOP. Object-orientation also encourages incremental system specification and enhancement using a base-displacement thought process. This feature is called inheritance. Some of the key benefits claimed by object-

orientation are reusability, productivity, flexibility, maintainability, and quality of the software (Huff, 1993).

1.4.4 Hypertext

Hypertext consists of nodes of information and links between them. The concept was first mooted by Vannevar Bush in 1945 (Bush, 1945) and the term first coined by Theodor Nelson in 1965 (Nelson, 1981). In hypertext, the links between the nodes are machine-supported and the movement between two nodes takes place automatically. Horn (1989) listed three characteristics of hypertext that make it distinctive from just another text database. These features are:

- the ability of the user to conduct rapid browsing and navigation
- the ability to have information partially or non-linearly structured
- the ability to personalise hypertext and in some cases to make it a read-only hypertext.

According to Nelson (1981), hypertext is the associative links between nodes, which are one or many screens of information. Alternatively, hypertext is also commonly defined as a database that has active cross-references and allows the reader to jump to other parts of the database as desired (Shneiderman & Kearsley, 1989). In other words, hypertext would enable the reader or learner to view information or learning material in a nonlinear (Shneiderman & Kearsley, 1989) or nonsequential (Nielson, 1990) manner. Nevertheless, this feature is subjected to the links that the hypertext provides for the user. It has been suggested that this feature is to mimic the associative networks of human memory (Jonassen, 1990).

1.5 SUMMARY

In this chapter, an introduction to two main areas of this research - Software Engineering and Computer Supported Cooperative Work - has been provided. An introduction and definition of Software Engineering has been given, including the area of software quality.

A definition of CSCW following Wilson, has been given. The idea of software development as a group activity was presented. The aim of this research was presented, which is to develop and evaluate a model and a prototype for a flexible distributed software inspection groupware.

The enabling technologies: the world wide web, java technology, the object oriented paradigm and hypertext, have been outlined.

Subsequent chapters will provide a detailed literature review. The next chapter presents a detailed review on software quality.

Chapter 2

Review of Literature on Software Quality

2.0 Introduction

The review of literature is divided into three sections. The first section, in this chapter, deals with software quality (SQ). The second section, in the next chapter, covers computer supported cooperative work (CSCW). The third section, in Chapter Six, deals with the enabling technology that will be used in this research: the world wide web (WWW), java technology (Java), the object oriented paradigm (OOP) and hypertext.

In this chapter, topics covered are as follows:

- **Section 2.1** topics related to software engineering are reviewed, particularly definitions of software engineering, software development process models, software engineering management, and case tools.
- **Section 2.2** presents the definition of software quality. Verification and validation and software testing are covered.
- **Section 2.3** introduces the topic of software inspection. Explanation on different versions of the inspection process is given.

2.1 SOFTWARE ENGINEERING

Software has now surpassed hardware as the key to the success of many computer-based products (Pressman, 1992). Software is the factor that differentiates between the usage of the computer, whether it is used for business, education, government or military. The full realisation of the potential of computers depends on the software (Conte *et al.*, 1986). At the dawn of programming, it was viewed by many as an art form with few formal methods available. Software cost was much smaller than the hardware cost at that time.

That was then. Today, the cost of software is usually one of the largest budget items in any system development. The cost of hardware to perform a given function continues to decrease sharply, while the cost of software required for that function has continued to increase (Conte *et al.*, 1986). The fact that the software technology lags behind hardware technology, and its ballooning cost, creates a great concern. As the end of the nineties comes closer, software problems continue to grow. Pressman (1992) stated:

- that hardware sophistication has outpaced our ability to build software to fully realise the hardware's potential
- the demand for new programs continues to exceed our ability to build new programs
- poor design and inadequate resources threatened our ability to maintain existing programs.

The software crisis is partly the result of poor management of the software development process (Pressman, 1992). Software engineering practice is the response to this crisis. Software engineering was first defined by Fritz Bauer in 1968 at a conference sponsored by the Science Committee of the North

Atlantic Treaty Organization (NATO) (Bauer, 1976). Bauer (1976) defined software engineering as:

> *"the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."*

Besides the original definition of software engineering, there exist other definitions. Boehm (1976) defines software engineering as:

> *"the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them."*

Meanwhile, Fairley (1985) defines software engineering as:

> *"the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates."*

Fairley further states that the primary goals of software engineering are to improve the quality of software products and to increase the productivity and job satisfaction of software engineers.

The common theme in all the definitions above is the requirement of engineering discipline in software development. As Jackson (1985) stated, software development is engineering because it is concerned to make useful physical devices to serve practical purposes in the world. Fairley's definition off software engineering is the most comprehensive of all three. It touches the issue of software quality explicitly. Any reference or discussion on software engineering later on will be based on Fairley's definition.

2.1.1 Software Engineering Methodology

This section looks at the software development phases in general terms. The generic view of software engineering is independent of the software development process model used. Software engineering consists of a set of structured methods, procedures, and tools used to engineer quality and cost-effective software (Sodhi, 1991).

Software development processes consist of four components: the study process, design process, implementation process, and maintenance process (Sodhi, 1991). The study process in turn comprises requirement determination, requirement specification and requirement analysis. The implementation process comprises coding and testing. Sodhi characterised the component of software development by differences in their purpose and in the nature of constituent activities to achieve these purposes. Sodhi makes a distinction between the software development process and software engineering development phases. Chapter one discussed the twelve activities of software engineering development phases or software life-cycle that Sodhi (1991) presented.

Meanwhile, Schah (1993) divides the software process into 8 phases: the requirements phase, specification phase, planning phase, design phase, implementation phase, integration phase, maintenance phase, and retirement. Schah (1993) described the implementation phase as the phase where the various components are coded and tested.

Pressman (1992) divides the software development process into three generic phases. The three phases are as follows: definition, development, and maintenance. The definition phase comprises of systems analysis, software project planning, and requirements analysis. The development phase consist of software design, coding, and software testing. And the maintenance phase embodies correction, adaptation, and enhancement.

There are a number of software process models in use nowadays. Some of the popular models are: the waterfall model, the rapid prototyping model, the incremental model, and the spiral model (Boehm, 1976, 1988; Sodhi, 1991; Pressman, 1992; Schah, 1993). A more detailed discussion on these software process models is in the next sub-section.

There are also a number of software development approaches available. The principal approaches for software development are: the structured approach, object-oriented approach, entity related approach, event-oriented approach, and stepwise refinement approach (Sodhi, 1991; Pressman, 1992; Schah, 1993).

2.1.2 Software Process Model

The primary focus of software methodology is on how to navigate through each phase and how to represent the product of each phase (Boehm, 1988). Boehm (1988) further stated that the software process model differs from software engineering methodology because it addresses the question of what shall we do next and how long shall we continue to do it. The software life cycle or the software process model is important because it determines the order of the stages involved in software development and evolution and establishes the transition criteria for progressing from one stage to the next (Boehm, 1988). These include completion criteria for the current stage plus choice criteria and entrance criteria for the stage. The life cycle encompasses many concepts throughout software engineering (Sodhi, 1991).

The waterfall model was introduced by W. Royce in 1970 (Royce, 1970). The life cycle proceeds in an orderly sequence of transition from one phase to the next (see *Figure 2 - 1*). Each phase is conceived in terms of input, process, and output. A critical point regarding the waterfall model is that no phase is

complete until the documentation for that phase has been completed, and the products of that phase have been approved by the software quality assurance group (Schah, 1993). The waterfall model is modelled after the conventional engineering cycle (Pressman, 1992).

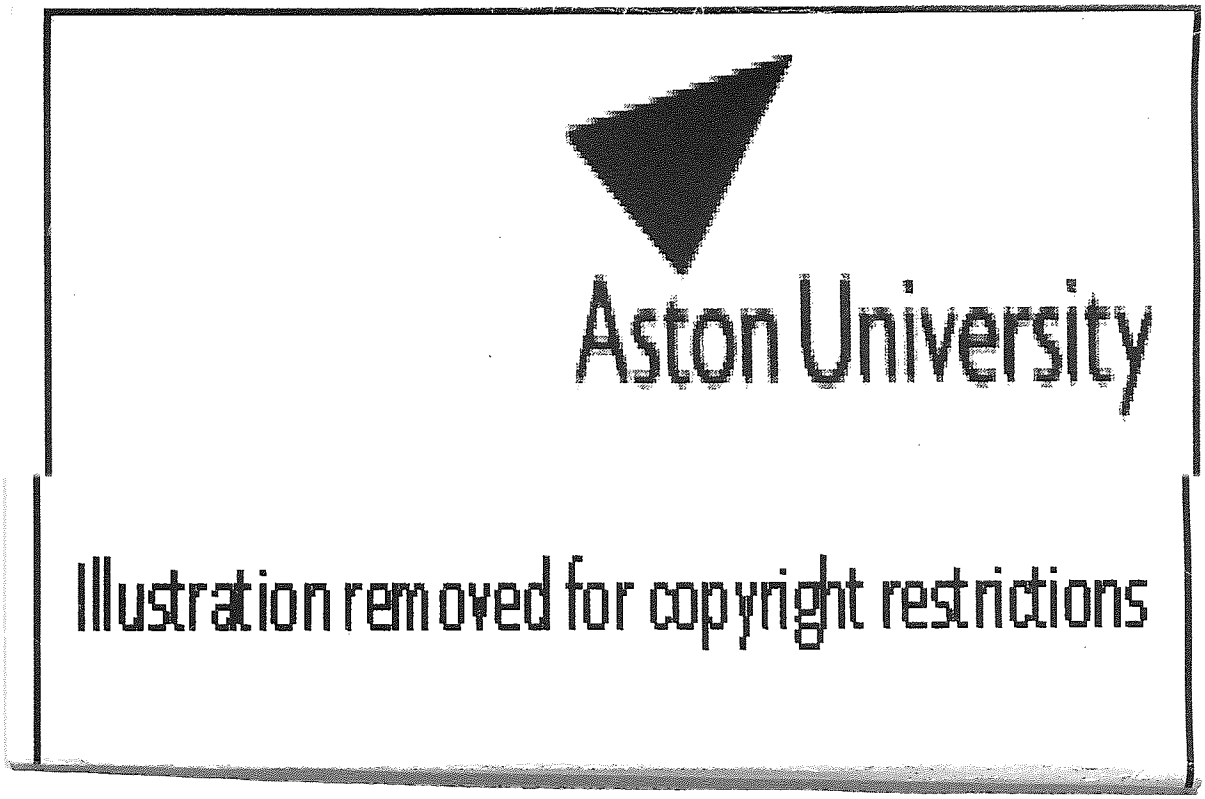

Figure 2 - 1 : *The Classic Waterfall Model of Software Life Cycle*

In the rapid prototyping model of software process, components representing the eventual system is constructed early (Sodhi, 1991; Pressman, 1992). A rapid prototype is a working model that is functionally equivalent to a subset of the product (Schah, 1993). The first step in the rapid prototyping model is to build a rapid prototype with the intent of providing an opportunity for feedback from the user. The developers can draw up the specification document based on the feedback.

In the spiral model of software development and enhancement, the model involves multiple iterations through cycles that analyse the results of prior phases by determining risk estimates for future phases (Boehm, 1988; Sodhi, 1991). The model was developed by Boehm (1988). At each phase, alternatives are evaluated with respect to the objectives and constraints that form the basis for the next cycle of the spiral. Each cycle is completed with a review process which includes interested parties (Sodhi, 1991). The radial dimension represents cumulative cost to date, the angular dimension represents progress through the spiral (see *Figure 2 - 2*) (Boehm, 1988). The basic premise of the model is that a certain sequence of steps is repeated while developing or maintaining the software.

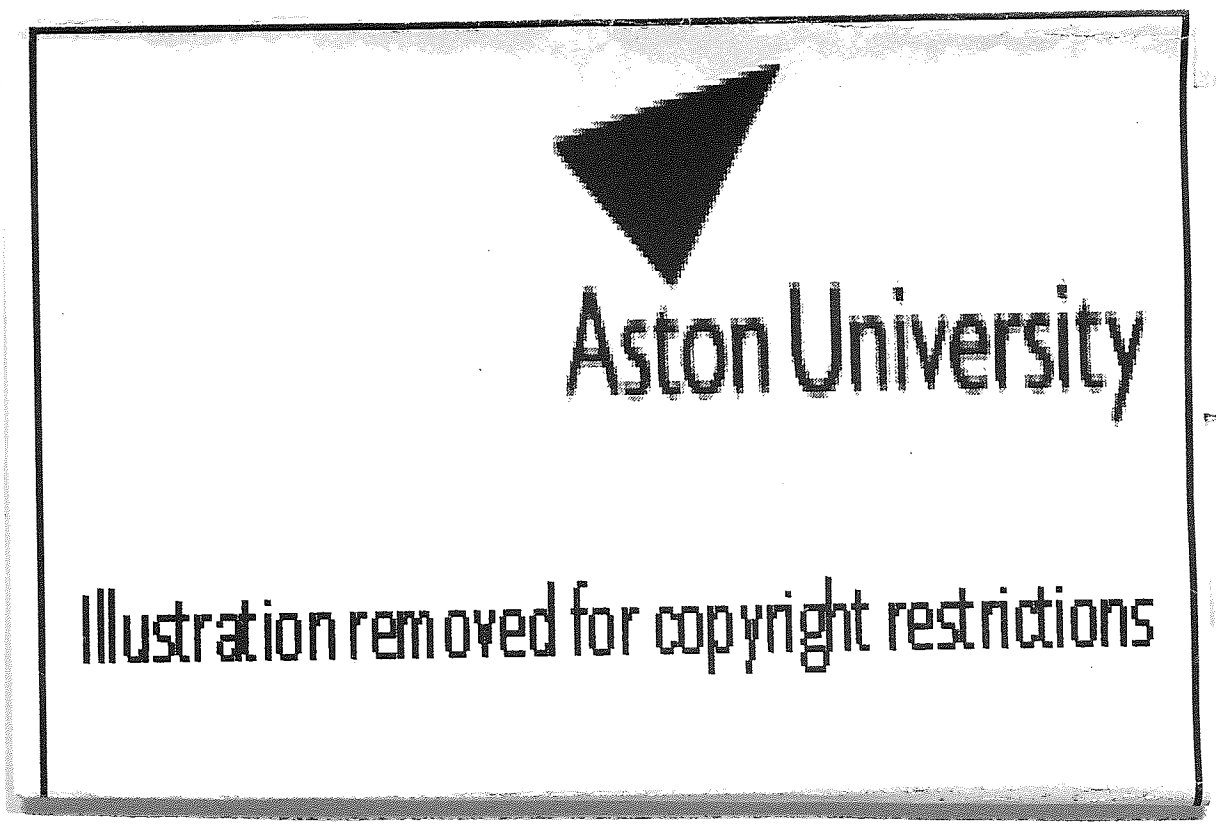

Figure 2 - 2 : *The Spiral Model of Software Development*

The incremental model exploits the fact that in practice software is engineered incrementally. An initial subset of the system is fully developed. More elaborate versions are built on the previous steps (Sodhi, 1991). The product is designed, implemented, integrated, and tested as a series of incremental builds, where a build consists of code pieces from various modules that interact together to provide a specific functional capability (Schah, 1993).

2.1.3 Software Engineering Management

There are two types of aims in software engineering management, one is the technical aim and the other is the corporate financial aim (Sneed, 1989). The technical aims of software management are:

- to increase software productivity
- to improve software quality.

Software productivity is measured in units produced. The productivity can be quantified by program statements, function points, components, data elements, decisions, or a combination of these elements (Sneed, 1989). The characteristics used to measure software quality are the following: reliability, security, efficiency, user friendliness, maintainability, adaptability, and portability. The characteristics must be weighted according to the user's criteria. This is because quality is relative to the user's expectation. Sneed (1989) further stated that the corporate financial aims are:

- the automation of all corporate applications
- machine storage of all corporate data.

In managing software, there is a need to understand software economy. Software economy means the largest quantity of software of the highest possi-

ble quality should be produced using the smallest amount of staff, hardware, and software in the shortest possible amount of time (Sneed, 1989). From this statement, it can be derived that software economy depends on five factors, namely: quantity, quality, time, costs, and productivity. Time is measured in months or years and cannot be influenced or increased. Costs are a measure of the use of staff, software, and hardware resources for a project's development. Quantity is the amount of software delivered to the user or the scope of the system. Quality is the worth of the software delivered to the user. And productivity is the relationship of quantity and quality to cost.

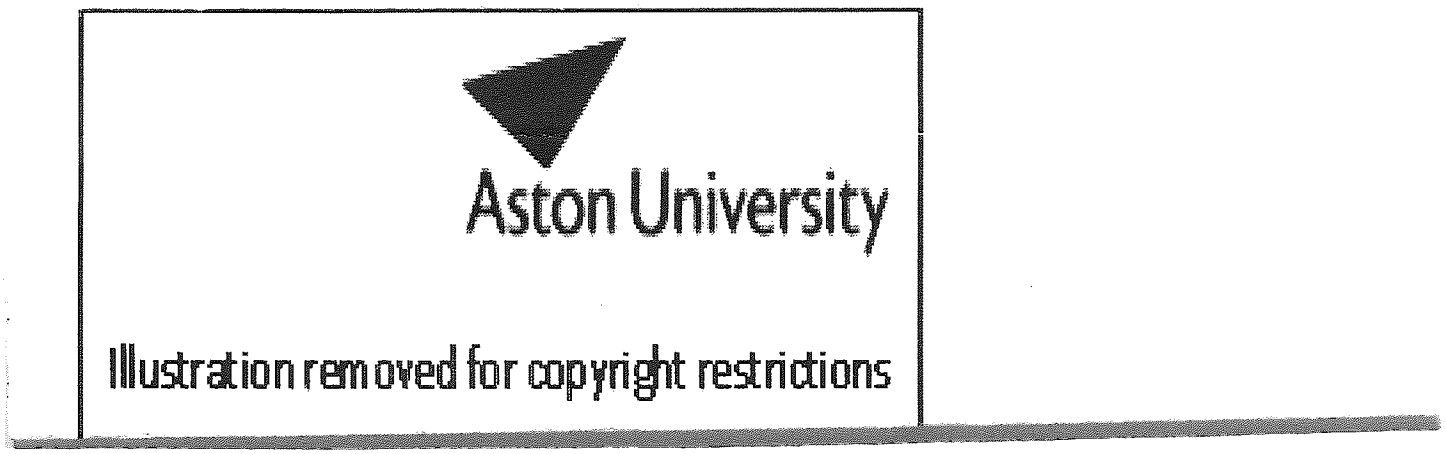

Figure 2 - 3 : *The 'devil's square' (Sneed, 1989)*

The basic model of software economics is a square of four corners: quantity, quality, time, and costs (see *Figure 2 - 3*). Sneed (1989) further elaborates that if time and costs are fixed, then quantity and quality depends on them. If quantity and quality are fixed, then time and costs depend on them. There is conflict between quantity and quality. The larger the scope of the software, the lower the quality, provided the productivity is fixed. A similar conflicts occur between time and cost. Productivity is always constant at any point in time

and can only be increased by improving: staff, computing capacity, and software tools (Sneed, 1989).

On top of the general management topic of software engineering, there exist several other sub-topics, namely: configuration management, risk management, and quality management. These three software engineering management sub-topics address important areas.

Configuration Management

Software work requires configuration management, which is the identification, control, and tracking of all items which constitute the software product and all versions of these items (Daily, 1992). Control management helps control changes and co-ordinate the work products of the many different people who work on a common software project (Humphrey, 1989). The goal of configuration management is to maximise productivity by minimising mistakes and confusions (Babich, 1986). The need for configuration management tends to become apparent only when the problem arises (Daily, 1992). In the real world, no software exists in only one version (Babich, 1986). Daily (1992) stated that configuration management allows projects to:

- uniquely identify every module, document and system created and released
- ensure the retention and safe custody of the items and control of the issue and distribution of copies
- identify, record, and control all changes to these items
- monitor the status and history of each item as it is developed, modified, and used.

Configuration management procedures provide the following set of control: version control, build control, change control, and status accounting and reporting (Daily, 1992).

- Version control identifies each item and records its history of development. This includes incorporation in successive builds, versions, and releases
- Build control controls the building of software component items
- Change control regulates the changes to items and to product releases. The completed set of products at the end of a project phase can be established. This is sometimes called a 'baseline'
- Status accounting and reporting is important so that the completeness and validity of a release can be checked and an audit conducted.

Appropriate tools are an important part of configuration management (Sneed, 1989). Configuration management tools may need to be large and complex to support the needs of large development (Daily, 1992). For a small team, manual configuration control procedures can be quite effective (Babich, 1986). The aims of a configuration management tool is to assist and automate the entry, access, and analysis of configuration information (Daily, 1992). Daily (1992) said that most of the tools are implemented around a database which has been adapted for the purpose. Most recent tools allows users to customise the tool to match their development process, instead of the other way around (King & Cohn, 1993).

Risk Management

Risk management is an informed decision making approach that involves consciously anticipating what could go wrong, assessing the potential loss, and incorporating this broadened perspective into program planning, activities, and

decision making (Higuera & Gluch, 1993). Higuera & Gluch (1993) presented six elements of risk management, they are:

- Identify - search for and locate risks before they become problem
- Analyse - process risk data into decision making information
- Plan - translate the information into decisions and actions
- Track - monitor the risk indicators and known risks
- Control - correct for deviations from the planned risk actions
- Communicate - provide feedforward and feedback data internal and external to the program on the risk activities, current risks, and emerging risks.

This paradigm is presented by Higuera & Gluch (1993) as a wheel to reflect that risk management is a continuous process. Risk management provides a means to achieve: building the 'right' product, building the product 'right', and delivering the product at the 'right' time (Higuera & Gluch, 1993).

Quality Management

It is comparatively recently that quality management has been accepted in the software industry (Daily, 1992). Some of the quality related practices like the inspections and reviews, compliance to procedures and standards, generation of project quality plans and assessment of quality systems have started to be accepted. Product quality is a key measure of the software process because it provides a clear record of development progress, a basis for setting objectives, and a framework for current action (Humphrey, 1989). Humphrey (1989) further stated that the basic principles of software quality management are much like those for cost management: set goals, make plans, track performance, and adjust the plan.

Daily (1992) proposed a quality management system which comprises quality control (QC), quality assurance (QA), and quality management (QM). Daily (1992) elaborated that quality control is the checking for errors and problems in the work, quality assurance is assessing independently the work done and the way it is done, and quality management is the organisational framework and associated responsibilities of quality. A quality management system must be defined against its own organisational objectives, assessed systematically and then corrected and improved where necessary (Daily, 1992). Daily (1992) further stated that a quality management system provides a point of reference for existing practice in quality related activities, such as quality control, testing, and quality assurance.

2.1.4 CASE and Other Tools

The software industry has searched for solutions to its systems development problems since the inception of computers. In 1980s, the search has focused on computer-assisted software engineering or CASE. Such tools aid systems developers in specifying, designing, and constructing software systems. The basic reason for using CASE systems is to improve the quality and productivity of the work (Humphrey, 1991). Humphrey (1991) stated that by reducing tasks to routine procedures and mechanising them, labour is saved and sources of human error are eliminated. The ultimate goal of CASE technology is to separate design from implementation (Fisher, 1991). One definition of computer-aided software engineering is:

> *"the use of tools that provide leverage at any point in the software development cycle"* (Fisher, 1991).

Schah (1993) stated that the simplest form of CASE is the software tool, which assists in just one aspect of the production of software. CASE tools that sup-

port during the earlier phases of the process are sometimes termed upperCASE or front-end tools, while those that assist with the later stage of the process are named lowerCASE or back-end tools (Schah, 1993). Hence, the CASE environment is a collection of tools that together support one or perhaps two phases of the software process (Schah, 1993).

Pressman (1992) divides CASE tools according to its function. The list given by Pressman (1992) is as follows:

- Business systems planning tools - these tools provide a 'metamodel' from which specific information systems are derived by modelling the strategic information requirements of an organisation
- Project management tools - such tool focus on one specific element of project management. This category can be divided further into: project planning tools, requirement tracing tools, and metric and management tools
- Support tools - this category comprises systems and application tools that complement the software engineering process. Support tools include: documentation tools, system software tools, quality assurance tools, and database and software configuration management tools
- Analysis and design tools - this class provides the means for the software engineer to create a model of the system to be built. Tools in this class consist of: structured analysis and structured design tools, prototyping and simulation tools, interface design and development tools, and analysis and design engines
- Programming tools - the tools in this group include: conventional coding tools, fourth-generation coding tools, and object oriented programming tools
- Integration and testing tools - this category comprises: static analysis tools, dynamic analysis tools, and test management tools

- Prototyping tools - any tool that supports prototyping can be called a prototyping tool. The range covers from the lowest end of the implementation scale, such as tools to create a 'paper prototype', progressing to the screen painters, prototype and simulation tools, CASE prototyping tools, and finally to CASE with code generations, which reside at the highest end of the scale
- Maintenance tools - this type of tools can be divided into the following: reverse engineering tools and re-engineering tools
- Framework tools - tools in this category provides database management, configuration management, and CASE tools integration capabilities.

On the other side, an integrated project-support environment or IPSE supports every phase of the process, not only technical tools, but also management information tools (Schah, 1993). Schah (1993) asserted that IPSE should support a variety of languages and techniques, meaning that an IPSE cannot be language-centred, structure-oriented, or method-based. Nevertheless, in practice there are number of IPSEs that are method-based, they support all phases and provide management information but enforce one particular method (Schah, 1993). Schah (1993) contend that IPSE may well provide the necessary support for developers of large scale software to produce high quality software efficiently, within time and cost constraints.

CASE tools currently on the market are designed for the single user (Vessey & Sravanapudi, 1995). However, complex software systems require the collaboration of many specialist working together over period of time to create information-intensive products. Vessey & Sravanapudi (1995) stated that CASE tools intended to support a work group in the system development process should address issues such as co-ordinating interdependent activities and project management.

2.2 SOFTWARE QUALITY

In Chapter One, an introduction to software quality was laid out. This section elaborates further on this topic. The definition of software quality includes discussion on quality assurance and quality control. Discussion on verification and validation, software testing, and software inspection is also covered.

From the discussion on software engineering in the preceding section, the primary goals of software engineering are to improve the quality of software products and to increase the productivity and job satisfaction of software engineers (Fairley, 1985). Yourdon (1996) stated that three main ingredients for delivering quality are people, technology, and processes. Quality cannot be relied on to come easily or cheaply for a product as complex as software (Daily, 1992). Furthermore nowadays, the challenges of writing high-quality software for client-server applications are bigger than for simple one-machine and one-user jobs (DeJesus, 1996). A survey by Standish Group reports that the typical client-server project is fifty percent over budget and fifty percent behind schedule (Yourdon, 1996). Knutson (1996) stated that quality must be paid attention to early in the software life cycle, and programmers must be closely involved in quality assurance during all phases of development. Dobbins & Buck (1987) emphasised that the closer to the front of the life cycle the failure removal activity is accomplished, the greater the impact and the more significant the cost reduction.

2.2.1 Definition of Software Quality

Two formal definitions of software quality were given in Chapter One. Pressman (1992) defines quality as conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and

implicit characteristics that are expected of all professionally developed software. Meanwhile Yasuda (1989) defines quality as the degree of user satisfaction. Schulmeyer (1987a) defines software quality as:

"the fitness for use of the total software product."

Smith & Wood (1989) classified the activities and methodologies which contribute to software quality into two. They are:

- quality activities which attempt to identify and remove errors
- specification and programming methods and tools which removed many of the traditional steps in software design.

The latter makes it highly unlikely that certain types of software error will occur. Analysing and removing identified errors in the software area provides the basis for improving software quality (Schulmeyer, 1990). Schulmeyer (1990) further wrote that some organisations spend half their software quality effort on error tracking and analysis. A software product can be defective in one of four ways: a design defect, a construction defect, an expressed warranty failure, or a failure to adequately warn the user. A number of defect removal activities as presented by Dunn (1987) are as follows: reviews of requirement specifications, reviews of design specification, code reviews, unit testing, integration testing, sub-system and system testing, and qualification testing. Arthur (1993) stated that Fujitsu currently experiences only ten defects per million lines using known software techniques. System testing has been dropped from their software process because the low level of defects has made testing so expensive in terms of per defect found (Arthur, 1993).

To achieve quality in software products, there is a need to have a proper management system. The control of the design and development efforts is a key factor affecting the quality of the resulting software (Strange, 1993).

Humphrey said that the quality of a software system is governed by the quality of the process used to develop it (Curtis, 1992). Daily (1992) proposed a supplier Quality System which includes the element of Quality Control, Quality Assurance, and Quality Management. Quality Control (QC) check for errors and problems in the work. Quality Assurance (QA) assesses independently the work done and the way it is done. Quality Management (QM) is the organisational framework and associated responsibilities for quality. In QC, techniques such as inspections, review and testing are part of the process (Daily, 1992). The detailed discussion on the topic of QM was in *Section 2.1.3*.

Quality Assurance

> *"The issues with software are not whether checks are needed, but who does them and how."*

This statement by Humphrey (1989) highlights the important of software quality assurance. Quality Assurance (QA) embraces the activities which check that the quality control process is taking place and the standards, methods and tools are adequate (Smith & Wood, 1989). QA assesses independently the work done and the way it is done. The roles of software quality assurance (SQA) range from acting as an extension of development for debugging software products, to development process definition and control (Hoffman, 1993). Similarly, Schah (1993) stated that SQA applies to every aspect of the software process. SQA responsibility is shouldered by many different constituencies in an organisation, namely: software engineers, project managers, customers, salespeople, and SQA team members (Pressman, 1992). The formal definition of software quality assurance as given by Schulmeyer (1987a) is:

> *"independent evaluation to assure fitness for use of the total software product."*

According to Humphrey (1989), it is possible that no SQA activity is needed in a very small organisation. This is because it is possible for software managers to monitor the work. But when the size of the organisation grows beyond a few dozen people, Humphrey (1989) contended that management must resort to SQA solution. There are three goals of SQA according to Humphrey (1989). The goals are:

- To improve software quality by appropriately monitoring both the software and the development process that produces it
- To ensure full compliance with the established standards and procedures for the software and the software process
- To ensure that any inadequacies in the product, the process, or the standards are brought to management's attention so these inadequacies can be fixed.

It is important to remember that the SQA organisation is not responsible for producing quality products because these are development jobs (Humphrey, 1989).

Pressman (1992) listed seven major activities of SQA, they are as follows:

- application of technical methods
- conduct of formal technical reviews
- software testing
- enforcing standards
- control of change
- measurement
- record keeping and reporting.

Likewise, the QA activities presented by Yasuda (1989) are nearly identical to those outlined by Pressman. Yasuda (1989) describes the SQA activities as the following: development plan audit, design review, document inspection, intermediate quality audit, product inspection, system inspection, and field activities. Pressman (1992) asserted that the central activity that accomplishes quality assessment is the formal technical review. Reviews have been found to be as effective as testing in uncovering defects in software (Pressman, 1992). Among the responsibilities that the SQA should be given is to participate as inspection moderators in design and code inspections (Humphrey, 1989).

Hoffman (1993) stated that there are five different types of software quality group (see *Table 2 – 1*).

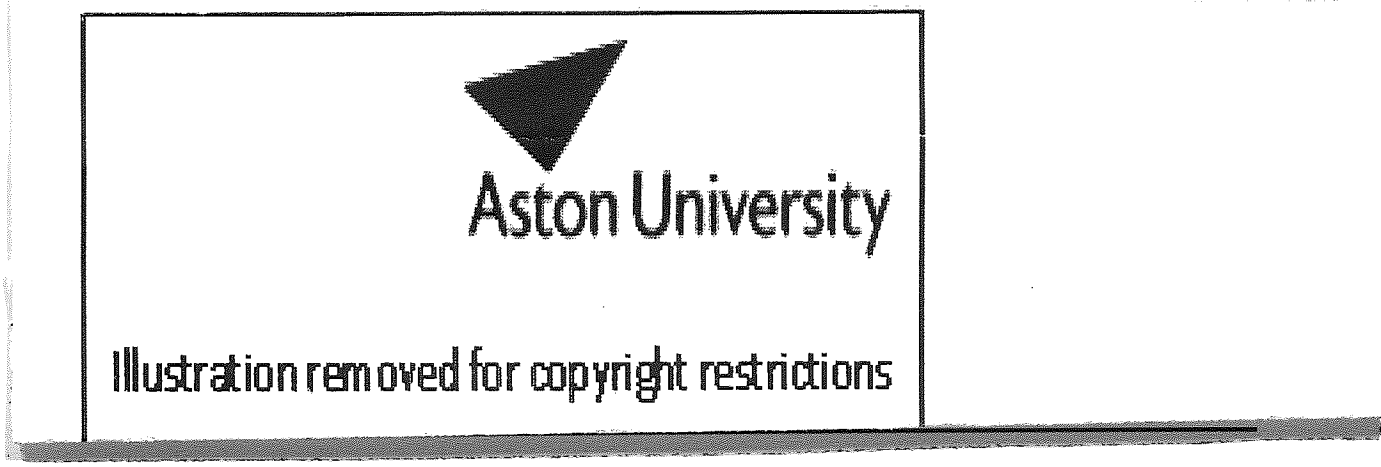

Table 2 - 1 : *Types of Quality Group*

Quality Control

Quality Control (QC) is the activity of measuring achievements and performance against some preceding specification or standard (Smith & Wood, 1989). Schulmeyer (1987a) defines software quality control as:

> *"the systematic activities providing evidence of the fitness for use of the total software product."*

Yasuda (1989) stated that the Japanese Industrial Standards defines QC as the following:

> *"a systematic method of economically providing products or services that meet the user's needs."*

Yasuda (1989) outlined the software quality control approach as a plan, do, check, and action control cycle. QC check for errors and problems in the work. In QC, techniques such as inspections, review and testing are part of the process (Daily, 1992). Similarly, the software quality control outlined by Yasuda includes inspection as part of its feature. The complete features of QC as describes by Yasuda (1989) are as follows:

- There is a software factory system similar to that for hardware manufacturing. The factory system features a line organisation clearly divided into design department and inspection department
- The inspection department is responsible for product inspection. It has the right to reject products that are judged to be unacceptable
- All the workers take part in QC, which includes personnel from top management to programmers.

The basis of all control and quality is measurement (Arthur, 1985). In some organisations, the responsibility of QC is handled by the SQA organisation.

2.2.2 Capability Maturity Model

Software quality and productivity improvement is dependent on process improvement. One of the process improvement approaches available today is the

Capability Maturity Model (CMM), which is the most popular model to date (Lai, 1993). Humphrey (1988) suggested that an important first step in addressing the software problem is to treat the entire development task as a process that can be controlled, measured, and improved. In order to improve their software capabilities, Humphrey (1988) outlined five steps that must be taken. They are:

- understand the current status of their development process or processes
- develop a vision of the desired process
- establish a list of required process improvement actions in order of priority
- produce a plan to accomplish these actions
- commit the resources to execute the plan

Humphrey (1989) later added a sixth step which says:

- start over at step 1.

The maturity framework presented by Humphrey (1988) addresses these five steps by characterising a software process into one of five maturity levels. The five levels of process maturity, as proposed by Humphrey (1988, 1989), are as follows:

- Initial - Until the process is under statistical control, no orderly progress in process improvement is possible
- Repeatable - The organisation has achieved a stable process with a repeatable level of statistical control by initiating rigorous project management of commitments, cost, schedule, and changes
- Defined - The organisations has defined the process, to ensure consistent implementation and provide a basis for better understanding of the process. At this point, advanced technology can usefully be introduced

- Managed - The organisation has initiated comprehensive process measurements, beyond those of cost and schedule performance. This is when the most significant quality improvements begin
- Optimising - The organisation now has a foundation for continued improvement and optimisation of the process.

The process maturity framework later evolved into the CMM (Paulk *et al.*, 1993). CMM maturity levels contains key process areas. Key process areas indicates where an organisation should focus to improve software process (Paulk *et al.*, 1993). They identify the issues that need to be looked into to achieve a maturity level.

The key process areas of CMM maturity levels as outlined by Paulk *et al.* (1993) are as follows.

Level one has no key process areas.

The key process areas for level two includes: requirements management, software project planning, software project tracking and oversight, software subcontract management, software quality assurance, and software configuration management. The key processes at this level focus on establishing basic project management control (Paulk *et al.*, 1993).

The key process areas for level three consists of: organisation process focus, organisation process definition, training programs, integrated software management, software product engineering, inter-group coordination, and peer reviews. The focus here is on project and organisational issue (Paulk *et al.*, 1993).

The key areas for level four are: quantitative process management, and software quality management. The emphasis of key process areas at this level

is on establishing quantitative understanding of software process and software work products being built (Paulk *et al.*, 1993).

The process for level five includes: defect prevention, technology change management, and process change management. The key process areas at level five emphasise on issues that affect continuous and measurable process improvement (Paulk *et al.*, 1993).

As the organisation grows and changes, the needs and roles also change. Hoffman (1993) stated that the level of maturity of the organisations roughly correlates with the role of the software quality group (see *Table 2 - 2*).

Table 2 - 2 : *Organisation Maturity and SQA Roles*

The only drawback of CMM is that it cannot easily be applied to small organisations and small projects where practices and structure differ considerably from the large organisations upon which the CMM was modelled (Brodman & Johnson, 1997). Among the problems identified by Brodman & Johnson (1997) are as follows:

- documentation overload
- unrelated management structure
- inapplicable scope of reviews

- high resource requirements
- high training costs
- lack of need guidance, and
- unrelated practice.

Cousin (1993) reported the same problem with implementing the CMM program in small organisations with resource constraints. Cousin (1993) also asserted that an organisation's corporate culture may be structured such that the application of the program would be difficult or disruptive. As an alternative, Cousin (1993) proposed an eclectic model of CMM level progression. In this model, similar levels of CMM progression could be effectively achieved by repeatedly applying a program of practical CMM improvement. This consists of studying the prescribed actions of the CMM, implementing these actions from each CMM level for which the organisation have adequate resources. The process is repeated again and again.

2.2.3 Verification & Validation and Software Testing

This section describes two methods used to achieve software quality. They are verification and validation (V&V) and software testing. The definition and brief description for both methods is covered.

Verification and Validation

Boehm (1982) gave a short definition of verification and validation (V&V). He stated that:

> *"verification is doing the job right and validation is doing the right job."*

In a more detailed definition, Schulmeyer (1987a) defines verification as:

> *"the process of evaluating software to ensure compliance with specified requirements."*

He defines validation as:

> *"the process of determining whether or not products of a given phase of the software development cycle fulfil the requirements established during the previous phase."*

Validation of software requirements is an important part of software engineering (Reese & Leveson, 1997). In a study by Mills *et al.* (1987), they discovered that human verification, even though fallible, could replace debugging in software development. They further stated that even informal human verification can produce software sufficiently robust to go to system test without debugging. Mills *et al.* (1987) established that human verification need take no more time than debugging although it takes place earlier in the cycle. Mills *et al.* (1987) used a combination of formal design methods and mathematics-based verification and they discovered that more than ninety percent of total product defects were found before first execution compared to sixty percent normally.

Software Testing

Myers (1979) defined testing as:

> *the process of executing a program with the intent of finding errors.*

In another definition, Schulmeyer (1987a) defines software testing and evaluation as:

> *"the process of exercising complete programs to judge their quality through the fulfilment to specified requirements."*

The main purpose of software testing is to serve as a risk-reducing activity (Offutt, 1991). Offutt (1991) further argued that testing can consume up to half of the cost of developing software. Offutt also stated that developing test cases is technically difficult, tedious, and time consuming.

Testing can be done in two ways, the black-box testing and the white-box testing (Humphrey, 1989). In black-box testing, the tester is not concerned with the internal behaviour and structure of the program, but is more interested in finding out in which circumstances the program fails to meet its specification. In white-box testing, the internal structure of the program is examined in order to derive test data. Ideally, exhaustive path testing would be done. Myers (1991) also suggested that a programmer should avoid attempting to test his or her own program. According to Humphrey (1989), there are seven types of software testing. The seven basic types are as follows: unit tests, integration tests, external function tests, regression tests, system tests, acceptance tests, and installation tests.

Myers (1979) further questioned whether it is possible to test a program to find all its error. Myers (1979) showed that this is not possible, even for trivial programs. Grady (1993) showed that on the average, only between fifty to sixty percent of the code is covered during testing even though they thought that they had tested their products thoroughly. Correspondingly, Humphrey (1989) asserted that testing is an inefficient way to find and remove many types of bugs.

Notwithstanding its limitation, software testing is a critical part of the software process (Humphrey, 1989). Strange (1993) asserted that software testing is the most visible, quantifiable tool for software quality assurance. The potential results from poorly controlled testing are very serious. Strange (1993) listed seven effects of poorly planned testing, they are:

- delivery of poor software quality
- substantial re-writes of modules after implementation
- difficulty with addition of enhancements to the system
- inaccurate projections of maintenance costs
- inappropriate confidence on the part of senior management
- lengthy, costly, or failed implementations
- unacceptable down-time of applications once in production.

2.2.4 Inspection, Review, and Walkthrough

Schulmeyer (1987b) presented eighteen major software quality related standards that have existed since 1974. The standards were from organisations such as IEEE (Institute of Electrical and Electronic Engineers), United States Department of Defence, and NATO. Only four out of eighteen of the standards outlined did not include code reviews. All the standards that were proposed since 1981 include code reviews process as part of the standard. This shows the importance of code reviews and its variants in the process of achieving high software quality. Inspection or reviews was also mentioned frequently as one of the most important techniques in SQA, QC, and CMM, as discussed in previous sections. Yasuda (1989) goes to the extent of saying that SQA activities begins with the introduction of an inspection system.

In practice, there are a number of variations with regard to inspections and reviews techniques. The reasons for this vary from place to place. Freedman & Weinberg (1990) listed down three principal reasons for this:

- Different external requirements, such as government contract provisions
- Different internal organisations, such as the use or non-use of teams
- Continuity with past practices.

Matters take a turn for the worse when different authors uses a different phrase for an identical implementation of the technique. Variations of this evaluation technique include the following: inspection, formal technical review, management review, walkthrough, round-robin reviews, informal reviews, and audits (Freedman & Weinberg, 1990; Gilb & Graham, 1993; Smith & Wood, 1989; Pressman, 1992; Daily, 1992; Weinberg & Freedman, 1984; Humphrey, 1989; Schah, 1993). The variations are divided into inspections, reviews, walkthrough, and audit and are discussed below.

Inspections

Inspections have at least three different variations. The variations include the original work on inspections by Fagan (1976), and the variation proposed by Gilb & Graham (1993) and Humphrey (1989). The differences between the variation is covered in *Section 2.3*. Inspection is a form of peer review (Daily, 1992). Generally speaking, the aim of inspection is to analyse a product or document to detect defects (Fagan, 1976; Daily, 1992). Fagan (1986) defines defect as the following:

> *"A defect is an instance in which a requirement is not satisfied."*

On the other hand, Freedman & Weinberg (1990) contended that inspection is used commonly as part of a feasibility study. Schulmeyer (1990) goes even further by dividing inspection methods into three. These are as follows:

- Judgement inspections that discover defects. A defective process will produce a defective product. Defects will not be reduced merely by making improvements at the final stage, although it may eliminate defects in the final goods
- Informative inspections that reduce defects. It is an inspection process in which information of a defect is fed back to the specific work process, which then corrects the process
- Source inspections that eliminate defects. This methods are based on discovering errors conditions that give rise to defects and performing feedback and action at the error stage so as to keep those errors from turning into defects, rather than stimulating feedback and action in response to defects.

Daily (1992) stated that inspection cannot be guaranteed to find all defects in the product. Inspection techniques include pre-defined procedures that include steps of individual preparation, performance of the meeting, and follow-up of defects and problems (Daily, 1992).

The benefit of using inspection is well documented (Ackerman, 1984; Kitchenham *et al.*, 1986). The potential saving from finding errors early in the life cycle are considerable (Daily, 1992). A study by Olsen (1993) shows that the detection efficiency for system design inspection is 17.3 percent, for module design inspection is 19.1 percent, and for code inspection is 15.1 percent. This gives a total of 51.4 percent detection efficiency for the inspection process in the study (Olsen, 1993). Further elaboration on software inspection is in *Section 2.3*.

Reviews

There is more variation in review techniques than in inspections. The variation includes: formal technical review, management review, round-robin review, and informal review (Freedman & Weinberg, 1990).

Formal technical review (FTR) objectives are to evaluate conformance to specifications and plans and to ensure change integrity (Humphrey, 1989). On the contrary, Pressman (1992) stated that FTR is a class of reviews that include walkthroughs, inspections, round-robin reviews, and other techniques. In another version of FTR as proposed by Freedman & Weinberg (1990), the team is consists of a leader, a recorder and a number of reviewer. The total number of participants is between four and seven. The group is expected not only to find defects and problems, but also to report back if the product is good or bad (Freedman & Weinberg, 1990). Freedman & Weinberg (1990) stated that the review group is expected to give an accurate message to management, producers, and other parties concerned with the technical quality of a product,

Management reviews are generally conducted for management and provide information for management actions (Humphrey, 1989). Among its objectives are to ensure progress, recommend corrective action, and ensure proper allocation of resources (Humphrey, 1989).

In round-robin reviews, the process is conducted by emphasising a cycling through the various participants (Freedman & Weinberg, 1990). Each person takes an equal and similar share of the entire task. Freedman & Weinberg (1990) further affirmed that this type of review is useful in situations where the participants are at the same level of knowledge. On the other extreme, informal review have no set way to do them. Nevertheless, Freedman & Weinberg (1990) listed three roles that need to be filled. They are the leader, program writer, and the reader.

Walkthrough

Walkthrough is the least formal variant among these techniques. Nevertheless, there are two variations of walkthrough. In some cases, walkthrough has been used in the literature as a synonym of inspection or review. Freedman & Weinberg (1990) stated that in its most usual form, walkthrough is done on the basis of a step-by-step simulation of a procedure, as when walking through code, line by line. This is done with an imagined set of inputs. Walkthrough doesn't demand any preparation on the participant's part and it can serve an educational purpose (Freedman & Weinberg, 1990; Humphrey, 1989). The producer of the reviewed material guides the progression of the walkthrough. Data are collected by taking notes and making an individual or group report at the end (Freedman & Weinberg, 1990). The problem of walkthrough, as identified by Freedman & Weinberg (1990) is the lack of preparation on the part of the participant. The high degree of ego involvement by the presenter, who is the producer of the material, can be the most serious problem of all with walkthrough (Freedman & Weinberg, 1990).

Structured walkthrough is another variants of walkthrough. It was proposed by Yourdon (1989). Structured walkthrough has more in common with inspections than with normal walkthrough. Yourdon (1989) promotes it as a semi-formal walkthrough. Structured walkthrough involves several people who have a definite role. Yourdon (1989) suggests seven roles to be played. They are as follows:

- The presenter - the task is to present the product. In most cases, the presenter is also the producer or the author. The product can also be presented by somebody else or in some cases, with no presenter at all
- The co-ordinator - ensures that the activities are planned and organised properly. During the walkthrough, the co-ordinator serves as a moderator

- The secretary or scribe - takes notes during walkthrough. The notes serve as a permanent record of the results
- The maintenance oracle - to review the product from the viewpoint of future maintenance
- The standard bearer - concerned with adherence to standards. The person can be the same as the maintenance oracle.
- A user representative - someone who can ensure that the product is meeting the customer's needs
- Other reviewers - to give a general opinion of the correctness and the quality of the product.

Audits

There exist another variation in this assortment of evaluation techniques, which is the audit. In audit, the process is conducted by an fully independent external person or group (Daily, 1992). It is a formal process and performed in a manner that ensures credibility. A defined audit process has to be followed. Daily (1992) said that conclusion by the audit team must be based on tangible evidence.

2.3 SOFTWARE INSPECTION

For the purpose of this research, the term software inspection is used when referring to an evaluation technique to find defects and problems in a product. In certain cases, the term review is use when quoting from other studies. If that

is the case, the term review refers to the same implementation of the evaluation process that this research refers to as software inspection.

To recap from *Section 2.2.4*, the aim of inspection is to analyse a product or document to detect defects (Fagan, 1976). Grady (1993) extended this concept further by stating that while inspections are primarily a defect-prevention technique, their experience showed that the process of preparing for the inspection also results in defect prevention. Inspections processes also impose discipline, provide measurable feedback, and provide a way to educate a team (Grady, 1993). Three aspects set inspection apart from other evaluation process (Abbot, 1986). These are as follows:

- it is a highly-directed process
- it is maintained by the use of feedback
- it is highly formalised.

Arthur (1993) highlighted that you don't inspect code to find defects in the code, but rather to find defects in the software process. Routine inspection to improve quality is equivalent to planning for defects (Arthur, 1993). Inspection require proper training of personnel and full support from management (Daily, 1992). The inspection technique has pre-defined procedures that include the steps of individual preparation, performance of the meeting, and follow-up of defects and problems (Fagan, 1976; Daily, 1992; Schah, 1993). Software inspection is a manual process. The most difficult and time consuming phase are:

- comment preparation, where code inspectors analyse the module to be inspected
- code inspection meeting, where inspectors and author of the code meet to discuss and record the comments prepared.

In the standard software inspection procedure, there are moderators to administer the meeting, readers to read the text, scribes to record any proposed comment, the author of the code and an inspector.

Daily (1992) stated that inspection cannot be guaranteed to find all defects in the product. Nonetheless, the benefit of using inspection is well documented (Ackerman, 1984; Kitchenham *et al.*, 1986). The potential saving from finding errors early in the life cycle are considerable (Daily, 1992). Inspections also provide control throughout the entire development process (Dobbins, 1987). Grady (1993) also said that inspections help them uncover many defects before they become very costly. Agreeing with this, Schulmeyer (1990) stated that inspections of the proper type are clearly of great benefit for software development. Substantial net improvement in programming quality and productivity have been obtained through the use of formal inspections of design and code (Fagan, 1976). A study by Olsen (1993) shows that a total of 51.4 percent detection efficiency was achieved for the inspection process. Baker (1997) disclosed that there is 40 percent reduction in the number of defects found during testing when code reviews were conducted.

Humphrey (1989) said that while inspections are labour-intensive, they are often more effective than testing for removing defects. A study by Grady (1993) confirmed this. Grady (1993) reported that the defect-finding rate for code inspections was 4.4 times better than testing. Experience at Bell-Northern Research also confirmed this. Russell (1991) stated that inspections of 2.5 million lines of code yields between 0.8 to 1 defect per man-hour, which is two to four times faster than detecting code errors by execution testing. Other experiments have shown that code inspections save time and money in the detection of certain types of coding errors before testing (Ackerman, 1984). In the original study by Fagan (1976), coding productivity was increased by 23 percent

when I1 and I2 was introduced (see *Figure 2 - 4*). 82 percent of total error was also found by I1 and I2.

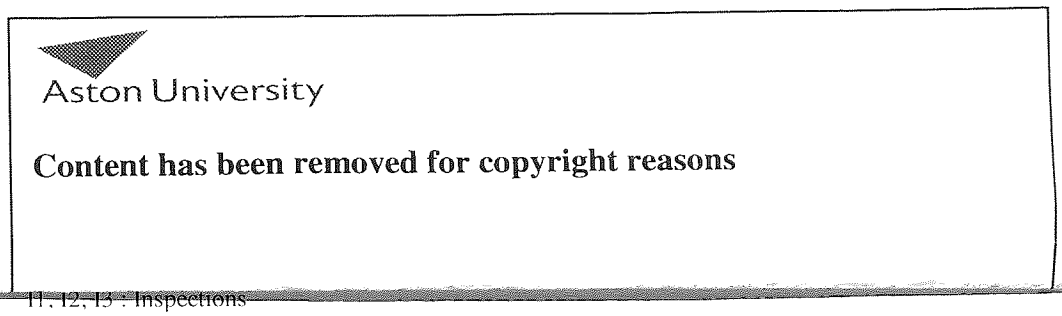

Figure 2 - 4 *: Coding Productivity according to Fagan (1976)*

There are many factors that can affect the costs and benefits of software inspection implementation. Some of the factors are process structure, process technique, process inputs, and also the process environment itself (Porter *et al.*, 1997). Data from Bull HN Information System shows that inspection should be conducted before unit tests (Weller, 1993). Weller (1993) showed that unit test conducted before inspection resulted in more defects after shipping. Weller also revealed that four-person teams are twice as effective as three-person teams. In theory, any software document can be inspected. Abbot (1986) asserted that only design documentation, code and test plans are inspected in this way (Abbot, 1986). Abbot (1986) reasoned that documents produced at earlier stages are less amenable to the detailed and specific investigation characteristic of inspection, Fagan's style. On the contrary, Daily (1992) stated that inspection is relevant to the earlier stages, such as requirement specifications, design, test plan, and coding. Meanwhile, Schulmeyer (1990) believes that inspections flow throughout the software development process. Daily (1992) stated that implementing a highly formal inspection can be difficult to practice outside large corporations. Cousin (1993) also stated that small projects tradi-

tionally have no formal peer review of work, although there exist informal reviews for specifications, designs, and code. Three of the major variants of formal software inspections are describes in the following sections.

2.3.1 Fagan's Software Inspection

Software inspection was introduced in IBM in 1972 (Fagan, 1986). Fagan (1976) stated that inspections are a formal, efficient, and economical method of finding errors in design and code. In Fagan's inspection process, there are basically five steps in the process. The process steps are as follows (see *Figure 2 - 5*):

- Overview - The overview is needed in design inspection but not in code inspection. In the overview phase, the whole inspection team is involved. The designer describes the overall and specific area in detail
- Preparation - The preparation is done on an individual basis. Reports from previous inspections and a checklist should be given to the participant
- Inspection - In the inspection phase, a formal meeting is conducted with the whole team. All instructions are addressed at least once in the conduct of inspections. A reader, who is normally the coder, describes the document. Errors are recorded by the moderator
- Rework - Finally the follow-up is conducted by the moderator to ensure that the recommendation is followed through.
- Follow-up - Rework is done by the coder or implementor

Figure 2 - 5 : *Fagan's Code Inspection Process*

A planning phase was later added to the improved version of the inspection process (Fagan, 1986). At this stage, the materials to be inspected must be deemed to have met inspection entry criteria.

Fagan's inspection teams usually consist of four people. However, the number can change if circumstances indicate otherwise (Fagan, 1976). The inspection team roles described by Fagan (1976) are as follows:

- Moderator - The duties include managing the inspection team, scheduling suitable meeting places, reporting inspection results within one day, and follow-up on rework.
- Designer - The programmer responsible for producing the program design.
- Coder or Implementor - The programmer responsible for translating the design into code. The coder also acts as a reader.
- Tester - The programmer responsible for writing and executing test cases.

The roles were later modified slightly by Fagan (1986). They are as follows: moderators, authors, readers, and tester. The author is a coder or a designer. The total time taken to complete the whole process is between 90 to 100 people-hours (Fagan 1976). Fagan (1976, 1986) recommended that one inspection session should not last more than two hours. Errors found are classified by type, and frequency of type occurrence is ranked.

In the inspection process of Fagan, an analysis is also conducted in order to get feedback, feedforward and self improvement (see *Figure 2 - 6*). Analysis

of the inspection process is conducted and is passed as a feedback to 'Operation 1'. It is also passed as a feedforward to 'Operation 2' (see *Figure 2 - 6*). The analysis is also used as self improvement tool for the next inspection process. Fagan (1976, 1986) stressed that the inspection results must never be used for personnel performance appraisal.

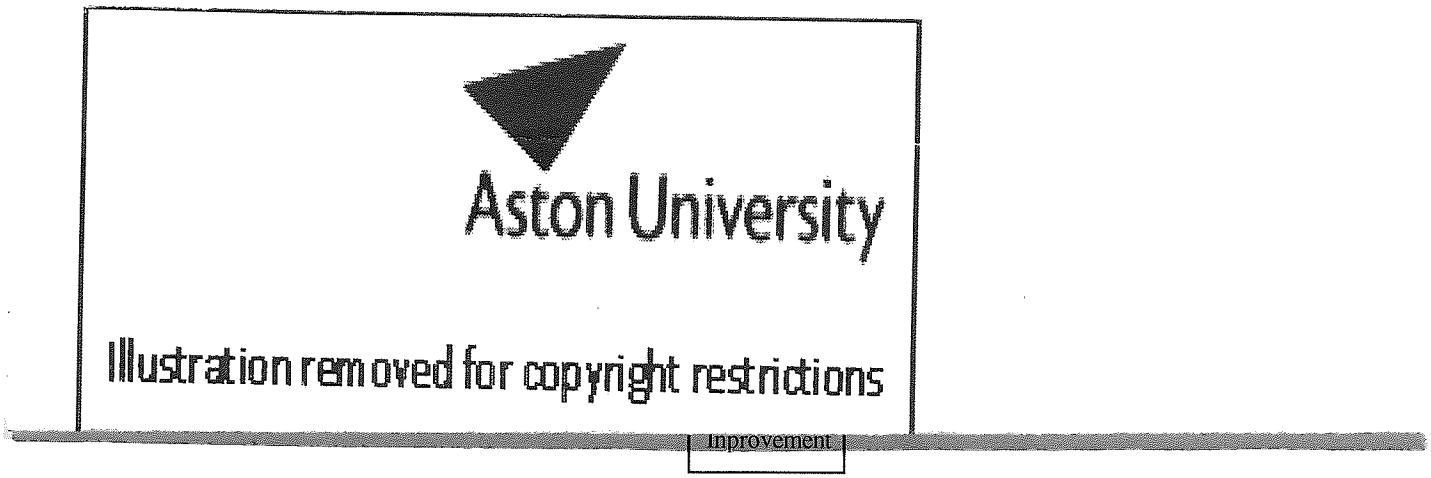

Figure 2 - 6 **:** *Inspection Process with Analysis*

2.3.2 Gilb and Graham's Software Inspection

In the software inspection model proposed by Gilb & Graham (1993), ten inspection phases are identified (see *Figure 2 - 7*). The inspection process phases are as follows:

- Request - Initiated by the author or owner of product document to start the inspection process
- Entry - Entry criteria area set to make sure products meet certain standard before inspection can proceed
- Planning - Planning is done by the leader to determine objectives and tactics

- Kick-off meeting - The kick-off meeting is held to train and motivate the inspection team. This is to ensure that the checkers know what is expected of them
- Individual checking - The inspection start with individual checking to search for potential defects. The product document inspected is compared against source document. Rules, procedures, and checklists are provided
- Logging meeting - This is to log issues found in previous phase and check for more potential defects. Anything logged at the meeting are known as an 'item'. An item can be an 'issue', which is a potential defects, 'question of intent', or a process improvement suggestion. Suggestions for improvement are allowed at the end of the meeting
- Edit - The log of issues is given to the author for editing
- Follow up - A follow up by the team leader is arranged. The leader checks that editor action has been taken on all logged issues. Process improvements suggestion are sent to the owner of the product document
- Exit - The inspection leader compare the products against the exit criteria to ensure that it meets the standards and the products is release
- Release - The product is made available to the next phase.

The inspection model by Gilb & Graham (1993) contains another phase which is called process improvement. Every issues log in the logging meeting from every inspection meeting is forwarded to the process change management team. This will contribute to the overall improvement of the development process. Gilb & Graham (1993) suggest that inspections can be used on requirements, architecture, design, or code documents. They recommend an inspection team to consist of two to three people for maximum efficiency. On the other hand, they recommend four to five people for maximum effectiveness. Gilb &

Graham's (1993) inspection team consists of an inspection leader, author, and checker. During the logging meeting a moderator, who is normally the inspection leader, leads the meeting. A scribe is appointed to record the meeting.

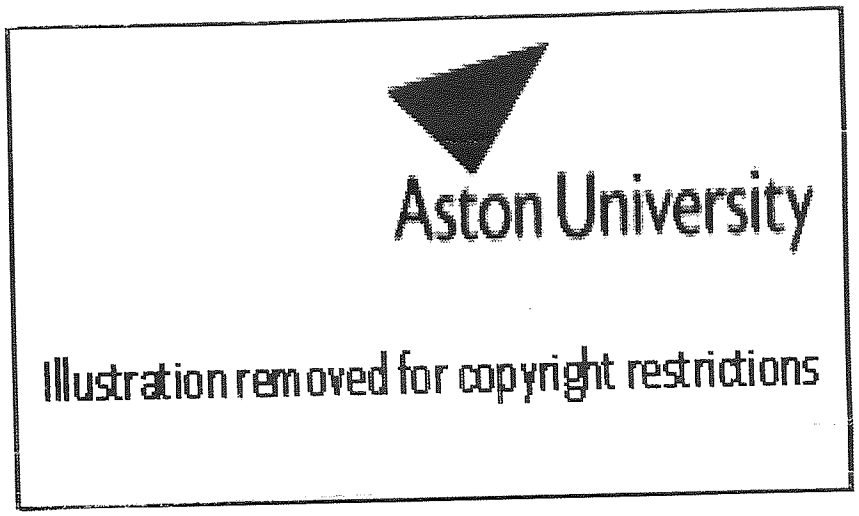

***Figure 2 - 7** : Software Inspection as Proposed by Gilb & Graham*

2.3.3 Humphrey's Software Inspection

Humphrey (1989) stated that while there is no limit on what can be inspected, there is a question of cost. Inspections, according to Humphrey, are invariably cost-effective for design and code and often for requirements, test cases, and documentation. A less formal walkthrough process is generally adequate when the cost of inspections does not seem warranted (Humphrey, 1989). Humphrey spell out six basic principles of inspection process, these are as follows:

- It is a formal, structured process with a system of checklists and defined roles for the participants.
- Generic checklists and standards are developed for each inspection type and are tailored to specific project need, where appropriate.

- The reviewers are prepared in advance and have identified their concerns and questions.
- The focus of the inspection is on identifying problems, not resolving them.
- An inspection is conducted by technical people for technical people.
- The inspection data is entered in the process database and used both to monitor inspection effectiveness and to track and manage product quality.

The inspection participants, as recommended by Humphrey (1989) are as follows: the moderator, the producers, the reviewers, and the recorders. The inspection process consists of preparation phase, the inspection itself, and post-inspection activity (Humphrey, 1989). The preparation phase consists of entry criteria and opening meeting. The inspection meeting begins with a submission of inspection reports from the individual inspection session. The producer reviews each major error in the report and clarifies or accepts it. The product is then inspected again to identify other concerns. The moderator makes sure that the metrics are inserted in the process database.

2.3.4 Other Model of Software Inspection

Phased inspection was developed by Knight & Myers (1991). This technique consist of a series of co-ordinated partial inspections called phases. Each phase is designed to ensure the product possess either a single, specific property or a small set of related properties (Knight & Myers, 1991, 1993). Phased inspection can be conducted either using the single-inspector technique or the multiple-inspector technique. In a single-inspector phase, a rigorous checklist is used. The work product cannot complete the inspection until it complies with all of the checks in the list. The multiple-inspector phased inspection is divided into two phases. In the first phase, the product is examined individually by

each inspector using a checklist. The individual meeting is followed by the second phase which is a meeting called reconciliation (Knight & Myers, 1993, 1991).

N-Fold inspection uses formal inspections but replicates these inspection activities using *N* independent teams (Martin & Tsai, 1990). The same software artefact, namely user requirements document or functional specification document, is given to all *N* teams. A moderator is appointed to supervise all *N* teams. Martin and Tsai (1990) suggests that *N*-Fold is suitable for the initial phase of software development for mission-critical software. The primary use of *N*-Fold inspection is to identify faults that might not be detected by a single inspection team.

2.3.5 CASE Tool for Software Inspection

From *Section 2.1.4,* the basic reason for using CASE systems is to improve the quality and productivity of the work (Humphrey, 1991). Schah (1993) also mentioned that the simplest form of CASE is the software tool, which assist in just one aspect of the production of software. In one of the category of CASE tools, Pressman (1992) stated that support tools category comprises systems and application tools that complement the software engineering process. Support tools include documentation tools, system software tools, quality assurance tools, and database and software configuration management tools. Thus, CASE tools for software inspection can be categorised as support tools. Vessey & Sravanapudi (1995) stated that CASE tools intended to support a work group in the system development process should address issues such as co-ordinating interdependent activities and project management.

Manual inspection is labour intensive, requiring the participation of more than four people over a period of time (Macdonald *et al.*, 1996). By pro-

viding a CASE tool for the process, it has the capability of being made more effective and efficient. Macdonald *et al.* (1996) stated that some of the features of software inspection that have been tackled by current inspection tools are document handling, individual preparation, meeting support, and data collection. One of the first CASE support tool for software inspection was ICICLE which was proposed by Brothers *et al.* (1990).

ICICLE

Brothers *et al.* (1990) proposed a system called Intelligent Code Inspection Environment in a C Language Environment (ICICLE). The system that they proposed includes software tools to detect routine sorts of errors, offers various forms of knowledge about the code being inspected, allows code inspectors to traverse source code in a windowed environment and it is designed to render the code inspection meeting paperless. The first instance of communication is through the code window where the reader will guide all the inspectors through the module being analysed. ICICLE locks all code windows synchronously with the reader. The second instance of communication concerns the propagation of annotations from one screen to the screens of other inspectors. Inspectors may interrupt the activity to proposed a comment. Any inspector may also 'point' to something on the other user's screen. ICICLE will simply draw a mark on the other inspectors' code windows.

ICICLE groupware is intended to be used in one room with all the team members close enough together for easy conversation (Brothers *et al.*, 1990). The system is intended to be used in a face-to-face situation. ICICLE based code inspection is essentially based on Fagan's version of inspection and it is designed specifically for C and C++ programming languages. There is no change in the flow of the process, except that the inspection phases are done

with the help of the system. The role of the scribe is merged into the role of moderator because of the existence of tools in the system. The system also includes tools to assist individual preparation.

The ICICLE implementation does not support dynamic changes to the meeting structures (Brothers *et al.* 1990). It is also strictly for code inspection only and is less suitable for supporting general inspections (Macdonald *et al.*, 1996). There is also no support for distributed inspection meetings.

InspeQ

InspeQ stands for Inspecting software in phases to ensure Quality. A toolset, InspeQ (Knight & Myers, 1993, 1991), was developed to support phased inspections. The toolset only supports single-inspector inspection and the individual phases of the multiple-inspector-inspector inspection. InspeQ facilities include displaying the work product, documentation display facilities, facilities for the inspector to record his comments, and facilities to enforce the Phased Inspection process (Knight & Myers, 1993, 1991).

2.4 SUMMARY

In summary, *Section 2.1* discussed software engineering issues in general. Issues on software engineering methodology, process model, software engineering management and CASE tools were presented. *Section 2.2* focused on the issue of software quality. Definition of software quality was given along with discussion on QA and QC. Techniques and methods to achieve high software quality were furnished. These include description on CMM, V&V, software testing, and evaluation techniques. In *Section 2.3,* the topic is narrowed down to the

issue of software inspection. Major variants of software inspection were discussed. The last section discussed costs and benefits of support tools for inspection. The issue of Computer Supported Cooperative Work and the relationship between Software Engineering and CSCW in general and Software Inspection and CSCW in particular is discussed in the next chapter.

Chapter 3

REVIEW OF LITERATURE ON CSCW

3.0 INTRODUCTION

This chapter deals with the second main area of this research, that is, computer supported cooperative work (CSCW). In this chapter, topics reviewed are as follows:

- **Section 3.1** spells out the computer supported cooperative work topic. Work related to fundamentals of computer supported cooperative work and early development issues in groupware are reviewed.
- **Section 3.2** discusses design issues in CSCW, including acceptability issues.
- **Section 3.3** discusses the relationship between software engineering and computer supported cooperative work. Topics on groupware support for software development are included.
- **Section 3.4** canvasses the subjects of groupware support for software inspection.

3.1 COMPUTER SUPPORTED COOPERATIVE WORK

Chapter One has dealt with the introduction of computer supported cooperative work (CSCW). It was first coined by Irene Greif as simply a shorthand way of referring to a set of concerns about supporting multiple individuals working together with computer systems (Greif, 1988; Grudin, 1993). Wilson's (1990, 1991) definition of CSCW was also discussed. The definition given by Wilson for CSCW was as a generic term that combines the understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services and techniques. Desktop conferencing, videoconferencing, co-authoring features and applications, email and bulletin boards, meeting support systems, voice applications, workflow systems, and group calendars are common examples of groupware (Grudin, 1994). In this section, further discussion on CSCW is furnished. These include detailed classifications of CSCW from different point of views and early development issues.

3.1.1 Classification of CSCW

The first chapter presents Wilson's classification of CSCW in a detailed manner. To reiterate, Wilson (1990, 1991) divides CSCW into two major areas: the group working process and the enabling technology to support it. The group working process is further subdivided into four areas, namely, individual aspects, organisations aspects, group working design aspects and group dynamics aspects. Wilson further divides the enabling technologies into four areas. Those four areas are communication systems, shared work space systems, shared information systems and group activity support systems.

On the other hand, Ellis *et al.* (1991) present two taxonomies for viewing groupware. The first taxonomy is based upon notions of time and space; the second on application-level functionality. These time and space considerations suggest the four categories of groupware.

- Face to face interaction at the same place and in the same time, e.g., meeting room technology.
- The asynchronous interaction at the same place but in different time, e.g., physical bulletin board.
- Synchronous distributed interaction at different places but at the same time, e.g., real-time document editor.
- Asynchronous distributed interaction at different places and in different times, e.g., electronic mail system.

This consideration is based on the work of Johansen (1988) where he present his space-time matrix for groupware (see *Table 3 - 1*).

Table 3 - 1: *Space Time Matrix of Johansen*

A complete groupware system might include support for all the categories mentioned above.

The second taxonomy is based on application-level functionality. The purpose here is to give a "rough" idea of the breadth of the domain in CSCW. The second taxonomy can be classified into six major categories, namely, message systems, multi-user editors, group decision support systems and electronic

meeting rooms, computer conferencing, intelligent agents, and co-ordination systems. Overlapping does exist and merging of functionalities will happen in the future as demand for integrated systems increases.

According to Ellis *et al.* (1991), the message system is the most familiar example of CSCW system. Included in this group are electronic mail, computer conferencing and bulletin board systems. In this kind of system, the asynchronous exchange of textual messages takes place between users. Multi-user editors are used to jointly compose and edit a document. The goal of group decision support systems and electronic meeting rooms is to improve the decision making process by speeding up the process or by improving the quality of the result. Computer conferencing allows the user to do the following: real-time computer conferencing, computer teleconferencing, and desktop conferencing. Intelligent agents are generated automatically by the system. The agents are responsible for a set of tasks and their actions resemble those of other users. In a coordination systems, users will be allowed to view their actions and the actions of others, within the context of the overall goal. Coordination systems are normally based on these four models: form-oriented model, procedure-oriented model, conversation-oriented model, and communication structure-oriented model.

There is some similarity between Ellis *et al.* (1991) application-level functionality taxonomy and Wilson's enabling technologies classification. Both classifications divide and group CSCW system and research area based on the type of the application. Both approaches talk about applications and systems, although they disagree on how many classes they should have.

With regard to viewing the CSCW research from a space-time angle, Blair & Rodden (1994) said that there is some difficulty caused by classifying CSCW using the space and time taxonomy because work often switches rapidly between synchronous and asynchronous interactions. Olson *et al.* (1993) stated

that although this characterisation helps separate different major modes of group work, these distinctions can be blurred to advantage by the development of certain technologies. Olson *et al.* (1993) further asserted that technology has the power to blend synchronous and asynchronous work in new ways. Continuing on the same topic, Rodden & Blair (1992) argued that CSCW can be classified by the form of cooperation. Three general classes proposed by Rodden & Blair (1992) are purely asynchronous systems, purely synchronous systems, and mixed systems.

3.1.2 Early Development of CSCW

Opper & Fersko-Weiss (1992) wrote that among the first CSCW systems was the computer conferencing program. The first conferencing-like abilities were introduced on an existing email network in early 1970's (Opper & Fersko-Weiss, 1992). Around the same time, a few systems were developed which includes the Electronic Information Exchange System. It was developed by Hiltz & Turoff (1978). A number of major computer vendors were also interested in this area and were creating ad hoc internal CSCW systems to take advantage of their widespread telecommunications facilities. The early systems existed long before people had personal computers and before the users were interested in groups working together electronically.

In the software universe, groupware lies somewhere between single-user applications and information systems that support an organisation (Grudin, 1994). Organisation here means a company, or enterprise, or corporation. Each software development area emerged independently. Systems designed to support organisations achieved prominence first. "Organisation goals" are major goals typically defined by upper management. These research activities have variously been labelled data processing (DP), information systems (IS),

management information systems (MIS), and information technology (IT). According to Grudin (1994), conditions that encourage the emergence of CSCW in the 80's included:

- computation inexpensive enough to be available to all members
- technological infrastructure supporting communication and co-ordination
- widening familiarity with computers
- maturing single-user application domains that pushed developers to differentiate products.

Most systems addressing organisational goals are developed in-house or contracted out. But most single-user applications are commercial off-the-shelf product. In another study, Grudin (1991) said that European research in CSCW focused on internal development to address organisational needs, while in the United States, the research is focused on the off-the-shelf products. It is a move by product developers to expand beyond single-user applications. Groupware is largely a new market for product developers. As developers shift from supporting individual users to supporting groups, many encounter the challenges described in *Section 3.2*.

3.1.3 Other Views on CSCW

There are many points of view that can be found on CSCW. Even the terms coined for this area can be different. Every researcher will view CSCW according to his or her own background and perspective. Other classifications are covered here and any similarity with Wilson's classification is highlighted.

Other Definitions

As mentioned earlier, there are a variety of other terms that have been used in describing this field: Groupware, Workgroup Computing, Technology for Teams and Computer-Aided Teams (Wilson, 1991). Johansen (1988) describes groupware as a generic term for specialised computer aids that are designed for the use of collaborative work groups. He described these groups as small project-oriented teams that have important tasks and tight deadlines. Groupware can involve software, hardware, services, and group process support.

Meanwhile Opper & Fersko-Weiss (1992) describe Groupware as any information system designed to enable groups to work together electronically. They listed three requirements for a product to be groupware: it must permit people to communicate electronically, it must facilitate the management of the information they use in common and it must have woven throughout its features the assumption that it will be used by a group, not just by individuals.

Other Classification

Bannon & Scmidt (1991) meanwhile divide CSCW into three core issues:

- articulating cooperative work
- sharing an information space
- and adapting the technology to the organisation and vice versa.

In any cooperative effort, tasks are to be allocated to different members, where that worker is accountable for accomplishing that task and finally combining all the efforts of individuals and ensembles. Cooperative work can be conducted in a distributed way. Thus, any computer systems must support retrieval of information by other co-workers in order to support cooperative work.

3.1.4 Summary

The main idea in the classification of CSCW that is of interest to this research is the group work design aspects and the group dynamics aspects in the group working process as proposed by Wilson (1990, 1991). In group work design, research is being done in user involvement, prototyping and usability testing and group work design procedures. In the enabling technologies area, research in the shared work space systems, shared information systems, and group activity support systems is also important to this research. The space and time taxonomy of groupware (Johansen, 1988; Ellis *et al.*, 1991) provides an important method in analysing the working mode of a group. Work often switches between synchronous and asynchronous interactions (Blair & Rodden, 1994), thus making it difficult to classify any work in just one quadrant of the space-time matrix.

Throughout the remainder of this thesis, the term CSCW is used. Wilson's description of CSCW is used because it embraces most of the other terms used. The term groupware is also used to describe specific systems.

3.2 ISSUES AFFECTING CSCW

In this section, two issues affecting computer supported cooperative work are presented. These are the acceptability issue and the design issue.

3.2.1 Acceptability of CSCW

When an organisation purchases an expensive mainframe system, the upper management know organisational change is likely. Among those changes are

job redesign and creation, providing training, restructuring to work around important individuals and positive leadership through inspiration or example. These social and political factors that affect the introduction of large systems are little known to developers of single-user applications. Yet similar forces affect groupware. Groups share characteristics of organisations, but groups are not organisations and CSCW systems are different from large systems. Management is less committed to the less expensive groupware applications. An organisation will not restructure itself for each new application. A small application program must adapt to the organisation. Groupware is marketed as a product and is designed and evaluated to obtain a broad, competitive appeal. Declining technology costs bring large system software within the economic reach of groups. CSCW research aims to be inclusive, encompassing these and every activity at the organisational level. Grudin (1994) emphasises that because of the social and political factors, achieving groupware acceptance is much trickier than single-user product acceptance. Grudin (1994) contended that computer support has focused on organisations and individuals. Groups are different. Repeated, expensive groupware failures result from not meeting the challenges in design and evaluation that arise from these differences (Grudin, 1994). These are also caused by not understanding the unique demands this class of software impose on developers and users (Grudin, 1994).

Cockburn & Jones (1995) contended that although social implications of groupware are often cited as the fundamental barrier to its success, they noted that it is because of an accumulation of individual rejections. Individuals interact with a groupware application, bringing interface design challenges in the same way as single-user applications and also new challenges arising from its direct involvement in group processes. Robert Kraut said (Kraut, 1990),

"the only successful CSCW application has been email"

and Lee Sproull stated (Sproull, 1990),

> *"groupware will never be practical and widely used in organisations if it follows its current trajectory."*

These assessments are explained in the design issues later on in the following section.

Work conducted collectively may require interaction of people with multiple goals of different scope and nature. This gives rise to several problems such as different people preferring different problem solving strategies, decisions being always generated within a specific conceptual framework and the assumption that information is something innocent and neutral. Winograd & Flores (1986) wrote,

> *"Every time a computer-based system is built and introduced into a work setting, the work is redesigned, either consciously or unconsciously. We cannot choose to have no impact, just as we cannot choose to be outside of a perspective. We can make conscious choices as to which ones to follow and what consequences we anticipate."*

Cockburn & Jones (1995) asserted that if principles for groupware design make systems more acceptable to individuals, without hindering their value in group support, they are likely to be more acceptable to the user society.

The problems of trying to fit technology into the workplace are more serious with CSCW. Assumptions about the use situation become more important variables. In order to understand the interaction between the technical subsystem, the work organisation and the requirements of the task environment, a theoretical framework is needed. Electronic group work differs in fundamental ways from face-to-face behaviours. Opper & Fersko-Weiss (1992) stated that people learn the basics of group behaviour in childhood and adult group behaviour is rooted in these early behaviour pattern. Cockburn & Thim-

bleby (1991) also said that now there is a risk of CSCW systems ignoring the needs of the individual. Anything that changes or adjusts the way people work in groups is not easily tolerated. Roger (1994) stated that to gain the full benefit of groupware, it must be used within the complete social context of the work environment. Roger further said that groupware should not be used simply to mechanise processes that are already in place but should be used as an opportunity to redesign the complete operation of a department or company.

Opper & Fersko-Weiss (1992) wrote that users of all groupware systems need training on using the program, creating their own applications, and need also learn how to manage their groups with the new technology. There is also a need for a new set of work conventions so everyone can work in a similar fashion. Grudin (1994) wrote that products such as email, databases and code management systems are used successfully in group contexts. Email provides an equitable balance for sender and recipient, it is compatible with social practice, and it does provide exception handling (Grudin, 1994). These reasons might explain why email is relatively heavily used groupware that has been widely accepted by the users.

Sponsors, moderators and facilitators all have different roles in making groupware successful. The sponsor is someone who promotes the use of groupware. The moderators are responsible for a particular discussion and are usually the experts in the subject matter being discussed. The facilitator need not have any knowledge of the topic being discussed. Their job is to help groups and individuals function well in the groupware environment.

3.2.2 Design of CSCW

Ellis *et al.* (1991) wrote that there are at least five key perspectives for successful groupware, namely:

- distributed systems perspective
- communications perspective
- human-computer interaction perspective
- artificial intelligence perspective
- social theory perspective.

The importance of the relationship between CSCW and these areas is that each discipline advances our understanding of the theory and practice of CSCW (Ellis *et al.*, 1991).

Davies *et al.* (1994) stated that there are two approaches to product development, namely, technology driven and market driven. In the technology driven approach, the developers start with an idea, develop it into a prototype, look for a market to match the prototype and evolve the prototype into the final product. In the market driven approach, a market need is identified, then a prototype is developed, tested on prospective users, then fine-tuned to develop the final product. Ideally, these two approaches will meet somewhere. If this happens, a market need is perceived, a search for technology ensues and a prototype is developed. The prototype is shown to the prospective market for feedback. The prototype's use of new technology drives changes in perceived market need. This causes further revaluation of the market and further refinement of the product.

In their study, Davies *et al.* (1994) found that individuals and organisations needed the ability to represent and view data in many different ways. They also found that there is a significant overlap between groupware requirements and individual needs. Most features that the individuals favoured offered secondary benefits to the group.

Problem of Design

The eight problems listed below by Grudin (1994) stem from the social dynamics of groups, drawn from developer experiences, descriptions of short-lived products and research products, and experimental and modelling studies in the literature.

- Disparity in work and benefit - Groupware applications often require additional work from individuals who do not perceive a direct benefit from the use of the application. A groupware application never provides precisely the same benefit to every group member. Groupware is expected to provide a collective benefit. Some people must adjust more than others.

- Critical mass problems - Groupware may not enlist the "critical mass" of users required to be useful, or can fail because it is never to any one individual's advantage to use it. Most groupware is only useful if a high percentage of group members use it. Different individuals may choose to use different word processors but two co-authors must agree to use the same co-authoring tool

- Disruption of social processes - Groupware can lead to activity that violates social taboos, threatens existing political structures, or otherwise demotivates users crucial to its success. Groupware may be resisted if it interferes with the subtle and complex social dynamics that are common to groups. Central to group activity are social, motivational, political and economical factors that are rarely explicit or stable. Conflicts of interest can become major obstacles to success when group members have very different occupations or roles.

- Exception handling - Groupware may not accommodate the wide range of exception handling and improvisation that characterises much group activity. Work process can normally be described in two ways: the way things

are supposed to work and the way they do work. Groupware that enforces "standard procedures" may also bring work to a halt.

- Unobtrusive accessibility - Features that support group processes that are used relatively infrequently require unobtrusive accessibility and integration with more heavily used features. Many organisations are structured and responsibilities are divided in order to minimise the overall communication requirements and social interdependencies. This has two implications: groupware features will fare better if integrated with features that support individual activity: and infrequently used groupware features must not obstruct more frequently used features, yet must be known and accessible to the users.

- Difficulty of evaluation - The obstacles to meaningful, generalizable analysis and evaluation of groupware prevent us from learning from experience. Task analysis, design and evaluation are much more difficult for multi-user applications than for single-user applications. Groupware often must interface simultaneously to users with different and sometimes shifting roles, preferences and backgrounds. Evaluation will take longer because group interactions unfold over days or weeks.

- Failure of intuition - Intuitions in product development environments are especially poor for multi-user applications, resulting in bad management decisions and an error-prone design process. Decision makers rely heavily on informed intuitions. Good intuition for multi-user applications is unlikely to be found anywhere in a product development environment.

- The adoption process - Groupware requires more careful implementation in the workplace than product developers have confronted. Products development have been isolated from user environments and have little awareness

that factors other than utility and usability govern a product's acceptance or rejection.

The problems listed call for better understanding of work environments and for corresponding adjustments by developers (Grudin, 1994). The first five points requires better knowledge of the intended users' workplace. The last three require changes in the development process.

Forming and maintaining a collaborative relationship is difficult and involves trade-offs, and give and take between colleagues (Cockburn & Jones, 1995). Cockburn & Jones argued that the inclusion of computer support in this complex balance will hinder group efficiency and cohesion. Cockburn & Jones (1995) ascribe the groupware failure to several factors which are listed below:

- Effort inherent in collaboration - The undesired aspects of collaboration are the overheads of effort beyond that required to execute personal work tasks.
- The effort of system requirements - Many system require additional effort from users in order to support their functionality.
- Effort imposed by lacking flexibility - Several systems have been based on explicit theories of co-operative tasks. Inflexible and constraining systems enforce a form of 'work to rule', a phrase synonymous with inefficient, restricted and inflexible working practices.
- Effort imposed by lacking integration - Effort is also required to manage the various tools, facilities and communication mechanisms that make up the work environment.
- Adoption and critical mass - Groupware tools are prone to a vicious cycle that restricts the realisation of system borne work enhancements.
    - Benefit and benefit-lag - Willingness to adopt a system is dependent on the benefits derived from its use. The longer the period of 'bene-

fit-lag', when effort outweighs the benefit, the higher the chance of user rejecting the system.

- Attainment of critical mass - Achieving critical mass depends on adoption by a sufficient group of individuals.
- Adoption by individuals - Personal use is most likely to be stimulated by personal benefits.
- The vicious cycle of adoption - Critical mass depends on adoption by individuals which is encouraged by benefits, but benefits are contingent on a critical mass of users.

There exist some similarity in the eight points presented by Grudin (1994). These include the issue of adoption and critical mass and the lacking of flexibility.

Design Strategies

These observations provide a foundation for the principles that assist designers in avoiding failure in system-design (Cockburn & Jones, 1995). These principles offer generic strategies to break the vicious cycle mentioned previously. These foundations as proposed by Cockburn & Jones (1995) are as follows:

- Maximise personal acceptance - This is concerned with encouraging individuals to adopt new systems. Some strategies includes:
  - Catch penny systems - A form of feature ticking could be used to supply instant user-appeal.
  - The 'Reflexive Perspective' - This perspective blurs the distinction between support mechanisms for personal and group work.

- Champions and encouragement - Groupware enthusiasm is greatly enhanced by 'champions' that promote the technology and raise awareness during initial system use.

- Minimise requirements - This is to reduce the disparity between groupware's costs and benefits to user-acceptable levels. Some of the strategies are as follows:

  - Avoid dependence on user actions - Systems that depend on users providing structured information encounter serious problems when guidance is absence.

  - Use what's available 'for free' - There are sources other than the users when guidance is required. These include e-mail headers to infer conversational relationships between e-mail messages and natural language parsing and keyword scans within text-based communication.

  - Enable shifts of cost and benefit - By shifting the provision of guidance onto users gaining the benefit, the cost-benefit disparity is reduced. Users execute additional actions when they are willing and able.

- Minimise constraints – This focuses on how systems retrieve the information they require. The aims is to avoid inflexible and constraining styles of use. This principle for minimised constraints argues for groupware that leaves the users free to develop protocols governing collaborative work as they, rather than their systems, see fit.

  - Be aware of two-level perspective of technology - There is a conflict between the increased efficiency enabled by computer support and the negative social implications. It is better to provide acceptable mechanisms providing some benefit than unacceptable ones.

- Beware of rigid models and theories - The lack of maturity and incomplete state of research in collaborative activity makes the use of 'universally applicable' explicit models in current groupware inappropriate.
- Open, unconstrained enhancement - This allows users to develop protocols governing collaborative work as they see fit.

- External integration - This attends to the user-effort that results from the changes between differing applications or communications environments.
  - Video fusion - Fusion allows separate video images to be overlaid. This enhances compatibility between personally favoured tools.
  - Heterogeneous environments - These augment collaboration by drawing together access to collaboration resources.
  - Minimise dependence on structure and format - System specific information formats and structures reduce the potential for integration.
  - Implementation platforms - Designers must either replicate some of the implementation to support variety of hardware or use an interface development application that can generate code for several graphical user interface environments.

Meanwhile, Grudin (1994) wrote that email demonstrates how important it is to adopt workplace perspective rather than technology perspective. Grudin proposed the following methods to help overcome the behavioural and social challenges facing groupware development and use:

- extend the use of single-user applications in group settings by adding groupware features; find niches where existing groupware succeeds

- build on object management or shared information system that have performed better than those that incorporate elements of organisational structure and work process
- find ways to provide direct benefits for all group members
- educate managers and developers about groupware, the risk involved and the resources and approaches that are required
- and we need better understanding of decision-making process in development.

Flexibility

The issue of flexibility was not only raised by Grudin (1994) and Cockburn & Jones (1995), but also by Glance *et al.* (1996). Glance *et al.* (1996) stated that among the setback of workflow solutions is the rigidity of their work representations. This rigidity arises partly from viewing work processes as unfolding along a single line of temporarily chained activities. In truth, work unfolds both horizontally, in the cooperation of causally unrelated, but information sharing tasks, and vertically, in the coordination of causally dependent activities (Glance *et al.*, 1996). Concurring with this, Robinson (1993) stated that applications that mediate between people are increasingly discovered to be used in ways that were not anticipated by their designers. Tools such as group editors and shared workspaces, do not dictate behaviour, but simply coordinate concurrent access and provide status information about other users' activities.

The nature of the working team also involves flexibility in terms of composition (Fafchamps & Garg, 1994). Fafchamps & Garg said that the variations in the composition of the flexible teams affect computing requirements.

Glance *et al.* (1996) argued that these raises the issue of how to design systems that embed flexible representations of work. Robinson (1993) asserted that work is best supported by the provision of resources. The nature of work procedures was put perfectly by Suchman (1983):

> *"While for computer scientist "procedure" has a very definite technical sense, for practitioners of office work the term has some other more loosely formulated meaning and usefulness. The distinction is something like that of a predetermined and reliable sequence of step-like operation versus an unelaborated, partial inventory of available courses and desired outcomes"*

Robinson (1993) emphasised that none of this means that organisational procedures, structures, roles, workplans, and objectives are without value. Local flexibility, articulation of work, and the shifting of requirements all happen in order to get the work done, within a framework of plans and objectives (Robinson, 1993). Robinson also noted that 'successful' applications seemed to allow two modalities of communication: natural, fairly unrestricted conversation, and communication via a system where actions were constrained by formation and transforming rules. As Robinson (1993) put it:

> *"Procedures are more like advice than algorithms."*

Trevor *et al.* (1993) also discusses about the drawback of existing environments. They stated that the environments are closed applications rather than open platforms and do not provide facilities to support a number of different approaches to cooperative work. Trevor *et al.* (1993) further argued that the problems of merely extending existing support models include: unrealistic models of the real-world, constraining models of control, lack of awareness of group, and limited support for sharing. Haake & Wilson (1992) further affirmed that both synchronous and asynchronous cooperation are extremely im-

portant aspects of working in groups. From this, they infer that to ignore one or the other is to supply only half a solution to the user.

Malone *et al.* (1992) stated that it is important to match the systems to the situations in which they are used when designing software to support groups of people working together. They proposed a 'radically tailorable' systems that allow end users to create a wide range of different applications by progressively modifying working system. The building blocs for the systems are: objects, views, agents, and links (Malone *et al.*, 1992).

Meanwhile, Kaplan *et al.* (1992a) argued that because of the nature of work activities that are heavily influenced by the surrounding which change from one place to the other, it is not possible to classify activities exactly. In order to address the issue of active support and the issue of flexibility, Kaplan *et al.* (1992a) suggest a system called the ConversationBuilder which has two mechanisms: protocols and obligations. Protocols allow different activity types and policies to be defined for the system. Obligations allow the user to dynamically weave individual activities together. The system aims to maximise both flexibility and active support. Kaplan *et al.* (1992a) stated that active support requires knowledge of the activity at hand, and defined flexibility as a measure of how well a system can respond to changes in activities.

Shared State

Another aspect that is important in the design of computer supported cooperative work system is the shared state. Shim *et al.* (1997) stated that the collaborators are able to work together by making changes to a shared state and observing the changes made by others. Ellis *et al.* (1991) define the shared state of applications in computer supported cooperative work as the following:

> *"a set of objects where the objects and the actions performed on the objects are visible to a set of users"*

Thus the management of a shared state may be defined as the management of actions, namely, accesses and updates, on the objects that constitute the shared state (Shim *et al.*, 1997).

Awareness

Information sharing, knowledge of group and individual activity, and the coordination are central to successful collaborations (Dourish & Bellotti, 1992). Information relating to these factors contributes towards the issue of awareness. Dourish & Bellotti (1992) define awareness as an understanding of the activities of others, which provides a context for your own activity. Dourish & Bellotti further stated that this context is used to ensure that individual contributions are relevant to the group's activity as a whole, and to evaluate individual actions with respect to group goals and progress. Awareness information is always required to co-ordinate group activities, whatever the task domain (Dourish & Bellotti, 1992). Awareness can be achieved by using one of three mechanisms. The first is an informational approach which provides explicit facilities through which collaborators inform each other of their activities. Second is a role restrictive approach which arises from explicit support for roles in collaborative systems. Dourish & Bellotti (1992) proposed a third approach called shared feedback. Shared feedback makes information about individual activities apparent to other participants by presenting feedback on operations within the shared, rather than the private, workspace.

Modes of Work

It is also important to consider a suite of groupware tools to support all modes of work (Olson & Teasley, 1996). When Olson & Teasley (1996) evaluates the technology currently available to the team members in one company and assessed the suitability to support the work flow, they found out that some of the technology is rarely or sometimes never used. Among these are the video conference room and desktop video for synchronous work conversation, and computer whiteboard work synchronous object sharing. Telephone and audio teleconference were used more often in synchronous work conversation. In asynchronous work, voice mail and e-mail are used in conversation, and a shared file server provides asynchronous object sharing.

3.2.3 Summary

In the topic of acceptability of CSCW, it is important to realised that it is affected by a number of issues. These include social and political factors (Grudin, 1994). Groupware can also fail because of the accumulation of individual rejection (Cockburn & Jones, 1995). Interaction of people with multiple goals also gives rise to different people preferring different problem solving strategies. In CSCW, the problems of trying to fit technology into the workplace are more serious. Cockburn & Jones (1995) stated that if groupware is more acceptable to individuals without hindering their value in group support, then they are likely to find it more acceptable.

Among the problems of CSCW design identified by Grudin (1994) are disparity in work and benefit, critical mass problems, disruption of social processes, exception handling, unobtrusive accessibility, difficulty of evaluation, failure of intuition, and the adoption process. Cockburn & Jones (1995) identi-

fied these problem as among the factors for groupware failure: effort inherent in collaboration, the efforts of system requirements, efforts imposed by lack of flexibility, effort imposed by lack of integration, and adoption and critical mass.

Cockburn & Jones (1995) proposed design strategies to overcome the problem mentioned previously. Among these are: maximise personal acceptance, minimise requirements, minimise constraints, and external integration. Grudin (1994) also proposed that the workplace perspective is adopted rather than the technology perspective.

The importance of flexibility in the design of groupware was also raised by Glance *et al.* (1996), Robinson (1993), Fafchamps & Garg (1994) and Kaplan *et al.* (1992a). Other related work in this issue is by Trevor *et al.* (1993), Haake & Wilson (1992), and Malone *et al.* (1992).

Other important aspects in the design of CSCW is the shared state (Shim *et al.*, 1997), awareness (Dourish & Bellotti, 1992), and modes of work (Olson & Teasley, 1996).

3.3 SOFTWARE ENGINEERING AND CSCW

Much of today's work is not done individually, but rather in groups (Olson *et al.*, 1993). Almost everyone is part of at least one group, typically several groups at any point in time (Nunamaker *et al.*, 1991). Olson *et al.* (1993) also mentioned that most real work is collaborative in nature, due to the complexity of the task, the severe time constraints, or the requirement for broad experience. Groups communicate, share information, generate ideas, organise ideas, draft policies and procedures, collaborate on the writing of reports, share a vision, build consensus, make decisions, and so on (Nunamaker *et al.*, 1991). It is also increasingly possible to consider forming work teams that are not physi-

cally collocated with the advent of networked technologies and groupware applications (Olson & Teasley, 1996). Hence, most cooperative systems are related to supporting or simulating such meetings. Such system include: desktop conferencing, electronic meeting rooms, group decision support, group problem solving, etc. (Beaudouin-Lafon, 1990). On the other hand, there exist other activities that involve communication between participants without such an evident notion of a group (Beaudouin-Lafon, 1990). Beaudouin-Lafon (1990) stated that software development is such an activity.

3.3.1 Software Development as a Group Activity

Studies by DeMarco & Lister (1987) suggest that on large projects, developers spend 70 percent of their time working with others, while Jones (1986) reports that team activities account for 85 percent of the cost of large software systems. As Sellars (1987) states:

> *"It is now well established that, for project teams, much of the available resource is consumed by the sheer effort of maintaining communication and control. Potential benefits to be obtained from a large number of individuals working in co-operation are rarely realised. If significant gains are to be made in the productivity of software developers then it is these synergistic properties of teams which must be tapped."*

The development of application may also be distributed, with teams in far-flung locations working on pieces of the same program (Knutson, 1996).

3.3.2 Groupware Support for Software Development

CASE tools currently on the market are designed for the single user (Vessey & Sravanapudi, 1995). Complex systems require the collaboration of many specialist working together over a period of time to create information-intensive products. Members of the work group share information, perform independent tasks, and create an identifiable, communal product. The capability to co-ordinate a group working on a single problem is critical. Vessey & Sravanapudi (1995) stated that CASE tools intended to support a work group in the system development process should address issues such as co-ordinating inter-dependent activities and project management.

Vessey & Sravanapudi (1995) developed a conceptual architecture for collaborative support technologies and they also developed exploratory models of the type of support needed in multi-user CASE tools. These models were used to develop an evaluation instrument to assess the effectiveness of CASE tools in supporting work groups. They stated that the majority of studies suggesting relationships between CSCW and the systems development process have adopted almost exclusively the notion of using the electronic meeting room concept to facilitate face-to-face activities such as joint application design sessions, design sessions and design reviews. The distinction between support for meetings, per se, and support for specific work groups is the need to share raw materials and work products.

The conceptual architecture proposed is based on task specificity (Vessey & Sravanapudi, 1995). The most basic support is taskware, which is for carrying out the task itself. The next level of support is teamware, which is for group working which allows sharing of raw materials and work products. The third level support communication explicitly between group members. According Vessey & Sravanapudi (1995), teamware, which is the second level

support, is the least developed and least known. An exploratory analysis was conducted on four network version of CASE tools. The results that Vessey & Sravanapudi (1995) found are as follows.

- Few of the control facilities were implemented.
- Information sharing was better supported.
- Only a few monitoring capabilities were available.
- The CASE tools also provided only a few of the cooperation features desired.

Beaudouin-Lafon (1990) contended that based on their experience with software development by small groups, there is a need for some specific tools to support the cooperative aspects of the development process. Beaudouin-Lafon further listed three tasks in software development that could benefit from co-operative support. These are: editing, debugging, and message systems. The model for collaborative editing proposed is called 'embedded context-dependent version model'. This model includes two tools, namely the 'X Television', which is designed to develop centralised graphical multi-user applications, and the 'Unix Channel', which is dedicated to distributed applications (Beaudouin-Lafon, 1990).

The more people are involved, the more important becomes the collaboration and communication between individuals (Leonhardt *et al.*, 1995). The different participants will have different views on and assumption about the problem domain. Leonhardt *et al.* (1995) stated that this necessitates organised interaction including conflict detection and resolution. Leonhardt *et al.* further stated that in a decentralised data storage and consistency control, conflict detection and resolution have to be based on interaction and local knowledge about the system. They proposed a fine-grained, decentralised model which can be used to drive conflict detection and resolution. This model is not intended to automate the development process but rather serves as a guide to

the developer (Leonhardt *et al.*, 1995). The model initiates and monitors consistency checks in order to gain knowledge about the system under development. These consistency checks are the prime means of coordination.

In an earlier work, Narayanaswamy & Goldman (1992) put forward a 'lazy consistency' method to maintain certain global consistency properties in cooperative software development. In this architecture, the announcements, which are a notification of changes, deal with impending and proposed changes as well as changes that have already occurred. The process consists of intra-step consistency, where programmers debated proposed change notification, and inter-step consistency, where collisions arise between steps at the level of proposed changes (Narayanaswamy & Goldman, 1992).

In another study that deals with the issue of concurrency control problems, Barghouti (1992) presented a mechanism where the encoding of the software development process in process-centred software development environments can be used to provide more appropriate concurrency control. The approach is to extract semantics about concurrent interactions and consistency constraints from the process model, and then use such semantics to provide flexible and extensible concurrency control (Barghouti, 1992).

Rodden (1990) proposed a mechanism for supporting team members in a software development team to reduce the overhead generated by their cooperation. The mailtray system introduced by Rodden (1990), consists of three mechanisms:

- the mailtray metaphor, which unifies the automatic structuring of messages to minimise information overload with the automatic handling of routine information.
- an object based register, which allows destinations to be inferred from an objects attribute values.

- the linking of trays to form conversations.

Twidale *et al.* (1993) proposed a system to support cooperative software design. They developed the 'Designers' Notepad', which is based on an iterative approach to development. The system addressed two tensions that exist in the design process. These are as follows: the private versus communal nature of design, and freedom of expression versus formalisation of shared understanding (Twidale *et al.,* 1993). Grinter (1997) examines a workflow-like system that helped reduce the complexity of work in a way that was helpful to those using the system. Grinter studies the role of configuration management tool in supporting that work. The workflow system works in this instance because of: understanding and accepting the model of work, understandable and useful representations, automating the 'right' work, and a supporting company (Grinter, 1997). ConversationBuilder, which was furnished by Kaplan *et al.* (1992a) is also a basis for support of collaborative software development activities. The system is an open, flexible, and active support environment for software development (Kaplan *et al.*, 1992b).

3.3.3 Summary

Olson *et al.* (1993) stated that much of today's work is done in groups rather than individually. Almost everyone is part of at least one group, typically several groups at any point in time (Nunamaker *et al.*, 1991). On the other hand, software development is an activity that involves communication between participants without such an evident notion of a group (Beaudouin-Lafon, 1990). It is also increasingly possible to consider forming work teams that are not physically collocated with the advent of networked technologies and groupware applications (Olson & Teasley, 1996). Similarly, Knutson (1996) stated that

the development of application may also be distributed, with teams in far-flung locations working on pieces of the same program.

Vessey & Sravanapudi (1995) said that CASE tools currently are designed for single users. They developed a conceptual architecture for collaborative support technologies and they also developed exploratory models of the type of support needed in multi-user CASE tools. These models were used to develop an evaluation instrument to assess the effectiveness of CASE tools in supporting work groups. Beaudouin-Lafon (1990) also stated that there is a need for some specific tools to support cooperative aspects of the development process.

Leonhardt *et al.* (1995) and Narayanaswamy & Goldman (1992) work in the area of consistency aspects of cooperative software development. Leonhardt *et al.* (1995) proposed a fine-grained, decentralised model which can be used to drive conflict detection and resolution. Earlier, Narayanaswamy & Goldman (1992) put forward a 'lazy consistency' method to maintained certain global consistency properties in cooperative software development. In another study, Barghouti (1992) presented a mechanism where the encoding of the software development process in process-centred software development environments can be used to provide more appropriate concurrency control. Rodden (1990) proposed a mechanism for supporting team members in a software development team to reduce the overhead generated by their cooperation by using the mailtray system. Twidale *et al.* (1993) proposed the 'Designers' Notepad', which is based on an iterative approach to development, to support cooperative software design. Grinter (1997) examines a workflow-like system that helped reduce the complexity of work in a way that was helpful to those using the system. ConversationBuilder, by Kaplan *et al.* (1992a), is also a basis for support of collaborative software development activities.

3.4 GROUPWARE SUPPORT FOR SOFTWARE INSPECTION

Groupware support is not only confined to the areas of software engineering mentioned in the previous section. Support for the software inspection process is also available and it includes: Collaborative Software Review System as proposed by Johnson & Tjahjono (1993), Collaborative Software Inspection as forwarded by Mashayekhi *et al.* (1993), and Scrutiny as outlined by Gintel *et al.* (1993). The following sections discusses each systems in a more detailed manner.

3.4.1 Collaborative Software Review System

Collaborative Software Review System (CSRS) was designed to support the Formal Technical Asynchronous review method (FTArm) (Johnson & Tjahjono, 1997, 1993; Johnson *et al.*, 1993). FTArm is a development based on the Fagan model of inspection. CSRS is implemented on top of a multi-user, distributed, hypertext-based collaborative environment (Johnson & Tjahjono, 1993; Johnson *et al.*, 1993). It is designed to work with an incremental model of software development and exploits the use of an on-line, collaborative environment for review with automatic metrics collection. There are six phases in this model. In set-up, members of the team is chosen and the document to inspect is prepared. This is followed by orientation, which is equivalent to overview in Fagan inspection. In private review, the inspector reads each source node privately and creates issues and comment nodes. A node is a hypertext-style database (Johnson & Tjahjono, 1993; Johnson *et al.*, 1993). In the public review phase, the team react to all generated issue and action nodes. This phase concludes when all issues, action, and evidence nodes have been marked as reviewed by all team members (Johnson & Tjahjono, 1993; Johnson *et al.*, 1993). What follows

is the consolidation phase, where the moderator writes a report including summary of the comment. The last phase is the group review meeting to resolve any continued controversy, if any.

3.4.2 Collaborative Software Inspection

Collaborative Software Inspection (CSI) is based on the Humphrey (1989) model of inspection, which was discussed in Chapter Two. It is designed to support inspection of all software development products in a distributed environment (Mashayekhi *et al.*, 1993, 1994). Each inspector creates a list of faults during individual preparation and then the list is given to the author of the document before the meeting. Annotation during individual preparations is supported by creating hyperlinks between lines of documents and the reviewer's annotations (Mashayekhi *et al.*, 1993). The author will correlate the list and address it during the inspection meeting. The inspection meeting is guided by the author using the correlated list. The system is intended for synchronous distributed using an additional teleconferencing tool called Teleconf. Teleconf provides the audio for the meeting. The system contains eight components, namely: browser, annotation, note pad, action list, inspection summary, criteria, fault list, and history log (Mashayekhi *et al.*, 1993).

A different system called Collaborative Asynchronous Inspection Software (CAIS) allows the participants to inspection asynchronously (Mashayekhi *et al.*, 1994). CAIS consists of three objects: browser, history log, and meeting object. CAIS meeting conversation consists the following phases: annotations, discussions, and proposal. Voting is conducted on the proposal. CAIS was extended further to support world wide web based tool. The Asynchronous Inspector of Software Artefact (AISA) supports inspection of textual and graphical artefacts (Stein *et al.*, 1997).

3.4.3 Scrutiny

Scrutiny's inspection process consists of four stages: namely, initiation, preparation, resolution, and completion (Gintel *et al.*, 1993). The initiation is comparable to overview in the Fagan model and the preparation is the same. The inspection process is called the resolution and the completion combined both rework and follow-up. In Scrutiny, the role of the reader is merged into the role of moderator. There is also a producer to answer questions with regards to the document and a verifier to ensure that defects found are addressed by the author. Scrutiny can be used for same-place and distributed synchronous inspection. No audio facilities are supported by Scrutiny.

3.4.4 Summary

Works on groupware support for software inspection by Johnson & Tjahjono (1993), Mashayekhi *et al.* (1993), and Gintel *et al.* (1993) were described in this section. Basically, there are two types support that the tools offer, either the distributed synchronous or distributed asynchronous mode.

CSRS is a distributed asynchronous system (Johnson & Tjahjono, 1993). It is designed to work with an incremental model of software development and exploits the use of an on-line, collaborative environment for review with automatic metrics collection. CAIS also supports distributed asynchronous inspection (Mashayekhi *et al.*, 1994). CAIS was extended on the web as AISA but still only support only distributed asynchronous (Stein *et al.*, 1997).

CSI supports distributed synchronous inspection and is based on the Humphrey's model of inspection (Mashayekhi *et al.*, 1993). Scrutiny is loosely based on Fagan's version of software inspection and supports same time-same place and distributed synchronous inspection (Gintel *et al.*, 1993).

3.5 SUMMARY OF CHAPTER

It was mentioned earlier that the main idea in the classification of CSCW that is of interest to this research is the group work design aspects and the group dynamics aspects in the group working process, and research in the shared work space systems, shared information systems, and group activity support systems in the enabling technologies area as proposed by Wilson (1990, 1991). The space and time taxonomy of groupware (Johansen, 1988; Ellis *et al.*, 1991) provides an important method in analysing the working mode of a group.

In designing CSCW systems, it is useful to consider why they might fail. The acceptability of CSCW is affected by social and political factors (Grudin, 1994). Groupware can also fail because of accumulation of individual rejection (Cockburn & Jones, 1995). The problems of trying to fit technology in to the workplace are more serious in CSCW. The problem of CSCW design identified by Grudin (1994) are disparity in work and benefit, critical mass problems, disruption of social processes, exception handling, unobtrusive accessibility, difficulty of evaluation, failure of intuition, and the adoption process. Cockburn & Jones (1995) identified the following problems as among the factors for groupware failure: effort inherent in collaboration, the efforts of system requirements, efforts imposed by lacking flexibility, effort imposed by lacking integration, and adoption and critical mass.

Among the design strategies proposed to overcome these problems are: maximise personal acceptance, minimise requirements, minimise constraints, and external integration as proposed by Cockburn & Jones (1995), and adoption of the workplace perspective rather than the technology perspective as proposed by Grudin (1994). The importance of flexibility in the design of groupware was also raised by Glance *et al.* (1996), Robinson (1993), Fafchamps & Garg (1994) and Kaplan *et al.* (1992a). Other important aspects in the design

of CSCW are the shared state (Shim *et al.*, 1997), awareness (Dourish & Bellotti, 1992), and modes of work (Olson & Teasley, 1996).

Vessey & Sravanapudi (1995) said that CASE tools currently are designed for single user. Beaudouin-Lafon (1990) stated that there is a need for some specific tools to support cooperative aspects of the development process.

Work on groupware support for software inspection by Johnson & Tjahjono (1993), Mashayekhi *et al.* (1993), and Gintel *et al.* (1993) was discussed. All the works reviewed either only support distributed synchronous mode or only support distributed asynchronous mode.

In summary, *Section 3.1* discussed introduction to CSCW. Issues related to the classification of CSCW and its early development were reviewed. The classification of CSCW according to Ellis *et al.* (1991) was discussed. *Section 3.2* surveyed the acceptability and design issues of CSCW. Among the topics that were reviewed were the problem in design, design strategies, and the issue of flexibility. *Section 3.3* looked into the relationship between software engineering and CSCW. Examples of groupware support for software development process were given. *Section 3.4* narrowed down further to the topic of groupware support on software inspection. Groupware tools that support software inspection were presented. Topics of research design and procedures are presented in the next chapter. The literature that has been reviewed serves as the basis for the design. A general review of enabling technologies for the research is also provided in the next chapter.

Chapter 4

RESEARCH FORMULATION, DESIGN AND PROCEDURES

4.0 INTRODUCTION

This chapter examines the subject of literature review synthesis and the design of the research. The topics consist of the following:

- **Section 4.1** identifies the research problem based on the review of literature.
- **Section 4.2** presents the purpose and the objectives, and the importance of this research.
- **Section 4.3** discusses the research design and procedures. Discussion includes research methodology and design, assumptions, and limitations of this research.

4.1 FORMULATION OF THE RESEARCH PROBLEM

This research is concerned with the software inspection process. The issue of flexibility and acceptability of CSCW systems are also of interest to this research. This section formulates the research problem based on the review of literature covered in previous chapters. The shaping of the research problem is

guided by the material in the review of literature. This section serves as a linkage between the literature and the research problem. This section will discusses the design of the model and the tool for software inspection process based on the limitations of existing models and software inspection tools.

4.1.1 Importance of Software Inspection

Software quality is an important goal of software engineering. Software quality is achieved through a proper management system, which includes QC, QA, and CMM, and other methods, namely, verification and validation, and software testing.

One aspect that all these techniques have in common is that they make use of the software inspection process as part of each technique. The importance of inspection techniques is further demonstrated when fourteen out of the eighteen major software quality related standards presented by Schulmeyer (1987b) include code reviews as part of the standard. Thus, it can be concluded that inspection techniques plays an important role in achieving better quality software.

Software inspections aim to analyse a product or document to detect defects (Fagan, 1976). The importance of inspection techniques in software quality is also well supported by the documentation of the benefit of using inspection (Fagan, 1976; Ackerman, 1984; Kitchenham *et al.*, 1986; Humphrey, 1989; Schulmeyer, 1990; Russell, 1991; Olson, 1993, Grady, 1993). Despite the importance and the benefit of inspections, the practice in the real-world is difficult (Macdonald *et al.*, 1996).

4.1.2 The Drawbacks of Software Inspection

The benefit of software inspection is well accepted. Nevertheless, there are many factors that can affect the costs, benefits, and acceptability of software inspection implementation.

Comment preparation and the code inspection meeting in a manual inspection process are the most difficult and time consuming phases (Brothers *et al.*, 1990). An organisation may be reluctant to devote the necessary investment in time and money, especially when the project is behind schedule (Macdonald, 1996). Software inspection also carry the tag of a 'low-tech' image process (Russell, 1991). The placement of the inspection as a 'hurdle' to be 'jumped' is also not effective in the era of rapid prototyping and incremental development (Johnson & Tjahjono, 1993). Small projects traditionally have no formal peer review of work, although there exist an informal reviews for specifications, designs, and code (Cousin, 1993). The variations in types or names with regards to inspections and reviews techniques also contributed to this problem (Freedman & Weinberg, 1990).

4.1.3 Software Inspection as a Group Activity

Software inspection is a group activity. Beaudouin-Lafon (1990) stated that in software development, communication between participants occur without such an evident notion of a group. Forming work teams that are not physically collocated is possible with the advent of networked technologies and groupware applications (Olson & Teasley, 1996). In the same way, the development of application may also be distributed, with teams in far-flung locations working on pieces of the same program (Knutson, 1996).

4.1.4 CSCW for Software Inspection: Design Problem and Strategy

The importance and the problem of software inspection presented in the previous section is only half the picture. The design of a CSCW system for software inspection not only requires software inspection considerations, but also the consideration of requirements of good CSCW tools.

Shortcomings in Design of CSCW

The problems in the design of a CSCW systems range from technology to social issues. Early groupware systems suffered from acceptability problems.

One of the problems is the critical mass problems as presented by Grudin (1994). Adoption and critical mass problems were also highlighted by Cockburn & Jones (1995). Grudin (1994) also stated that disruption of social processes is another problem. Cockburn & Thimbleby (1991) said that anything that changes or adjusts the way people work in groups is not easily tolerated. Features that support group processes that are used relatively infrequently require unobtrusive accessibility and integration with more heavily used features (Grudin, 1994). Effort is required to manage the various tools, facilities and communication mechanisms that make up the groupwork environment (Cockburn & Jones, 1995).

The problem of lack of flexibility was raised by Cockburn & Jones (1995). They said that inflexible and constraining systems enforce a form of 'work to rule', a phrase synonymous with inefficient, restricted and inflexible working practices. Similarly, Glance *et al.* (1996) stated that among the setback of workflow solutions is the rigidity of their work representations. Trevor *et al.* (1993) also stated that current environments do not provide facilities to

support a number of different approaches to cooperative work. Works often switches between synchronous and asynchronous interactions (Blair & Rodden, 1994) and work also unfolds both horizontally and vertically (Glance *et al.*, 1996). Referring to the synchronous and the asynchronous interaction, Haake & Wilson (1992) stated that to ignore one or the other is to supply only half a solution to user. Grudin (1994) mentioned that an organisation will not re-structure itself for each new application and a small application program must adapt to the organisation. Robinson (1993) emphasised that none of the problems mentioned in this section mean that organisational procedures, structures, roles, workplans, and objectives are without value.

Design Strategy

From the previous section, one of the biggest problems is the flexibility issue. The design strategy proposed in this section provides the foundation that assists in avoiding failure in system design.

Cockburn & Jones (1995) proposed the 'maximise personal acceptance' strategy which sought to encourage individuals to adopt new systems. One of the strategies includes the 'reflexive perspective' which blurs the distinction between support mechanisms for personal and group work.

The 'minimise constraints' strategy focuses on how systems retrieve the information they require (Cockburn & Jones, 1995) and to avoid inflexible and constraining styles of use. Models and theories should not be rigid. The lack of maturity and incomplete state of research in collaborative activity makes the use of 'universally applicable' explicit models in current groupware inappropriate (Cockburn & Jones, 1995). Open, unconstrained enhancement allows users to develop protocols governing collaborative work as they see fit.

Cockburn & Jones (1995) also proposed the 'external integration' strategy, which attends to the user-effort that results from the changes between differing applications or communications environments. In the strategy, support for heterogeneous environments is needed to augment collaboration by drawing together access to collaboration resources. Designers must either replicate some of the implementation to support a variety of hardware or use an interface development application that can generate code for several graphical user interface environments.

Meanwhile, Grudin (1994) proposed that groupware features will fare better if integrated with features that support individual activity, and infrequently used groupware features must not obstruct more frequently used features, yet must be known and accessible to the users.

To get the work done, local flexibility, articulation of work, and the shifting of requirements should be allowed within a framework of plans and objectives (Robinson, 1993). Robinson noted that 'successful' applications seemed to allow two modalities of communication: natural, fairly unrestricted conversation, and communication via a system where actions were constrained by formation and transforming rules.

In another proposal, Shim *et al.* (1997) stated that the collaborators are able to work together over a distance by making changes to a shared state and observing the changes made by others.

The issue of awareness, which includes information sharing, knowledge of group and individual activity, and coordination, are central to successful collaborations (Dourish & Bellotti, 1992). The strategy can be achieved by using one of three mechanism: an informational approach which provides explicit facilities through which collaborators inform each other of their activities, a role restrictive approach which arises from explicit support for roles in collabo-

rative systems, and shared feedback which makes information about individual activities apparent to other participants by presenting feedback on operations within the shared, rather than the private, workspace (Dourish & Bellotti, 1992).

4.1.5 Applying the Strategy to the Software Inspection Process

From the material presented in the previous few sections, it can be inferred that the software inspection process is susceptible to a number of problems. The problems are directly related to the nature of software inspection process itself and also the design of CSCW system for software inspection.

The problems that are related directly to the software inspection process and to groupware support for software inspection have been researched quite extensively. The work by Brothers *et al.* (1990), Mashayekhi *et al.* (1993, 1994), Stein *et al.* (1997), Gintel *et al.* (1993), Knight & Myers (1993, 1991), and Macdonald (1996) proposed several solution to the problems.

On the other hand, the problems of acceptability of CSCW design for software inspection process are less well researched. Thus, the issues of critical mass, disruption of social processes, unobtrusive accessibility, effort imposed by lacking integration, and lack of flexibility in software inspection process have not been given the same attention as other problems. Consequently, the current models and tools that support software inspection lack support for these issues. The application of strategy to avoid failure in groupware systems designed for the software inspection process has also received less attention.

4.1.6 Design of Model

In order to develop a software inspection model that handles the issues mentioned in the previous section, there is a need to evaluate the strengths and limitations of existing models.

	Explanation
Processes	Planning, Overview, Preparation, Inspection, Follow-up, and Rework
Roles	moderators, authors, readers, and tester
Number of Participants	Four (can be change)
Number of Hours	Two hours per session (Total hours: 90-100 people hours)
Other Information	Analysis is also conducted in order to get feedback, feedforward and self improvement. Analysis of the inspection process is conducted and is passed as a feedback, feedforward, and self improvement tool.

Table 4 - 1 : *Fagan's Inspection Model (Fagan, 1976, 1986)*

Existing Software Inspection Models

Since the inspection process was first presented by Fagan (1976), there have been several other variants of the inspection process proposed by different researchers and practitioners. *Table 4 - 1*, *Table 4 - 2*, and *Table 4 - 3* briefly summarise each major model based on the description presented fully in Chapter Two.

	Explanation
Processes	Request, Entry, Planning, Kick-off Meeting, Individual Checking, Logging Meeting, Edit, Follow Up, Exit, and Release
Roles	Inspection Leader, Author, and Checker. Scribe is appointed during meeting
Number of Participants	2-3 (max efficiency) or 4-5 (max effectiveness)
Number of Hours	Not exceeding two hours per meeting session
Other Information	The model contains another phase which is called process improvement. Every issue recorded in the logging meeting is forwarded to the process change management team. This will contribute to the overall improvement of the development process

Table 4 - 2 : *Gilb & Graham's Inspection Model (Gilb & Graham, 1993)*

	Explanation
Processes	Preparation, which consists of Entry criteria and Opening Meeting, Inspection, and Post-Inspection Activity
Roles	Moderator, Producers, Reviewers, Recorders
Number of Participants	Should not exceed 5 or 6 persons
Number of Hours	Not exceeding two hours per session
Other Information	Recommend the upper limit on the size of product to be inspected should not exceed 500 Line of Code

Table 4 - 3 : *Humphrey's Inspection Model (Humphrey, 1989)*

Limitations and Strengths of Existing Models

The existing models contain their own limitation and strengths. Among the limitation of existing models are as follows:

- The three models presented above are for a formal manual inspection process. They are for a face-to-face software inspection process and distributed inspection is not supported. A formal meeting must be organised to conduct the inspection (Fagan, 1976, 1986; Humphrey, 1989; Gilb & Graham, 1993).

- The models are also not flexible in terms of the type of working mode supported. Only face-to-face meeting is supported (Fagan, 1976, 1986; Humphrey, 1989; Gilb & Graham, 1993). No support for distributed synchronous or distributed asynchronous inspection is provided.

- The models are designed for big projects in big organisations (Fagan, 1976, 1986) and mostly are part of a bigger process model (Humphrey, 1989). Cousin (1993) stated that mostly, there is only informal reviews in small projects.

- The models are more suitable to the traditional waterfall lifecycle and not compatible with incremental development methods (Johnson *et al.*, 1993). The model by Johnson *et al.* (1993), which was proposed with CSRS, is the only one designed for incremental development, but it supports distributed asynchronous inspection only.

On the other hand, the existing models do have strengths, which are listed below:

- The models are proven to be effective in improving quality (Fagan, 1976, 1986; Grady, 1993) provided that the organisation is willing to invest the time and money needed for the inspection process (Macdonald, 1996).

- The formal inspection process model is a well-defined process with proper description on the functions (Gilb & Graham, 1993) and has been used succesfully since the seventies (Fagan, 1976).

- The process includes process improvement (Gilb & Graham, 1993), analysis (Fagan, 1976, 1986), and metrics (Humphrey, 1989) to provide feedback for overall improvement of the project.

Extensions of Software Inspection Models

The research aim to provide an extension that can resolve the limitations while maintaining the strengths of existing models. The model also takes into consideration the design problems of CSCW systems and the strategies to solve these problems.

Among the extensions planned is addition of flexibility to the model of the software inspection process. The extension of the model seeks to provide added flexibility in terms of space-time matrix, and offer choices in the working mode of the process, allowing not only a formal software inspection process to take place, but also an informal inspection, if there is any need. The flexibility allows distributed synchronous or distributed asynchronous inspection to occur, alongside the formal face-to-face inspection process. The extension of the model draw the strengths of the existing model by maintaining most of the processes and roles they provide.

4.1.7 Design of Tool

In order to provide a flexible software inspection process model, a CASE tool needs to be designed. The tool not only supports the extended model, but also provides additional support to solve the problems in CSCW design. This section will outline the design consideration for software inspection tools and analyse the limitations and strengths of existing tools. A proposal for the extensions is provided.

Design Consideration

The design considerations for a software inspection tools are heavily based on the design considerations of a CSCW system, and are aimed at strong user acceptability of the system. Apart from the design considerations of CSCW for software inspection, the tool must include the minimum support expected from any inspection tool.

The acceptability of the system depends on the flexibility of the system, as pointed by Cockburn & Jones (1995). One aspect of the flexibility is support for different modes of working mode (Johansen, 1988). The inspection tools should support not only face-to-face inspections, but also distributed synchronous and distributed asynchronous inspection. The importance of support for different approaches was also pointed out by Trevor *et al.* (1993).

The tool should support more than one software inspection model. This is another aspect of increasing flexibility of the system. The ability to support multiple models allows the tool to be more acceptable because organisations will not restructure themselves for each new application (Grudin, 1994).

The tool should be able to run on multiple computing platform. The heterogeneous support is another important factor to increase acceptability. Heterogeneous support is also important in order to improve external integration with other tool (Cockburn & Jones, 1995).

In order to reduce the problem of critical mass (Grudin, 1994; Cockburn & Jones, 1995), the implementation of the tool must utilise a technology that is already widespread. By utilising a technology that is already reached the critical mass, the chances of the tool being accepted are improved further.

The least used group support processes should have unobtrusive accessibility (Grudin, 1994). Software inspection tool must ensure that least used

group support processes do not interfere with frequently used features of the group support.

The software inspection tool should also provide support for shared state (Shim *et al.*, 1997) to increased support for informal and indirect communication. Shared state allows the members of the inspection team to view changes from different geographical locations.

Another type of support for informal and indirect communication is the awareness features. The team awareness of members in a distributed inspection is central to successful collaborations (Dourish & Bellotti, 1992). Awareness in a distributed inspection can be achieved via information sharing and knowledge of group and individual activity.

The inspection tool should also try to reduce disruption of social process when introduced into an organisation (Grudin, 1994). This is an indirect benefit from several of the features supported above. The flexibility and the critical mass factor will ensure that the social process in the group is not disrupted by the introduction of an inspection tool.

Software inspection tool should support document handling (Macdonald, 1996). The documents that need to be supported in an inspection process range from the inspected document to other supporting document, namely specification documents, inspection objectives, rules, reports and metrics.

Individual preparation should also be supported by software inspection tool (Macdonald, 1996). The individual inspection support must include a browsing component, access to supporting documents, communication facilities if the need arises, and logging facilities.

Another aspect that is essectial is a tool for meeting support (Macdonald, 1996). The support that must be included are a browsing facility, log-

ging facility, voting support, remote browsing component, communication suite, and access to supporting documents and a help facility.

Software inspection tool should also support data collection (Macdonald, 1996). The metrics collected should be tailored for the author of the document, the software inspection management, and the quality management.

Existing Software Inspection Tools

CASE tools were developed in order to achieve quality (Humphrey, 1991) and tools that support the software quality process are categorised as 'support tools' (Pressman, 1992). But CASE tools currently are designed for a single user (Vessey & Sravanapudi, 1995) and there is a need for some specific tools to support cooperative aspects of the development process (Beaudouin-Lafon, 1990).

Not only is software inspection a group activity, but it is possible now for the process to be distributed. Current inspection tools support document handling, individual preparation, meeting support, and data collection. Among the first tools that support software inspection are the ICICLE (Brothers *et al.*, 1990) and InspeQ (Knight & Myers, 1993, 1991). Both tools did not support distributed inspection, in fact, InspeQ is meant for one user only.

ICICLE document handling is controlled by a window displaying the code with line number and two symbols referring to the comment (Brothers *et al.*, 1990). Support for individual preparation is provided by automated intelligent inspection and also manual inspection. Distributed meetings are not supported, although the reader can lock other team member windows to the reader's view and guide the meeting. A list of the defects, a summary of the defects, and a report revealing the time spent are generated.

InspeQ did not include any support for meeting and only provides minimal data collection support in the form of a list of comments. The tool supports document handling by allowing the inspectors to browse multiple copies of the document and record any issues found. Individual preparation is supported by a checklist display associated with current inspection and standard displays to show each standard in full (Knight & Myers, 1993, 1991).

Different Place	CSI Scrutiny	CAIS CSRS AISA
Same Place	ICICLE Scrutiny	
	Same Time	Different Time

Table 4 - 4 : *Time-Space Comparison between Inspection Tools*

The support for the distributed software inspection process can be categorised either as distributed synchronous or distributed asynchronous.

CSRS was designed to work with an incremental model of software development and to exploit the use of an on-line, collaborative environment for review, with automatic metrics collection (Johnson *et al.*, 1993). Document and annotation are stored as nodes. A node is a hypertext-style database. Individual inspection support includes a summary of which nodes that have been covered and which have not been covered. A voting mechanism is available to make a decision on each node. No synchronous meeting support is available. List of defects, time spent, and event log are produced as a reports.

The CSI system is designed to support inspection of all software development products in a distributed environment (Mashayekhi *et al.*, 1993, 1994). A browser is provided to view the document complete with line number. The line numbers provide hyperlinks to annotation windows. An on-line criteria for the inspection is provided as a support to individual preparation. A correlated single fault list is produced and is used by the author to guide the meeting. An action list is produced as a product of the discussion. An additional teleconferencing tool called Teleconf provides the audio for the meeting. Reports produced by the tools includes an inspection summary and a history log.

A different system, CAIS, allows the participants to conduct inspection asynchronously (Mashayekhi *et al.*, 1994). CAIS concentrates on the meeting support of software inspection only (Mashayekhi *et al.*, 1994). The discussion in the tool is sequence of comments and terminated by vote-taking.

CAIS was extended by Stein *et al.* (1997) to support a world wide web based tool and is called AISA. AISA supports inspection of textual and graphical artefacts (Stein *et al.*, 1997). In the document handling, the document is converted into a node known to the web server. The individual preparation is supported by allowing the user to use the home page to view the document. The faults submitted by the inspector are correlated by the producer. There is no support for synchronous meeting. A fault can be viewed, commented on, and voted on at the end of the asynchronous discussion. The metrics collected by the tool includes fault collection, fault correlation, and inspection meeting metric.

Scrutiny can be used for same-place and distributed synchronous inspection (Gintel *et al.*, 1993). The document handling includes the work product window to view the document and the annotation window for the inspectors to enter different types of annotation. There is no support for checklists or other supporting documentation in individual preparation. The meeting is

supported by the control window and the product window. Reports generated includes list of defects, summary information, and time spent by team members.

Limitations and Strengths of Existing Tools

The limitations and the strengths of existing tools are analysed in relation to the design consideration of software inspection tools. The limitations of existing tools are as follows:

- None of the tools support the flexibility (Cockburn & Jones, 1995) to allow different modes of working (Johansen, 1988).
- No software inspection tools provide the ability to support multiple inspection models. CSRS (Johnson, 1994) does support customisation in the form of process modelling language.
- The majority of the tools do not run on multiple computing platforms. Only AISA (Stein *et al.*, 1997) provides support for multiple computing platforms.
- No tools, except AISA (Stein *et al.*, 1997), utilise technology that has already reached the critical mass (Grudin, 1994; Cockburn & Jones, 1995).
- None of the software inspection tools addressed the issue of unobtrusive accessibility (Grudin, 1994).
- There is no mentioning of the tools supporting shared state (Shim *et al.*, 1997) to increase support for informal and indirect communication.
- Again, support for awareness in a distributed inspection is not mentioned explicitly by the tools. Although there is certain support for information sharing, knowledge of group, and individual activity, it is not mentioned explicitly as a support for informal and indirect communication.

- Meeting support (Macdonald, 1996) is only available on the tools that support distributed synchronous inspection. Distributed synchronous meeting is supported by CSI (Mashayekhi *et al.*, 1993) and Scrutiny (Gintell *et al.*, 1993) only.

The strength of the tools includes:

- The support by all the existing tools of document handling (Macdonald, 1996) is good.
- The support for individual preparation (Macdonald, 1996) is generally good. ICICLE (Brothers *et al.*, 1990) provides intelligent support for inspection. Most of the tools provides supporting documentation facilities, except Scrutiny (Gintell *et al.*, 1993).
- All the software inspection tools provided support for data collection (Macdonald, 1996).

Extension of Software Inspection Tools

The inspection tool extension aims to support the design consideration mentioned earlier and improve on the limitations of existing tools. The extended tool will support the extended model which includes the strength of existing model while providing new flexibility. Among other extensions that are included are the ability to support multiple inspection models, support for multiple computing platform, use of technology that has already reached the critical mass, provision of unobtrusive accessibility to the least used group support processes, and provision of shared state and awareness support to allow better informal and indirect communication.

4.2 RESEARCH PROBLEM

This section outline the aim of this research under the 'statement of the problem' heading. This is followed by the purpose, the objective, and the importance of this research.

4.2.1 Statement of the Problem

The problem identified is that existing models and tools are not sufficiently flexible. Therefore, the aim of this research is to design and build a groupware system which will allow members of a distributed group more flexibility in performing software inspection.

The groupware system aims to provide a system that will improve acceptability of groupware and improve software quality by providing a software inspection tool that is flexible and adaptable. The groupware system provides a flexible structure for software inspection meetings.

4.2.2 Purpose of Study

The purpose of this study is to develop a Flexible Software Inspection Groupware (FlexSIG) system which includes:

- the development of a flexible software inspection process model based on literature review
- an implementation of a prototype based on the model of the system
- the evaluation of the prototype in order to measure the acceptability of the system.

What flexible means in this research is as follows:

- The ability to allow the software inspection process to use all four quadrants of the Johansen (1988) space-time matrix or the four categories of the time and space taxonomy proposed by Ellis *et al.* (1991)
- The ability to use the tool across a distributed network on top of different computer platforms running different operating system. The tool develop is aimed to run on the internet or an intranet.
- The ability to let the user choose the software inspection process model that fits their requirements and style. The tool aims to support both the Fagan's and Humphrey's model. The Gilb & Graham's model can also be supported with minor modification.

As mentioned in the formulation of the research problem, the flexibility offers new working possibilities. It will not force the user or the group to change their way of working, and a system that can be adaptable to a more flexible pattern of a group of users is likely to be more acceptable.

A model of FlexSIG is constructed based on the literature review. A questionnaire was also used to assist the construction of the model. A prototype based on the model was built. In the implementation of the prototype, the WWW, java, OOP, and hypertext provides the technology to achieve the aim of this research. To gauge the effectiveness of the system, an evaluation is conducted on the prototype using information gathered through a questionnaire.

4.2.3 The Objectives of the Research

In line with the purpose of the study, the objectives of this research are as follows:

- To identify the requirements of a flexible software inspection process system
- To develop a model of software inspection process that is flexible in the following ways: allows the working mode of the software inspection process the flexibility in terms of space-time matrix; flexible in supporting both formal and informal inspections; and flexible in terms of allowing inspection process work to switch from one quadrant to another during an inspection process
- To integrate the design considerations for CSCW system into the design of the software inspection prototype. Among the design consideration that need to be integrated are as follows: maximise personal acceptance, minimise constraints, unobtrusive accessibility, external integration, local flexibility, provides shared state and awareness, and reduce social disruption
- To develop a prototype that supports the FlexSIG model developed earlier and other major software inspection process models, and provides heterogeneous support
- To evaluate the acceptability of the prototype system and the success of the model

4.2.4 Importance of the Study

Over the last few years, user acceptability of CSCW system has become an issue. Sproull (1990) said, "groupware will never be practical and widely used in organisations if it follows its current trajectory". This is because of the problems of trying to fit technology into workplaces are more serious with CSCW (Winograd & Flores, 1986).

These problems are addressed by this research, whick seeks to design a CSCW system model and a prototype CSCW system that is flexible. Group working processes as outlined by Wilson (1990, 1991) need to be understood in order for the correct model to be constructed and the system accepted. The model and the system must be adaptable, because anything that disrupts social dynamics and does not accommodate wide range of exception handling will bring trouble (Grudin, 1994).

The model developed in this research is based on the review of literature and questionnaires with the aim of being flexible to changes, making it easier to model the way the software inspection works. This in turn, will produce a more acceptable system to the user because it is more easily adaptable to the way the user works. Another aspect that will be enhanced is the structure of the software inspection process itself. Currently, the software inspection meeting must be conducted as a formal meeting or it must be conducted in only one mode of the space and time taxonomy, because there is a lack of tools that will allow otherwise. The flexible software inspection groupware will allow a flexible structure. The proposed system is not restricted to the face-to-face scenario, but can be used in all four quadrants of the space-time matrix. The existence of a tool that is not restrictive allows users to extend the structure of code inspection meetings beyond the existing formal structure.

The system proposed also can accommodate multiple computing platforms. Different computer systems, with each running different operating systems will be able to use the same system with the same user interface. Users across the network can still use their current platform and the issue of forcing the user to abandon their favourite platform does not arise.

The system developed is flexible enough to allow the user to choose the major method of software inspection process that they prefer. The prototype is not tied down to any particular model of the software inspection process.

Thus this research will extend current technology of software inspection groupware, and the flexible software inspection structure resulting from using the system will have a great significance on the non-execution based software testing in the industry. The novel aspect of this research are as follows:

- A flexible software inspection groupware system that is not limited to any one of the four categories of time and space taxonomy
- A flexible software inspection groupware system that is able to accommodate different software inspection process models
- A flexible software inspection groupware system that is not limited to any computer platform.
- A flexible software inspection structure based on the flexible software inspection groupware.
- A model of a flexible software inspection groupware system.

4.3 RESEARCH DESIGN AND PROCEDURE

In this section, the research design and procedure is looked into in a more detailed manner. The research design scheme, data sources for this research, the detailed research procedure, and method of data analysis are then outlined. Any assumptions or limitations of this research are covered at the end of this section.

4.3.1 Research Methodology

Research, in general, can be divided into several categories. Functionally, it can be divided into: descriptive, historical, correlational, developmental, ex-

perimental, and ex-post-facto (UTM, 1994). Descriptive research is concerned with determining the nature and degree of existing conditions. Historical research is concerned with determining, evaluating, and understanding past event for the purpose of better prediction of the future. Correlational research is concerned with discovering the degree of relation between two or more variables. Developmental research is concerned with the patterns of growth as a function of time. Experimental research is a scientific investigation in which one or more independent variables is manipulated and the dependent variable or variables is observed. Ex-post-facto research is to investigate possible cause-and-effect relationship by observing some existing consequence and searching back through the data for plausible causal factors.

In experimental research, two basic conditions exist: first, at least two methods are compared to assess the effect of particular treatments, and second, the independent variable is directly manipulated by the researcher (Hedrick *et al.*, 1993). There are a number of experimental designs that can be used which includes: one-shot case study design, one-group pre-test - post-test design, static-group comparison design, randomised post-test - only control group design, randomised pre-test - post-test control group design, and randomised Solomon four-group design. In the one-shot case study design, the test or the analysis is done after the treatment only. There will be only one group which is exposed to a treatment or event. And a dependent variable is subsequently measured.

The exact nature of this research is explained in *Section 4.3.2*.

4.3.2 Research Design

The research will be conducted in three stages. The first stage will involve developing the model for the flexible software inspection process. The second

stage will develop the prototype based on the model, and the third stage will evaluate the prototype to find it acceptability among the user.

For the modelling stage, the review of literature on existing groupware support for software inspection process is the basis for the construction of the model. A questionnaire is provided to a group of software engineers to help gather more information to provide additional input in the construction of the model. The model produced in turn, will be used to design the prototype.

In the second stage, the prototype will be built. It is needed in order to test the model. The prototype is implemented using the four enabling technology identified earlier. Once the prototype is completed, further tests can be conducted.

The third stage of this research is closely related to the one-shot case study of the experimental type mentioned before. There is only one group of software engineers involved. The group is exposed to one treatment, namely, evaluating the groupware prototype, FlexSIG. Data is gathered after the treatment in the form of a questionnaire to each individual. The results from the third phase are analysed to get the information on the acceptability and flexibility of the prototype, and thus the model.

4.3.3 Data Sources and Analysis

For the modelling of the flexible software inspection process, the data sources come from existing software inspection process models and tools that support them. A questionnaire given to a group of software engineers also forms another data source.

The data source for the evaluation of the prototype comes from the questionnaire collected from the group at the end of the 'treatment'. The data gathered is analysed using simple statistical tools.

To analyse the data gathered, percentage, means, standard error, and standard deviation are employed. Pie chart and histogram are also used. To analyse the relationship between different variables, crosstabulation and boxplots are put to use.

4.3.4 Assumptions

There are a number of assumptions that have to be made. In the first phase, the following assumptions are made:

- Data gathered from the review of literature of existing software inspection process and tools is sufficient as the basis for the FlexSIG.
- A small development team will interact and communicate in a same way a large development team interact and communicate.

For the third phase, several assumptions are made:

- It is assumed that the evaluation by one group of software engineers is sufficient in determining the effectiveness of the prototype.
- It is also assumed that the evaluation in an educational setting reflects the real world situation.

4.3.5 Limitations of the Study

There are certain limitations that have been imposed on this research. Some of the limitations are to make the research more manageable and some caused by

the availability of resources, for example, equipment and people. The limitations are as follows:

- This study is limited to a certain aspect of the software development process only. The review of literature of existing practices form the basis of the flexible software inspection model. Data gathered from a group of software engineers is used to assist the construction of the model.
- In the implementation of the prototype, this research concentrated on a smaller aspect of software inspection, namely, code inspection.
- The scope of CSCW enabling technology usage is also limited to text. No visual or audio aspects of CSCW are used in this research.
- The evaluation is done on a small group of software engineers in an educational environment. Only one-shot case study design is employed in the evaluation. The result might not reflect the true situation in the real-life working situation.

4.4 SUMMARY

As a summary, the aim of the groupware system designed and built in this research is to provide a system that will improve acceptability of groupware and to improve software quality by providing a software inspection tool that is flexible. The groupware system includes a flexible model of the software inspection process itself.

This research is experimental in nature and uses the one-shot case study design. There are three phases in this research. The first phase is to develop a model for the flexible software inspection process, the second phase is to develop the flexible software inspection groupware prototype, and the third

phase is to evaluate the prototype. Simple statistical method is used to analyse the evaluation.

Section 4.1 formulates the problem of this research based on the synthesis of the literature review. *Section 4.2* looks into the question of the aim, the objectives, and the importance of this research. *Section 4.3* details the issue of the design and procedure of this research.

In the next chapter, the discussion on the development of FlexSIG model is presented.

Chapter 5

MODELLING FLEXSIG

5.0 INTRODUCTION

This chapter discloses the model of FlexSIG. The FlexSIG model aims to be flexible in terms of working mode, type of platform, and variation in implementation of the software inspection process.

- **Section 5.1** discusses the proposed model in relation to the literature reviewed and the aim of this research.
- **Section 5.2** gives the conceptual framework of the FlexSIG. The conceptual guidelines for the FlexSIG model are presented in this section.
- **Section 5.3** deals with the FlexSIG functional model which includes the data and process model.

5.1 PROPOSED MODEL DESIGN

The model proposed is based on the literature review and data collected from a questionnaire. The model aims to meet the goals outlined in the overall purpose of this research given in the previous chapter.

5.1.1 The Aim of the Proposed Model

The purpose of this study, as described in Chapter Four, is to develop a Flexible Software Inspection Groupware (FlexSIG) system which includes:

- the development of a flexible software inspection process model
- an implementation of a prototype based on the model of the system
- the evaluation of the prototype in order to measure the acceptability of the system.

The model seeks to support designing a system for software inspection process by providing the overall structure using CSCW, taking into account the requirements of a good CSCW system, in particular addressing the issue of acceptability of the process in this context.

The model addresses the issues of critical mass, disruption of social processes, unobtrusive accessibility, integration of different tools, and lack of flexibility in the software inspection process. The design strategy proposed in the previous chapter forms the basis for the solution offered here.

5.1.2 Existing Models

Looking at the current software inspection models, none of the major variants provide any flexibility in the execution of the inspection process. The models proposed by Fagan, Gilb & Graham, and Humphrey are based on a formal manual inspection process. None of the models support the design issues mentioned in Chapter Four. In the other works related to software inspection tools, the research is concentrated on solving the problems directly related to the software inspection process itself. As mentioned by Macdonald (1996), the research on software inspection tools is concentrated in supporting document

handling, individual preparation, meeting support, and data collection. Although there is an attempt in these works to address the issue of groupware aspects of the software inspection process, their proposals were focused on providing support in only one quadrant of the space-time matrix.

5.1.3 The Proposed Model

The proposed model in this research is based on the combination and an extension of the Fagan (1976, 1986), Gilb & Graham (1993), and Humphrey (1989) inspection models found in the literature. For most parts of the model, the processes, the roles, and the techniques they described are retained. This is because these models have proven effective within the scope defined for their usage, so they provide an established foundation for the FlexSIG model.

The FlexSIG model makes changes to these established models in order to address the CSCW design issues raised in Chapter Four, which were also formulated from the literature. For the major part of the model development, the design strategy proposed in Chapter Four is used as the basis for the solution. The questionnaire also provided information to the model by raising further design issues.

Specifically, the model developed aimed to provide flexibility in the choice of working mode. The model provides the facility for the software inspection process to use all four quadrants of the Johansen (1988) space-time matrix or the four categories of the time and space taxonomy proposed by Ellis *et al.* (1991). In order to support working mode flexibility, the model needs to provide alternatives in the process flow to the user.

To maximise personal acceptance (Cockburn & Jones, 1995) the model is based on the technology that is already widely accepted, that is the world wide

web. Local flexibility (Robinson, 1993) is handled by providing multiple components to support the same function. The minimised constraint strategy (Cockburn & Jones, 1995) can be implemented by providing the flexibility mentioned above. The external integration (Cockburn & Jones, 1995) strategy is implemented by utilising the web and the java technology. Awareness (Dourish & Bellotti, 1992) is implemented using the communication suite and also utilising shared states (Shim *et al.*, 1997).

The questionnaire, which is used to obtain user views on design requirements, provides additional points. Based on the input from the questionnaire, the model aimed to offer ease in obtaining requirement documents; reduce the difficulty in conducting a meeting; provide alternatives in communication components; reduce the difficulty in handling large numbers of documents; provide help in categorising the defects; support a consistent graphical user interface on different computing platforms; and provide report generation. The next section summarises the input from the questionnaire.

Summary of Questionnaire

To gain user input on the design of the proposed system, a questionnaire was used. Details are provided in Appendices H and I. From the questionnaire, among the additional input that was gathered were the following:

- Users have difficulties in using the inspection and review process. The reasons stated were difficulties in categorising the errors and defects, no training given on how to conduct the process, not being familiar with the inspection process itself, the inspection process is time consuming, the process is a tedious job, there are a number of documents to handle in the process, and also difficulty in accessing the design document during the inspection process.

- It is difficult to get hold of the design and requirement document during the inspection or review process.
- The difficulty level of conducting a meeting, although on the easier side, is almost in the middle of the scale.
- The most common method of communication during software development project is a face-to-face meeting, followed by the telephone, e-mail, memos, and electronic chat. The respondents also mentioned other methods of communication such as informal conversation and informal meeting.
- Paper is still the major medium of project document distribution among team members. This is not surprising considering that the nature of the inspection process used by the respondents is still manual and formal in nature.
- The kind of improvement that the respondents would like in the inspection process is automation of the process in the form of a CASE tool. Among the features that they would like to see is direct access to specification documents such as process flow and relationships, and support for specification, design, and code inspection, a graphical user interface and good report generation features, and support for testing, such as an automatic test generator.
- A distributed CASE tool for the inspection process is a better choice than a stand-alone tool.
- The indication is very clear that the respondents are positive towards using a software inspection process in the future.
- The figures from the analysis also reveal that those who experienced less difficulty in obtaining reference documents during coding are more likely to

use an inspection process in the future than those who experienced more difficulty.

- The crosstabulation analysis showed that the more difficulty people have in conducting meetings, the less likely that they will use an inspection process in the future.

5.2 CONCEPTUAL FRAMEWORK OF FLEXSIG

The outline for the framework of FlexSIG is presented in this section. Assumptions, considerations, and fundamental concepts are outlined before the conceptual framework of FlexSIG is described. The conceptual framework is based on a similar approach by Hennessy (1990) and Lunt (1990). The findings from the questionnaire are not used in the development of the conceptual framework but are used for functional modelling.

5.2.1 Assumptions

A couple of assumptions had to be made with regard to the flexible software inspection process model. The first concerns the organisational structure of the software inspection process itself, and the second the role description of the members of the inspection team.

Structure of the Inspection Process

On the organisational structure of the software inspection process, this research assumes that the model created will be sufficiently generalised for it to be used by all aspects of the software inspection process. It is assumed that the

structure of the software inspection process can be applied to the requirement inspection process, the design inspection process, and the code inspection process. The generic organisational structure of the software inspection process is created once only. It is also assumed that the structure is generalised enough to support different modes of working, namely, face-to-face, distributed synchronous, and distributed asynchronous.

Role Description

It is assumed that the role of the members of the inspection team can be defined consistently across different aspects of the software inspection process. It is assumed that, for example, the role of moderator is the same in the design inspection process as in the code inspection process. Thus, we can define the role of the moderator just once, and it can be applied to different types of software inspection process. The same assumption holds true for other roles such as the reader, the author, and the inspector. The assumptions on the role description also apply to different working modes. It is assumed that the basic description of each role is the same in face-to-face, distributed synchronous, or distributed asynchronous scenarios.

5.2.2 Considerations

Among the elements that are considered when constructing the flexible software inspection model are:

- The inspection activities
- The role adjustment of the team members
- The information access by team members

- And the execution platform of the toolset to support the flexible software inspection model.

Inspection Process Mode

The inspection process mode needs to be flexible. This need has been pointed out in Chapter Two, Three, and Four. The process mode needs to be flexible in terms of the mode of the meeting being supported. The model must be able to address the formal set-up of the meeting as well as the informal set-up. The model must also be able to handle the fact that the team members may not necessarily be at the same place and at the same time. In other words, the model must be able to handle the distributed synchronous mode of the meeting and also the distributed asynchronous mode of the meeting, as well as the face-to-face meeting.

Roles Adjustment

The roles of the team members described by the model should also accommodate the fact that the role of a team member needs to be adjusted between different modes of the inspection process. The role of the moderator may change slightly between a synchronous meeting and an asynchronous meeting. The same argument holds true for other roles as well.

Information Access

A flexible software inspection model must be secure. A model that includes information access from a distance must have some form of access control. Access control of information must also be determined between different roles.

Team members with different roles should have different levels of access to the information stored and to the functions they can perform.

Platform

The flexible software inspection process model should also address the issue of heterogeneity of computer platforms: different users use different computer platforms running different operating systems. The model and the toolset should accommodate the different types of computer platform that exist. The consistency of the graphical user interface across different platform should also be considered.

5.2.3 High-level Concepts in the Software Inspection Process

In any software inspection process, there are four high-level concepts that can be categorised as the foundation of the process. These high-level concepts are as follows: the inspection phases or stages, the role of the team members, the documents, and the data. The four high-level concepts are based on viewing the software inspection process from the basic input-process-output flow view in software engineering. The input in this view is the documents, the output is the data (defects, queries, and suggestions), and the process is the inspection phases with inspection team members being assigned a specific roles. The high-level concepts is based on a similar approach by Hennessy (1990).

Inspection Phases

The inspection phases, or stages, are steps in the process of software inspection. Fagan (1976) originally defined five steps for the inspection process. Gilb & Graham (1993) defined the software inspection process as having ten phases. On the other hand, Humphrey (1989) argued that the inspection process consists of three major phases: preparation, inspection, and post-inspection activity. The steps or phases mark different activities or developments along the process.

The time duration for each phase is determined by the leader of the inspection team. Some phases require the participation of all team members, for example the inspection meeting, but some phases require participation from only the leader, for example the consolidation. All the phases in the model introduced in *Section 5.1* occur sequentially one after another and are geared toward face-to-face meeting. A model that supports different working modes of the inspection process must offer flexibility and choices in the execution of the stages.

Team Member Roles

Each team member is given a specialised responsibility during the software inspection process. Each role has a different scope and a different objective. The inspection team roles described by Fagan (1986) are as follows: moderators, authors, readers, and tester. Gilb & Graham's (1993) inspection team consists of an inspection leader, author, and checker. A scribe is appointed to record the meeting. The inspection participants, as recommended by Humphrey (1989) are as follows: the moderator, the producers, the reviewers, and the recorders. The number of roles in a software inspection process will determine the number of participants.

In the FlexSIG model, there are only three roles, namely: the moderator, the inspector, and the author. The number of roles in FlexSIG is based on existing models, minus any roles that can be handled automatically by the system. The moderator leads the inspection team, similar to Fagan's and Humphrey's moderator and Gilb & Graham's inspection leader. The role of the author is similar to Fagan's and Gilb & Graham's author and Humphrey's producers. The role of inspector is comparable to Fagan's tester, Gilb & Graham's checker, and Humphrey's reviewers. In this model, the role of scribe is handled automatically by the system, whereas in existing model, a scribe or a recorder is appointed during the meeting to note down the discussion and the defects found. Similar to Humphrey's and Gilb & Graham's model, FlexSIG does not have any special role for reader. The task is assumed by the moderator.

The role of the moderator spans from the planning stage up until the follow-up stage. The moderator is responsible for planning the whole inspection process and leads the team. The role of the author occurs in the inspection stage only. The author's role is to answer any query with regard to the document inspected. The role of the inspector spans the individual checking and the inspection stage. The objective for an inspector is to maximise the number of defects found.

Each role has its own access right to the documents. The access right is assigned based on the need of the roles mentioned earlier in this section. The access right allows each roles to accomplished their task and at the same time safeguard the information. The moderator needs to have access right to the document to be inspected and other supporting documents in order to configure and set-up the system. The moderator's job is to manage the inspection process and solve any problems (Fagan, 1976, 1986). The moderator has no right to enter any log or take part in the voting on any issue in the log file. This is because the moderator's role is to plan and lead the team, and not to find defects.

The author of a document also cannot log any comment or take part in the voting because the author's role is only to answer any query or misunderstanding during the inspection process (Fagan, 1976, 1986). Only the inspectors, whose role is to maximise the number of defects found (Gilb & Graham, 1993), can enter any defects or query into the log file and participate in the voting on any outstanding issue during the inspection meeting.

Documents

In the software inspection process, the written set of information that is the subject of the inspection is called a document. The document can be design material, or it can be program code. There are a number of associated materials related to the documents being inspected. In the case of written code being inspected, the related material is the requirement and the design material. Similar to Gilb & Graham's (1993) model, other supporting documents include briefing material, description of the project as a whole, description of the specific document inspected, rules, check-list, and procedures. In a manual software inspection process, the document is in paper form and is distributed during the kick-off meeting.

Data

The objective of the software inspection process is to find defects (Fagan, 1976, 1986; Gilb & Graham, 1993). An item suspected to be a defect is logged by the inspector during the individual inspection phase. The log item is accepted or rejected as a defect during the inspection meeting phase. Although the objective is to find defect, other secondary data is also collected from the process. Data collected includes queries on certain aspects of the document, and suggestion or improvements on the defects found (Gilb & Graham, 1993). Neverthe-

less, Gilb & Graham (1993) warned that suggestions or improvements are a by product of the inspection process and discussion on the solution to the defects is wasteful. Metrics collected at the end of the software inspection process and the report are sent back to the author and in some case to the software quality team.

The defects collected are classified into two categories. In one category, the data is classified according to its criticality. The data is categorised into major defect, minor defect, and warning (see *Table 5 - 1*) (Fagan, 1976, 1986). Gilb & Graham (1993) use a similar category for defects and stated that the category served as a guide for the author in correcting the defects found.

Type	Criticality of Defect
1	Major Defect
2	Minor Defect
3	Warning

Table 5 - 1 : *Type of Defect Criticality*

The data is also categorised according to its defect type. The defect types are: data defect, design defect, documentation defect, language defect, logic defect, performance defect, test & branch defect, maintainability defect, and other defects (see *Table 5 - 2*). The classification is based on Fagan (1976, 1986) method of categorising defects. Humphrey (1989) proposed a similar classification. The classification on the defect type serves as a self improvement tool for the author.

Type Num.	Defects Type	Type Num.	Defects Type
1	Data	6	Performance
2	Design	7	Test & Branch
3	Documentation	8	Maintainability
4	Language	9	Other
5	Logic		

Table 5 - 2 : *Type of Defects*

5.3 FUNCTIONAL MODEL OF FLEXSIG

The conceptual framework serves as the basis for the functional model. In this section, the outline of the functional model is presented. This is followed by the components and the services offered by the functional model, the data model, the process model, the user interface, the access control and the platform issues of the functional model. The functional model is based on a comparable approach by Zeb (1993) and Lunt (1990), and also from the analysis of the questionnaire.

5.3.1 Outline

The functional model of FlexSIG is described by examining its key components and services; its data model which represents the relationship between the artefacts in the model; its process model which outlines the process flow; its user interface; its information accessibility; and its platform dependence. This approach is based on a similar functional model used by Zeb (1993), the functional

architecture used by Lunt (1990) and the functional architecture used by Trevor *et al.* (1993).

5.3.2 Components and Services

There are six key components and services of FlexSIG. The components are as follows: access control, briefing, document browsing, communication, data logging, and supporting document facilities.

Access Control

Access control is an important component but not central to this research. The importance of access control in collaborative work was discussed by Shen & Dewan (1990). The access rights presented by Shen & Dewan (1992) provides the basis for FlexSIG access control.

Access control manages the security aspect of the model. Access into the system is managed by requiring team member to enter valid username and password. The information entered is compared with the information stored in the access control database. This in turn will determine the access level of the user. Menus presented to the user depend on their role. This means that certain functions or services are not available to certain users. The user's role also determines what file they can access and modify, and whether the user can vote on an issue or inspect a document.

Briefing

The briefing component describes the set-up information of the software inspection process. The contents of the briefing material are similar to the informa-

tion disseminated during the kick-off meeting in Gilb & Graham's (1993) model. The approach and the contents of the briefing component also aim to alleviate the issue raised by the questionnaire, including reducing the difficulty in conducting a meeting and reducing the difficulty in handling a large number of documents. The briefing material is prepared by the moderator and is presented to other team members. The function of the briefing component is to:

- Inform the composition and roles of the software inspection team. Details of team member such as name, role, e-mail and telephone are included.
- Inform the organisation of the software inspection process: how the process is going to be conducted, what mode is to be used, the length of each phase, deadlines, timetables, method of communications, etc.
- Inform the detail of the software inspection process: the name of the document to be inspected, the description of the document in terms of what it is for and what it's trying to achieve, the objective and the target, special instructions, inspection checklist, etc.
- Brief team members on the overall aspect of the software project the document belongs to. List the name and location of other document that are related to the document being inspected.

The information provided in the briefing component is a necessity before the actual inspection phase can begin.

Browsing

The document browsing component contains two sub-components, which are the facility to browse inspected document and to browse remotely document inspected by other team members. The browsing capability is the basic document handling expected in any inspection tools (Macdonald, 1996). The remote

browsing facility is to support synchronous communication during the group inspection phase.

The first sub-component allows team members to choose and browse the inspected document. The browser component is comparable to the one provided by other tools (Brothers *et al.*, 1990; Johnson & Tjahjono, 1993) and provides part of the solution to the issue of difficulty in handling large number of documents raised by the questionnaire. Users can select a document to be inspected from a list of files. Line numbers are added automatically to assist the user. In the second sub-component, users can browse the same document that other team members are currently looking at. The remote browsing sub-component is similar to Brothers *et al.* (1990). The sub-component also aims to provide part of the solution to the issue of reducing the difficulty in conducting a meeting as raised in the questionnaire. Users are given a list of other users by name. The list by name is chosen because the other users might view different documents at different times. Thus, by choosing any name given, the corresponding document which is currently being browsed by other users is displayed. Only the username of the user who is logged into the system is displayed in the list. The service is to assist synchronous discussion between two or more team members.

Communication

The communication component contains three sub-components, namely, electronic mail, chat, and pop-up message. The communication component seek to address the issue raised in the questionnaire, which is to provide alternatives in the communication component and reduce the difficulty in conducting a meeting. The aim of the communication component is to support both the synchronous and the asynchronous mode of communication. This is to provide the

model with the flexibility mentioned in the design goals of the model. The choice of different communication facilities also implement local flexibility as proposed by Robinson (1993). The e-mail sub-component is system specific and is limited to the team members only. Users can send e-mail directly from the sub-component to other team members. Users can also retrieve mail sent to them and reply back to team members. Ideally, the e-mail component should be integrated with the outside world. But the security issue in an organisation might prevent it from doing so.

The second sub-component is the chat system. The chat sub-component is divided further into two sub sub-components. The first is chatting in a pre-arranged room. Team members can chat in the common room of the team which is set-up by the moderator. This is the common room for all team members. They can also set-up their own room and chat with two or more team members. The second sub sub-component is one-to-one chatting with other member. In this mode of chatting, other team member activity is not disrupted and users can have privacy. The room concept was also used by Lee *et al.* (1996) in their system. Any general discussion regarding the inspection process can be conducted in the common room. The private room allows a smaller group discussion, for example among the inspectors, to take place.

The third sub-component of the communication component is the pop-up message. This sub-component provides the informal message structure raised in the questionnaire. In this sub-component, team members can pop-up a message to other users. This sub-component is also divided into two sub sub-components. The first is the facility to pop-up a message to the entire team. Every team members who is logged on will received the same message. The second sub sub-component has the capability to pop-up a message to one individual user only. The general pop-up message facility allows any group announcement or message to be send. The private pop-up message allows a more

discreet communication, for example, the moderator may wish to discuss some issue privately with the author.

Data Logging

The data logging component is another important component of the model. There are five types of data in the model: defect, query, suggestion, voting, and the overall metrics. The collection of defects is the reason inspection is conducted (Fagan, 1976, 1986). Query and suggestion facilities are included based on Gilb & Graham's (1993) method of data collection where queries and suggestions are collected alongside the defects. Voting (Johnson *et al.*, 1993) is needed to resolve any issues in order to accept or reject any potential defect during the discussion. No voting is required for queries or suggestions because these items are accepted as is. The collection of metrics is a standard feature of software inspection tools and is implemented by all the tools reviewed, except InspeQ, which provided a minimal support.

Defects, queries, suggestions, and votes can be logged only by the inspectors, and are consolidated by the moderator. Suggestions are related to a defect or a query logged by the users. They cannot stand on their own. Voting will determine whether a logged defect is accepted or rejected. Metrics collection is based on the defects and is consolidated by the moderator during the consolidation phase. There are two types of metrics collected: metrics for the document's author and metrics for the software quality team.

Supporting Document

The supporting document component contains information with regard to the document being inspected and also information about using the FlexSIG sys-

tem. Information about the document being inspected is in the form of requirement specification, design specification, syntax of the programming language, and syntax of the operating system. The type of information available depends on the type of inspection conducted. The inclusion of the supporting document facility is motivated by the questionnaire, where the respondents mentioned the difficulty in obtaining the document during the inspection process. The model aims to reduce this problem by providing the document on-line. This component also aims to alleviate the difficulty of handling large numbers of documents, an issue raised in the questionnaire.

Help on using the FlexSIG system includes general instruction on how to use the system and explanation on the modules of the system. The syntax of the programming language used in the coding and the operating system is included in the help component of the system.

The specification document is a component by itself.

5.3.3 Data Model

The data model of FlexSIG represent the relationship between artefacts in the model (see *Figure 5 - 1*). The artefacts here means the document, its supporting and output material, and the actions. The nodes in the data model represent the artefacts and the directed links represent the relationship between the artefacts. The data model show the dependencies of one artefact to the other. The representation of FlexSIG data model is based on a similar model by Johnson & Tjahjono (1993) when they described the CSRS inspection tools. Although there exist other approach to data modelling as proposed by Wang (1993), the approach by Johnson & Tjahjono (1993) was taken as the basis because it is the only model that is related to the software inspection process.

The document inspected is the centre of activity in this model. This is depicted by the node 'Document'. From the document inspected, users can log any defects or queries into the log file, which is denoted by the node 'Log'. During the initial stage of inspection, the briefing materials serve as the background information, explaining the process and any related materials. This is represented by the node 'Briefing'. During the inspection process, further clarification is available from the specification material, represented by 'Specification', and help material, represented by 'Help'. Further clarification can be solicited from other team members using the communication component, which is depicted by the node 'Communication'.

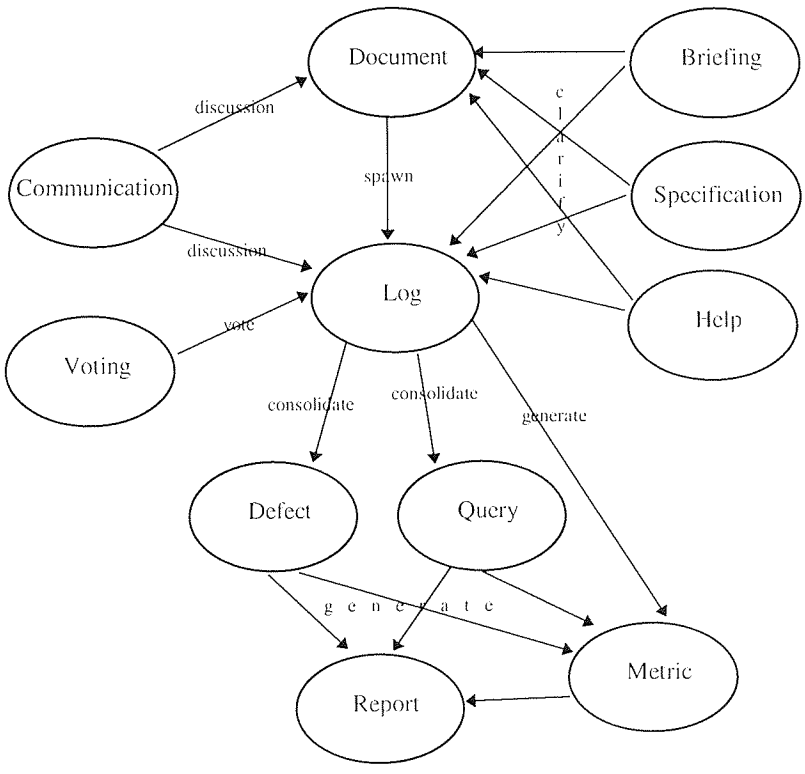

Figure 5 - 1 : *Data Model of FlexSIG*

Similarly, clarifications on the items in the log are available from 'Briefing', 'Specification', and 'Help'. Discussion and further clarification is requested via 'Communication'. The status of the items logged is settled by the voting mechanism. The final list of defects is consolidated by the moderator

based on the voting. This is represented by 'Defect'. In a like manner, the query list and the inspections metric are consolidated by the moderator. These are denoted by 'Defect' and 'Query', correspondingly. The report for the inspection process, symbolised by 'Report', is generated from 'Defect', 'Query', and 'Metric'.

5.3.4 Process Model

A process model is often used in describing an inspection model (Fagan, 1976, 1986; Gilb & Graham, 1993, Johnson & Tjahjono, 1993). The majority of existing models provide the process model to describes the structure of their approach. The process model of the software inspection process is a generic model: the model can be applied to the requirement inspection, the design inspection, and the code inspection process (see *Figure 5 - 2*). The process model represents the flow of the process along different stages of software inspection. The software inspection process starts when the document author requests the document to be inspected and the document is deemed to have satisfied the entry criteria requirement set by the quality team.

Initiation

The inspection process is initiated by the moderator calling all team members to a kick-off meeting or directing them to access on-line briefing material. The goal of this phase is to start the inspection process. E-mail is send to every team member to start the process moving, although the inspection process can be initiated using paper based solution. The username and password for team members are distributed along with the call to start the inspection process.

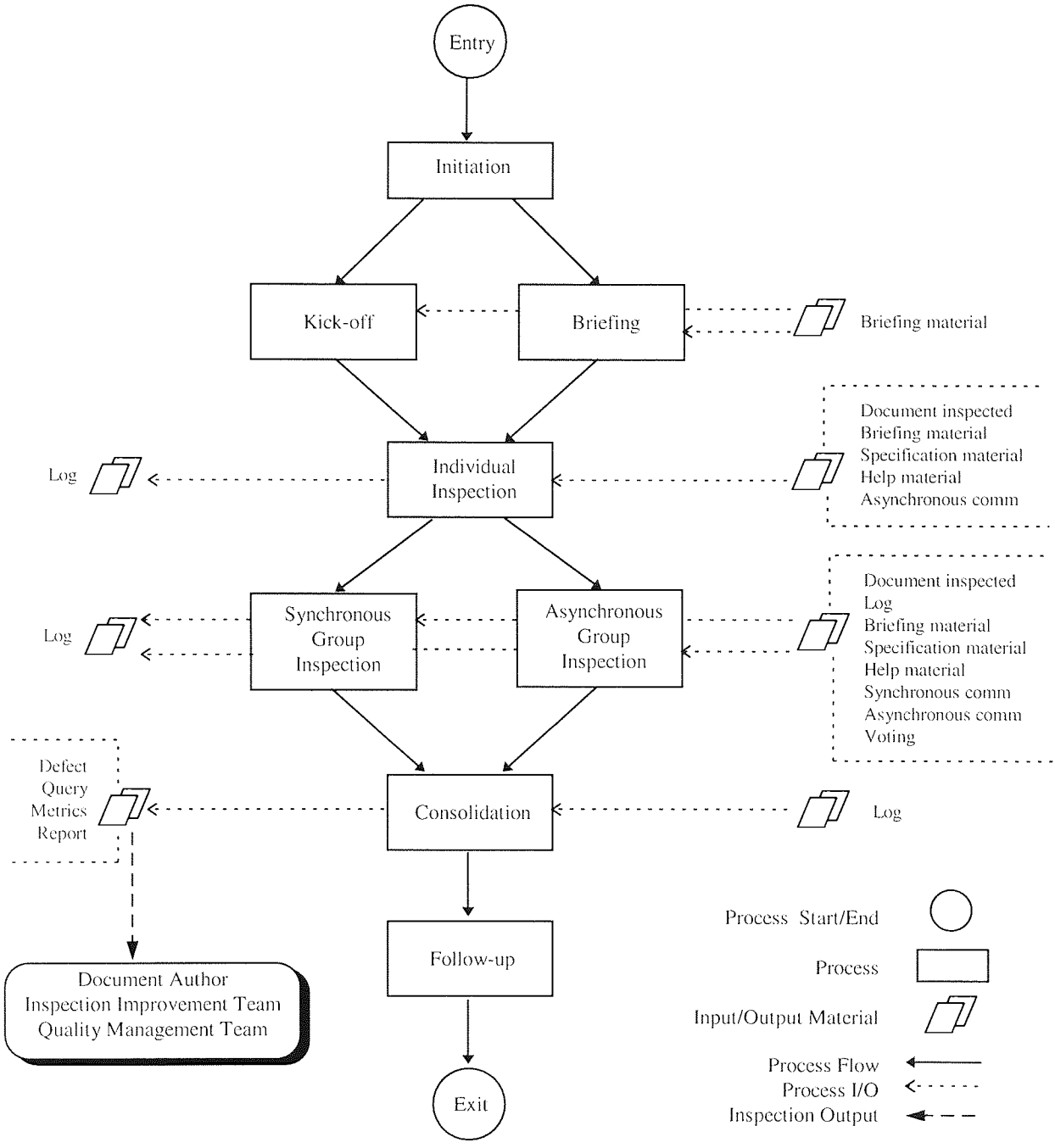

Figure 5 - 2 : *Process Model of FlexSIG*

The choice between conducting the kick-off meeting or just accessing the briefing material is decided by the moderator and the practice in the organisation. The planning and the material for the kick-off meeting or the briefing are prepared beforehand by the moderator.

Kick-off Meeting

The aim of the kick-off meeting is to inform the team members on aspects of the document being inspected. All necessary information needed for the inspection to take place is furnished. The kick-off meeting is conducted face-to-face. This phase is similar to Fagan's (1976, 1986) overview phase, Gilb & Graham's (1993) kick-off meeting, and Humphrey's (1989) opening meeting. The briefing phase is conducted, instead of the kick-off meeting, if the moderator decides this. The team members are introduced to other members and briefed on their roles and objectives. The organisation of the inspection process is furnished, for example: deadlines, timetable, and methods for the whole inspection process. Description of the inspected document, any special instructions and check lists for inspectors are given here. Explanation about the overall background of the project in which the document belongs is also provided. The material can be distributed in paper form or as on-line material.

Briefing

If the moderator decided that the information with regard to the document being inspected is to be distributed electronically, then this phase is conducted instead of the kick-off meeting. The briefing phase is the asynchronous equivalent of the kick-off meeting. This phase is created as a method to inform the team members on aspect of the document being inspected asynchronously. The material is exactly the same as explained in the previous section. All the material is accessible electronically. The location of the material and the information needed in order to log into the system is distributed in the e-mail that initiates the inspection process.

Individual Inspection

In the individual inspection phase, each inspector is instructed to study the document being reviewed and log any potential defect or query. Regardless of the software inspection process working mode that the moderator decides, the individual inspection phase stays the same. Each inspector is to work on their own using the toolset and log the potential defects, queries and suggestions directly into the system. If there is any need for clarification, the inspector can go back to the briefing material, or use the help or specification component of the system. Alternatively, the inspector can communicate directly with the moderator or the author using the communication component. This phase is the equivalent of Fagan's (1976, 1986) preparation phase.

Synchronous Group Inspection

Group inspection is to be conducted differently depending on the mode of the software inspection process. If the inspection process is conducted in a face-to-face mode, then the group inspection is held in a room with the toolset assisting team members, similar to ICICLE (Brothers *et al.*, 1990). If the inspection process is conducted in a distributed synchronous mode, the team members access the system from different places at the same time, similar to CSI (Mashayekhi *et al.*, 1993). The discussion here is through the synchronous communication component of the system, namely, chat, pop-up message, and remote viewing of other members document.

Team members can log further potential defects, queries and suggestions into the log file. Any duplication of potential defects and queries is identified and resolved in this phase. A decision on accepting the defect found in the individual inspection phase and this phase is done through a voting mechanism. In all modes of operations, clarification with regards to the inspected

document can be solicited using the on-line briefing, help, and specification material.

Asynchronous Group Inspection

If the software inspection process is in the distributed asynchronous mode, team members access the system from different places at different times, similar to CSRS (Johnson & Tjahjono, 1993). In this mode, a team member is given a time frame to discuss the issue. Discussion is via the asynchronous communication component.

In an asynchronous group inspection, the task is exactly the same as in synchronous mode. Team members can still react to log entries made by other inspectors and they can log new entries. The only difference is in the form of the discussion and the time frame given to discuss the issue, caused by the time delay in the asynchronous communication. Towards the end of the stipulated time, all the inspectors are expected to give their vote on any contentious issue on the defects logged.

Consolidation

During the consolidation phase, the moderator will merge all the defects, queries and suggestions raised by the inspectors during the inspection phase. Any duplicates or potential defects that were voted out in the previous phase are not included. The overall metrics of the inspection process are also collected here. The metrics collected are on the number of defects found and queries put forward by each inspector. The metrics also include the number of defects and queries for the whole inspection process. A report based on the consolidation of

the data collected and the metrics collected is also produced. This phase is comparable to the Johnson & Tjahjono (1993) consolidation phase.

Follow-up

The last phase is the follow-up. In this phase, the moderator is to forward the report on the defects, queries, and the inspection metrics to the document author. It is the responsibility of the moderator to make sure that all the recommendations are acted upon by the author. The metrics collected are also forwarded to the team in charge of the software quality management and the inspection improvement team in the organisation. The follow-up phase was also required in Fagan's (1976, 1986) and Gilb & Graham's (1993) model in order to make sure the defects are corrected.

The software inspection process terminates when the inspected document has met all the exit criteria set by the team responsible for the software quality in the organisation.

5.3.5 Information Accessibility

Information accessibility by the user is controlled by the security access level. This is determined by the role of the team member. There are two factors that need to be considered in the information accessibility issue. The first is that the information is required to be protected against unauthorised access. Secondly, the members of the inspection team need to be given enough access rights in order for them to fulfil their role specification. Therefore, the access rights are based on the roles.

In the software inspection team, the moderator has the highest level of access. The moderator can access every file related to the inspection. The moderator can modify team configuration files, set-up the briefing material, prepare the help material, and set-up the communication arrangements. The moderator is also responsible for consolidating the defects and queries from the log file.

The inspectors have the second highest level of access. They are allowed to log defects and queries into the log file. They can also modify their own password entry. An inspector is allowed to view the inspected document, briefing, specification and help material, and also log entries entered by other inspectors. They can also cast their vote on accepting or rejecting potential defects.

The author has the lowest access level. The author can only view document inspected, briefing, specification and help material, defects, and queries logged. The only file the author can modify is the password file in order to change the password entry.

5.3.6 Platform

FlexSIG needs to be available on different platforms in order to be flexible (Cockburn & Jones, 1995). The system should be accessible on as many platforms as possible in order not to force the user to use any particular system. The availability of FlexSIG on many different platforms provides flexibility in terms of not forcing the user to change from their current system, no extra training needed for new system, and no extra cost for new platform. FlexSIG aims to be available on any machine that can run a web browser with support for java applets.

5.3.7 User Interface

In making FlexSIG available on many platforms, the user interface needs to be consistent. The consistency of the user interface across all platforms needs to be maintained for the sake of training and the user manual for the system. The learning curve is also less steep because of the familiarity of user with the interface if they decide to change from one computer platform to the other.

5.4 SUMMARY

In this chapter, the conceptual framework of FlexSIG, which is comprised of the assumptions, considerations, and fundamental concepts was presented. The last section dealt with the FlexSIG functional model itself. The functional model was described in detail which includes the description on the components and services available, the data model, and the process model.

The functional model will, in turn, be the based of the functional architecture of FlexSIG which is covered in the next chapter along with other FlexSIG prototype issue.

Chapter 6

FLEXSIG PROTOTYPE

6.0 INTRODUCTION

The FlexSIG prototype is based on the functional model of FlexSIG. This chapter discusses the development of the prototype.

- **Section 6.1** look backs at the existing tools for software inspection and the proposed design.
- **Section 6.2** talks about the enabling technology that is used in the implementation. Technology discussed here is the world wide web, java technology, the object oriented paradigm, and hypertext.
- **Section 6.3** is a review on functional architecture of web-based groupware in general.
- **Section 6.4** deals with the architecture of FlexSIG system itself. The discussion includes system, database, user interface, and security access architecture.
- **Section 6.5** describes in detail the components of the FlexSIG prototype. This includes examples of screen shots from the prototype.

6.1 PROPOSED PROTOTYPE DESIGN

The prototype proposed is based on the model developed earlier. Similar to the model, the prototype aimed to solve the goals outlined in the overall purpose of this research in the previous chapter. The prototype implements the strategy outlined in the model using the enabling technologies.

6.1.1 The Aim of the Proposed Prototype

The aim of the prototype is twofold. The proposed prototype is to implement the model described in Chapter Five. The proposed model also further addresses CSCW design issues not implemented by the model. CSCW design issues of critical mass, disruption of social processes, unobtrusive accessibility, lacking integration, and lack of flexibility in the software inspection process were mentioned in the formulation of this research problem. These issues cannot be solved by relying on the functional model only. A number of strategies have to be implemented in the prototype in order to achieve the overall goals of this research.

The prototype aims to address the flexibility issue by supporting the implementation in a heterogeneous environment and also implementing not only the FlexSIG model of inspection, but by supporting the Fagan, Gilb & Graham, and Humphrey models of inspection.

6.1.2 Existing Tools

In Chapter Five, Macdonald (1996) stated that research in software inspection tools is concentrated on supporting document handling, individual preparation,

meeting support, and data collection. The groupware aspect of the software inspection process is addressed only by providing support in one mode of the space-time matrix (see *Table 6 - 1*). None of the tools supported more than one type of inspection model and all are implemented on a specific machine, except AISA.

Inspection Tools	Working mode supported	Inspection Model
ICICLE (Brothers *et al.*, 1990)	Intended to be used in one room	Fagan
InspeQ (Knight & Myers, 1993, 1991)	Single-inspector or individual phases of multiple-inspector inspection	-
CSRS (Johnson & Tjahjono, 1993; Johnson *et al.*, 1993)	Distributed asynchronous	Fagan
CSI (Mashayekhi *et al.*, 1993, 1994)	Distributed synchronous	Humphrey
CAIS (Mashayekhi *et al.*, 1994)	Distributed asynchronous	Humphrey
AISA (Stein *et al.*, 1997)	Distributed asynchronous	Humphrey
Scrutiny (Gintel *et al.*, 1993)	Distributed synchronous	Fagan

Table 6 - 1 : *Software Inspection Tools*

This section summarised existing tools available for software inspection. The first two in the list did not support distributed working. The rest of tools support either asynchronous distributed working or synchronous distributed separately. *Table 6 - 1* lists the existing software inspection tools name, and

shows the working mode that they support, and the inspection model they are based on.

6.1.3 The Proposed Prototype

The proposed prototype is based on the FlexSIG model developed in Chapter Five. Implementation decisions are guided by the needs of the model, making use of strategies that enhance CSCW systems.

The 'maximise personal acceptance' strategy (Cockburn & Jones, 1995) is implemented by using the world wide web as the platform for the prototype. Having reached the critical mass, the acceptance of a system that has a familiar user interface is expected to be better.

The FlexSIG model developed earlier provides the basis for use of the 'minimise constraints' strategy (Cockburn & Jones, 1995). The constraints are expected to be further reduced by implementing the prototype to support multiple models. The utilisation of java and object oriented implementation provides the flexibility in adding or removing components from the system. The availability of multiple components for the same task also allows further flexibility.

The world wide web based prototype also allows the integration of multiple applications in order to implement the 'external integration' strategy outlined by Cockburn & Jones (1995).

Grudin (1994) stated that features that support individual activity, and infrequently used groupware features must not obstruct more frequently used features, yet must be known and accessible to the users. Again the combination of java and the world wide web allows this to be implemented.

Awareness (Dourish & Bellotti, 1992) is achieved by using an informational approach which provides explicit facilities through which collaborators inform each other of their activities. The communication suite and the shared state provide the support for this.

6.2 ENABLING TECHNOLOGY

Four type of technology that are of interest to this research are the world wide web (WWW), java technology (Java), the object oriented paradigm (OOP), and hypertext. All four areas provide the technology to enable this research to accomplish its goal. Therefore, it is useful to describe the technologies before actually using them in the development of the prototype.

6.2.1 The World Wide Web

This section introduces the world wide web in a more detailed manner. Discussion on the support provided by the WWW for collaborative work in general is presented here.

WWW Revisited

Chapter One discussed briefly the world wide web. To continue from that, a Web browser or browser program is application software which, when run on a computer connected to the WWW, allows its user to move at will around the Internet (Berners-Lee *et al.*, 1994; Tatters, 1996). The browser is used to look at various Web pages and access Web resources installed on Web servers across the Internet (Horton *et al.*, 1996). However, it can only display Web documents

or Web content with a specified home page address or uniform resource locator (URL). The Web documents are written using hypertext markup language (HTML). HTML is a computer language that enables links to be embedded to other documents within a Web page, thus creating a third dimension (Berners-Lee *et al.*, 1994; Kennedy, 1995). The links on Web pages that point to another Web page in the same or different Web document or Web content are called hotlinks or hyperlinks. HTML is based on an older SGML (Standard Generalised Markup Language) which is a superset of HTML (Berners-Lee *et al.*, 1994). The Web content is transferred using the HTTP (Hypertext Transfer Protocol). HTTP is a protocol for transferring information with the efficiency necessary for making hypertext jumps (Berners-Lee *et al.*, 1994). The data transferred may be plain text, hypertext, images, or data in any other format.

The WWW and CSCW

This sections discusses applications where the WWW supports CSCW. Among these applications are online access to a variety of information resources, project repositories (Rein *et al.*, 1997), distributed applications (Ciancarini *et al.*, 1996), groupware (Trevor *et al.*, 1997), and collaborative information sharing (Bentley *et al.*, 1995). Bentley *et al.* (1995) stated that the world wide web offers huge potential to CSCW developers. This potential is achieved through:

- platform, network, and operating system transparency
- integration with end-user environments and application programs
- a simple and consistent user interface across platforms
- an application programmer interface for 'bolt-on' functionality
- ease of deployment facilitating rapid prototyping.

Similarly, Roberts (1996) stated that with regard to supporting groupware, the Web offers several advantages including: non-proprietary, wide choice of products and vendors, and easier administration. Dix (1997) in his analysis asserted that among the reasons behind the success of using the Web for cooperative systems are: a core initial user community, the integration of existing information, the use of de facto standards, the spanning of organisational boundaries, and a software platform which is public domain, cross-platform and extensible. Trevor *et al.* (1997) stated the most important reason given by Dix is that the Web has reached a critical mass, which is an important point as proposed by Grudin (1994). All these advantages have made the Web an important platform for CSCW development (Trevor *et al.*, 1997). Bentley *et al.* (1995) further added that the Web based collaboration system is classified into:

- Systems that are purely Web-based - Use standard WWW clients, HTML, and HTTP, and only extend server functionality using CGI (Common Gateway Interface)
- Systems with customised servers - Require special-purpose servers to provide behaviour beyond the possibilities offered by CGI
- Systems supported by customised clients - Require modified clients, often to support non-standard HTML tags
- Systems that are Web-related - May provide world wide web interface but support only limited interaction.

Bentley *et al.* (1995) proposed a system that extends the world wide web by providing a set of basic facilities for collaborative information sharing. The BSCW (Basic Support for Cooperative Work) Shared Workspace system is a document storage and retrieval system extended with features to support collaborative information sharing (Bentley *et al.*, 1995). The system can be classi-

fied as the second type in the classification given above. The system server maintains a number of workspaces which are accessible from different platforms using a standard W3 client. Bentley *et al.* (1995) stated that each workspace contains a number of shared information objects. Workspace members can perform actions to retrieve, modify, and request more details on these objects. The objects currently supported are documents, links, folders, groups, and members (Bentley *et al.*, 1995).

Trevor *et al.* (1997) argued that among the problems for CSCW applications which adopt the Web is that the HTTP is inherently stateless. Stateless here means that a web page cannot remember its previous state. A solution proposed by Trevor *et al.* (1997) is to provide some additional infrastructure which supplements the Web-based application. This architecture, which is called MetaWeb, allows information to flow both ways between user and the application, supporting immediate feedback of the user's actions to the application and notifying changes of application state to the user (Trevor *et al.*, 1997).

Ciancarini *et al.* (1996) presented an architecture to co-ordinate distributed applications on the Web. PageSpace uses distributed agents that coordinate their exchange of services using coordination technology. The architecture consists of three agents, namely: Alpha agents for user-interfaces, Beta homeagent to represent the user on the net, and Delta agents to form the applications (Ciancarini *et al.*, 1996). The coordination of agents is performed using a shared space and primitives that operate on it.

Palfreyman and Rodden (1996) presented a generic awareness mechanism for use across the internet. The protocol is intended to convey the presence of users to other Web users. The model of the extension proposed is to define a new protocol and implement new server systems which will run along side the current Web servers. This resulted in each site needing to run two

servers, normal the HTTP server and the awareness protocol server (Palfreyman & Rodden, 1996).

The WWW and Software Development

There also exist tools that support software development on the Web. Perpich *et al.* (1997) proposed a code inspections tool on an intranet Web. As a justification for this, Perpich *et al.* (1997) asserted that the asynchronous collection of inspection results is at least as effective as the synchronous collection of those results. By exploiting the information dissemination qualities and the on-demand nature of information retrieval of the Web, and the platform independence of browsers, the tool integrates seamlessly into the current development process. The inspection process based on hyperCode is divided into three phases: preparation, collection, and repair (Perpich *et al.*, 1997). The preparation phase is identical to other inspection process. The inspector preparation and collection is done concurrently. The resolution is done by the moderator and the author as part of the repair phase.

Kaiser *et al.* (1997) presented OzWeb, which is a general architecture for hypermedia subwebs, and groupspace services operating on shared subwebs, based on world wide web technology. Hypercode environments intertwine design, implementation and evolution, cross-referencing and assisting users in generating, retrieving, updating and exploiting all on-line materials that may be relevant to a project (Kaiser *et al.*, 1997). OzWeb provides project-specific organisation and search, process, and cooperative transactions to Web materials.

6.2.2 Java Technology

Java was introduced in 1995. It started as a research project to developed advanced software for a wide variety of network devices and embedded system (Linden, 1996; Sun, 1995; Cornell & Horstmann, 1996). It started out as a programming language called Oak in 1991 (Linden, 1996). Oak was part of a research project to develop advanced software for a wide variety of networked devices and embedded systems (Sun, 1995). Java was based on C++ without some of its negative features and new principles and structure were inherited from other object-oriented language, such as Eiffel, SmallTalk, Objective C, and Cedar/Mesa (Sun, 1995; Linden, 1996). Java contains libraries highly-tuned to the internet environment (Linden, 1996).

Java is designed to enable the development of secure, high performance, and highly robust applications on multiple platforms in heterogeneous, distributed networks (Sun, 1995; Cornell & Horstmann, 1996). Java is also architecture neutral, portable, and dynamically adaptable. The language is object oriented from the ground up and is multi-threaded and interpreted (Sun, 1995). Java applets run inside a Web browser and java applications run independent of a Web browser (Cornell & Horstmann, 1996).

Among the features that are of interest to this research are the security aspect, architecture neutrality, and portability. Security is of paramount interest in a distributed environments. The security features designed into java allows applications to be construct that are secure from intrusion by unauthorised code attempting to get behind the scenes (Sun, 1995). The architecture neutral and portable aspects are important because it allows the program to be deployed into heterogeneous networked environments (Sun, 1995). The use of bytecodes allows java to be architecture neutral. The architecture neutral and portable language environment is know as the Java Virtual Machine (Sun,

1995). The interpreted feature allows the bytecodes to be executed directly on any machine where the interpreter is running. The multithreading capability provides the way to build applications with concurrent threads of activity. The dynamic characteristic provides the capability to link classes as needed. New code modules can be linked in on demand from a variety of sources (Sun, 1995). Java standard classes also provides all the basic blocks necessary for client-server implementation (Flynn & Clarke, 1995).

Java and CSCW

Lee *et al.* (1996) constructed the CBE (Collaborative Builder's Environment) as a toolkit for creating extensible collaborative environments. CBE provides user-extensibility by allowing a 'collaboratory' to be constructed as a co-ordinated collection of group-aware applets (Lee *et al.*, 1996). Lee *et al.* added that to support dynamic reconfiguration of shared workspaces and to allow access over the Internet, CBE uses the metaphor of rooms as the high-level grouping mechanism for applets and users.

Trevor *et al.* (1997) in the construction of MetaWeb, as discussed in the section on WWW and CSCW, use java in the implementation. Ciancarini *et al.* (1996) also utilised java in their implementation PageSpace which was also reviewed in the same section.

6.2.3 Object Oriented Paradigm

In Chapter One, the object oriented paradigm was introduced as one of the methods used in computer software system development. It is a system development abstraction which addresses some of the problems the software industries have with today's system development methods. Rumbaugh *et al.* (1991)

stated that superficially, the term object oriented means that software organisation is a collection of discrete objects that incorporate both data structure and behaviour. Object-orientation features include encapsulation and inheritance. Some of the key benefits offered by object-orientation are reusability, productivity, flexibility, maintainability, evolvalibity, and quality of the software (Huff, 1993).

Definition and Taxonomy of OOP

Booch (1990) stated that since 1970s, a number of principles that help manage the complexity of the development have been identified, namely: abstraction, information hiding, modularity, localisation, uniformity, completeness, and confirmability. Of these seven fundamental principles listed by Booch (1990), abstraction and information hiding provide the greatest leverage for management of complexity. The evolution of high-order programming languages reflects the drive towards higher levels of abstraction.

A language is object-oriented if and only if it satisfies the following requirements: it supports objects that are data abstractions with an interface of named operations and a hidden local state; objects may have an associated object type; and types may inherit attributes from supertypes (Cardelli & Wegner, 1985). Object-oriented programming concentrates upon mechanisms for defining abstractions from the problem space, collections of such abstraction, and relationships among objects and classes.

On the issue of object-oriented design, Booch (1990) wrote that it is fundamentally different from traditional functional methods, for which the primary criteria for decomposition is that each module in the system represents a major step in overall process. Booch said that object-oriented design should be

viewed as just one approach that may be applied to help manage the complexity of a fairly broad set of problem domains.

Booch stated that an object may be defined as an entity that includes the following: has state; is characterised by the actions that it suffers and that it requires of other objects; is a unique instance of some class; is denoted by a name; has restricted visibility of and by other objects; and can be viewed either by its specification or by its implementation. Booch further added that to enhance reusability of an object or class, all operations should be primitive. A primitive operation is one that can be implemented efficiently only if it has access to the underlying representation of the object. Booch stated that the major steps in object-oriented design are the following: identify the objects and their attributes; identify the operations suffered by and required of each object; establish the visibility of each object relation to other objects, establish the interface of each object; and implement each object.

Booch's method of object oriented design is only one of many object oriented methods that exist. Rumbaugh *et al.* (1991) presented an object oriented development methodology called Object Modeling Technique. The methodology includes the following stages: analysis, system design, object design, and implementation. Meanwhile, Jacobson *et al.* (1992) proposed Object-Oriented Software Engineering which develops five models: the requirements model, the analysis model, the design model, the implementation model, and test model. Other object-oriented methods includes Object Oriented Analysis (Coad & Yourdon, 1991a) and Object Oriented Design (Coad & Yourdon, 1991b) as proposed by Coad and Yourdon.

OOP and CSCW

Achmatowicz (1994) said that real-time, object-oriented groupware must have the following characteristics:

- distributed - because the users of a groupware session will not be connected to the same machine
- shared environment - users can access shared document in a shared editor
- highly interactive - users of a shared editor will not want delay in carrying out editing tasks
- closely coupled with other participants - information about other participants is required in many cases for successful group work to take place and they are given mechanism to communicate with each other
- real-time notification of events - for example, an update by other users in a shared editor must appear instantly on other user screens.

Achmatowitz (1994) further stated that even though groupware systems are multi-user systems, they differ from traditional multi-user systems in their requirement for close coupling of user interactions. Traditional multi-user systems focused on separating users from each other. Groupware application architectures generally fall into one of two classes: centralised or replicated. Ahmatowicz (1994) proposed an object groups approach which is based on replicated architecture. An object group must have the following characteristics: named collection, interface, and communication. An object group itself is based on process groups, which views a distributed application as consisting of processes, process groups, and broadcast events.

Likewise, Prinz (1994) presented an object-oriented organisation model, TOSCA, to support the modelling and representation of organisational information for the support of CSCW. The system develop by Prinz (1994) allows the

provision of this kind of information both to applications via a service interface and to users via a multi-media user interface. TOSCA is composed of two major components, a organisational information server and the organisation information browser (Prinz, 1994).

The work of Brown & Najork (1996) centred around extra feature to support distributed computation where computation can span machines across the Internet. Brown & Najork assert that high-level support for distributed computation makes it easy to write groupware and CSCW applications as active objects. The environment is based on Obliq, an object-oriented scripting language, and the active objects written is called Oblet (Obliq applet) (Brown & Najork, 1996).

Smith *et al.* (1991) put forward a framework for modelling organisational processes. An object-oriented environment, the Activity Model Environment (AME), was used to explore the models. The AME prototype attempted to deal with the difficult problem of modelling group communication processes in organisations.

The needs for distributed, client-server based systems coincide with the encapsulated, message passing paradigms of object-based software. To function within increasingly complex, network-based environments, programming systems must adopt object-oriented concepts (Sun, 1995). Thus, java provides a clean and efficient object-based development environment (Sun, 1995).

6.2.4 Hypertext

Definition of hypertext was presented in Chapter One. To recap from the introduction, hypertext consists of nodes of information and links between them. The concept was first mooted by Bush (1945) in 1945 and first coined by Nelson

(1981) in 1965. In hypertext, the links between the nodes are machine-supported and the movement between the two nodes takes place automatically. Hypertext would enable the reader or learner to view information or learning material in a nonlinear (Shneiderman & Kearsley, 1989) or nonsequential (Nielson, 1990) manner.

Application of Hypertext

McKnight *et al.* (1991) assert that early examples of hypertext can easily be found. The first of two early examples of hypertext that they suggest is any text which refers to another text. This is basically, two nodes of information with the reference forming the link. The other example is any text that uses footnotes, which can be seen as containing nodes of information with the footnote marker providing the link or pointer from one node to the other. Current examples of hypertext includes dictionaries, encyclopaedias, product catalogues, creative writing, technical documentation, and software engineering (Shneiderman & Kearsley, 1989; Nielsen, 1990).

HTML (HyperText Markup Language) and SGML (Standard Generalised Markup Language) is a form hypertext. The SGML standard for defining formatting in text documents was established in 1986 (Goldfarb, 1990). Among early applications of SGML are the Electronic Manuscript Project of the Association of American Publishers and the documentation component of the Computer-aided Acquisition and Logistic Support initiative of US Department of Defence (Goldfarb, 1990). HTML is a subset of SGML (Vaughan-Nichols, 1997). HTML, is a computer language that enables links to be embedded to other documents within a Web page, thus creating a third dimension (Berners-Lee *et al.*, 1994)

Hypertext and CSCW

Wang (1993) wrote that complicated documentation maintenance environments require sophisticated computer support systems in order to maintain relationships between various parts of documents. He suggested that, in hypertext technology, it is possible to easily browse document contents. Wang (1993) proposed an extended hypertext data model, InterSect_DM, which can be supported by an object oriented database system. This is a general purpose hypertext data model and basic building blocks and operations. The model was formalised using the formal specification language Z in order to define the fundamental mechanisms and semantic constraints of operations. A prototype, InterSect, was implemented. It is designed to meet the requirements of complex documentation environments.

In another study, Zeb (1993) proposed the MUCH (Multiple Users Creating Hypertext) systems which provides collaborative authoring tool, browsing, searching, and annotating. This is from a series or prototypes of a collaborative authoring tool. The latest prototype provides a frequency of occurrence of a selected word across the outline as a tool for finding the required information.

Hahn *et al.* (1991) designed a hypermedia group authoring environment, CoAUTHOR. The system provides a real-time environment for multiple authors. The CoAUTHOR prototype consist of three severs. These are: a knowledge base management server, a multimedia database server, and a real-time conferencing facility. Hahn *et al.* (1991) explains two type of client. The Interactive toolbox supports hypermedia editors and browsers. And the Group Toolbox contains methodologies for voting, structured argumentation, management of authoring roles, etc.

Fowler *et al.* (1994) presented Virtual Notebook System (VNS) which is a distributed collaborative hypertext system based on single metaphor. The VNS is intended to emulate and enhance the use of an ordinary paper notebook (Fowler *et al.*, 1994). The VNS architecture is a multi-layered model consisting of a persistent object-repository, a transaction and lock manager, and client applications.

Hypertext and Software Development

Examples of hypertext application in software development includes work by Ferrans *et al.* (1992) and Fletton (1990). Roth (1994) categorises hypermedia support for software development into six categories:

- Argumentation or collaboration-based development activities
- Software interface development
- Documentation
- Creation and maintenance of code
- Support for software artefact reuse
- Prototyping.

Ferrans *et al.* (1992) proposed a framework called HyperWeb, which supports the construction of hypermedia-based software development environments. It co-ordinates the activities of an integrated set of tools through a message server, uses an object oriented database to store software artefacts, and supports hypermedia linking of these software artefacts (Ferrans *et al.*, 1992). The features include support for document linking, source code annotation and restructuring, and modification request tracking.

In another work, Fletton (1990) presented a hypertext-based technique for browsing and documenting source code using a purpose-built prototype. The technique was originally aimed at software maintainers to allow them to redocument a program, but it can also be applied to the production of maintenance documentation during development (Fletton, 1990). Fletton stated that the system uses hypertext links to allow programmers to locate areas of interest rapidly and efficiently in source code and to examine and update related documentation.

6.2.5 Enabling Technology Summary

The WWW in combination with java provides a flexible platform for the groupware across the network. Having reached the critical mass, acceptability of WWW is not an issue any more. With the hypertext property of HTML, it provides flexibility in arranging the information and the document. The object oriented property of java provides flexibility in adding new components to the system.

6.3 FUNCTIONAL ARCHITECTURE OF WEB-BASED GROUPWARE SYSTEM

The development of the functional architecture of a groupware system based on the world wide web requires the knowledge of how the world wide web works. In the most simplistic view, the user only needs to know how to use the web browser, which in turn interacts with the world wide web (see *Figure 6 - 1*). The web browser's basic function is handled through the HTML which can interact with executable content and gateway programs. The HTML documents interact with the web server, as explained in Chapter Four. The HTML docu-

ments can also be used as an interface and send data to a gateway program. In the case of executable content, it can be downloaded and run on the user's machine. Each of these components can interact independently with other machines in the world wide web. Chapter Four described how a web browser interacts with a web server, and the interaction between java executable content and server programs.

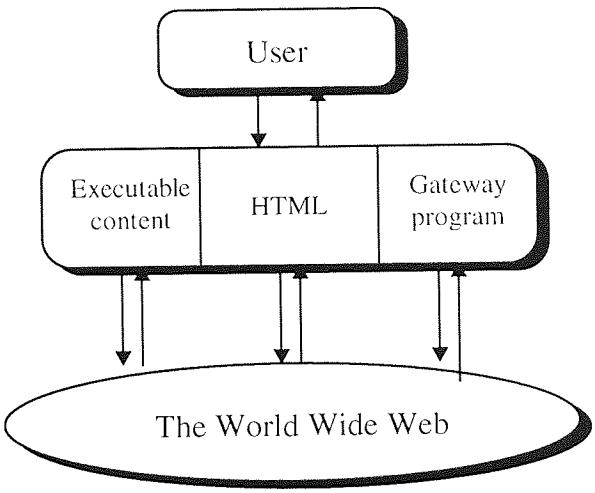

Figure 6 - 1 : *User Interacting with the Web*

6.3.1 Client-Server Architecture

The implementation of the prototype can be based on two architectures, namely the client-server or the peer-to-peer architecture. The peer-to-peer architecture requires each of the clients to have a different copy of program code and did not provide the flexibility that is required. A different set of code must also be provided for different computing platforms. On the other hand, the client-server architecture provides a centralised structure with the majority of the code located in the server. The client-server architecture also provides the basis for the flexibility required, as presented below.

In order to arrive at the functional architecture for a groupware system on the web there is a need to look at the general client-server architecture itself. In a simple client-server system, we have the client, the server, and the network that connects the two (see *Figure 6 - 2*). A server program runs on a server host and provides information and data to the client programs. A client program runs on a client computer and normally there is little processing here as it is mostly done on the server side. The clients and the server interact according to a pre-defined protocol. The partitioning of functions between server and client give an added advantage: the clients doesn't have to know the platform on which the server runs nor the operating system of the server, as long as both client and server communicate according to a pre-defined protocol. The server also doesn't have to know about the details of the implementation of the clients system. Every client is independent of each other in terms of the platform they run on.

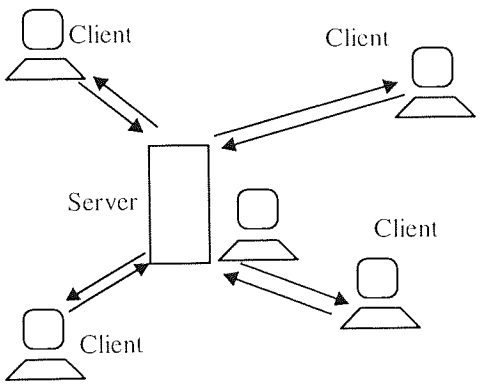

Figure 6 - 2 : *A Simple Client-Server Architecture*

The client-server architecture provides the basis for flexibility in terms of multiple computing platform implementation. The existence of the server is also needed in order to implement the shared state facility. The architecture is also consistent with the technology that is going to be used in the implementation, namely the world wide web.

6.3.2 Web-Based Groupware Architecture

The client-server architecture can be implemented using vendor-specific solutions, but the problem of the middleware and the difficulty for the server program to support different client computing platforms makes the web alternative a better solution. In the web architecture, the problem of middleware and support for different computing platform do not arise. Based on the general client-server architecture and the world wide web model, we can derive the functional architecture for a web-based groupware system. In this architecture, the basic architecture of a client-server system is maintained except that the connection between the client and the server is handled by the world wide web (see *Figure 6 - 3*). The server for the web-based groupware is part of the world wide web itself. This means that the server is accessible by any other client machines that is connected to the world wide web. All users of the groupware access the system through the client system which is connected to the world wide web.

A web-based groupware system can be implemented in the internet, intranet, or extranet environment. In terms of enabling technology, there is not much difference between these three environments. In the internet set-up, the web server is accessible from anywhere else in the Internet and security access control is minimum. In an intranet set-up, the web server and its client sit behind a firewall. In an extranet set-up, the server and the majority of its client sit behind a firewall. Access from the internet is limited to certain people, address, or companies by setting-up the server configuration and the firewall configuration.

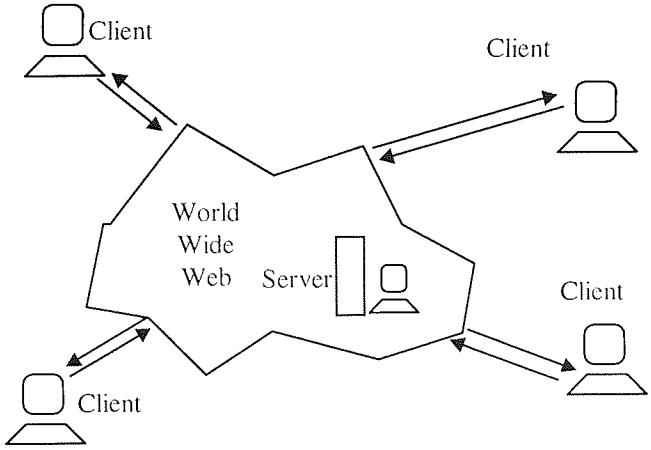

Figure 6 - 3 : *General Architecture of Web-Based Groupware System*

6.4 FLEXSIG ARCHITECTURE

FlexSIG architecture is based on the functional model described in Chapter Five. The architecture presented here is derived from the functional model, making use of the enabling technologies.

6.4.1 Outline of Architecture

The architecture of FlexSIG can be viewed from several angles. The system architecture presents the system view of the architecture. The database storage architecture specifies the internal organisation of the file system. The user interface architecture describes the interface between the system and the users. Finally, the security access outlines the level of access control in the system.

Based on the general architecture of the web-based groupware system from the previous section, we can illustrate the outline of the FlexSIG architecture. The FlexSIG architecture has one server running the server program.

Members of the software inspection team, namely the moderator, the inspectors, and the author, each connect to the server as a client. FlexSIG architecture is constructed in an Intranet environment in which the FlexSIG is implemented behind a firewall for protection against unauthorised access from the Internet. The concern about security is because of the nature of the software inspection process itself, which deals with the sensitive project documents. The documents are usually secret and should not be accessible by anybody from outside the organisation.

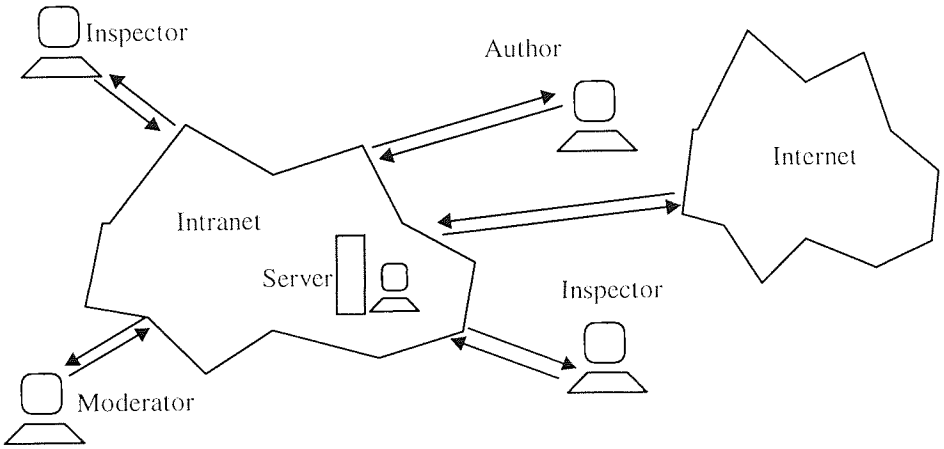

Figure 6 - 4 : *The Overall Architecture of FlexSIG*

6.4.2 System Architecture

The FlexSIG architecture utilises world wide web technology in order to achieve its objective. The system architecture consists of two sections, namely: clients and server (see *Figure 6 - 5*). The client machines run the web browser and the server machine runs the web server (see *Figure 6 - 6*). The system is initiated when the client requests the starting page of FlexSIG from the web server (see Step 1, *Figure 6 - 5*). The server responds by downloading the page to the client machine and the client browser displays it (Step 2). Depending on

the commands or responses made by the clients, further action is executed by the server. There are three types of response:

- If the response is to access another page of information, the request is relayed back to the web server (Step 3b). The appropriate page is then downloaded from the server to the client and displayed to the user (Step 4b). An example of this kind of operation is when a client requests briefing material or a specification document.
- The response from the server can also be to start an applet running on the client (Step 3c). In this case, the client request is again conveyed to the web server and the appropriate applet is downloaded to the web browser on the client machine (Step 4c). The applet is then interpreted and displayed on the browser. An example of this type of response is the request to change a user's password.
- The third kind of response is a request to get information from another web server (Step 3a). The request is forwarded to the appropriate web server and the information is downloaded and displayed by the client browser (Step 4a).

The applet displayed on the client machine can accept further input from users and display further information. These applets communicate directly with java server programs running on the web server (Step 5). The java server programs control several database files directly (Step 6a). The server program is also responsible for accessing the document to be inspected and producing the report (Step 6b). An example of this kind of operation is the request to browse a document to be inspected. The client enters the file name in an applet, which we assume has already been downloaded to the client machine. When the client submits the request, the applet will connect itself to the server program using a pre-determined port number and pass the request and

the parameter to the appropriate server program. The server program will then interpret the request and open the file to be inspected. The file is then passed back to the applet on the client machine to be displayed. The server program can also access another HTML document and pass it back to the client (Step 6c).

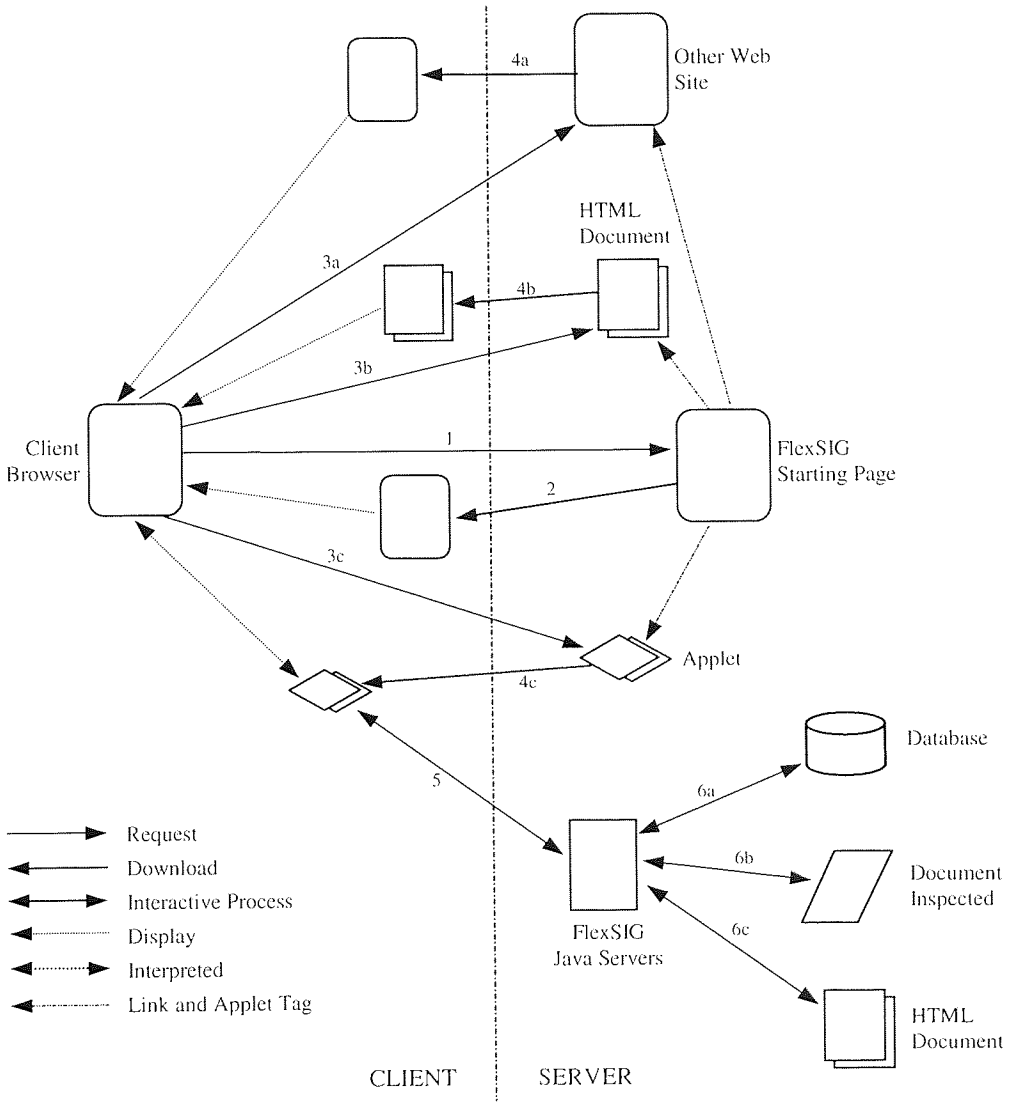

Figure 6 - 5 : *FlexSIG System Architecture*

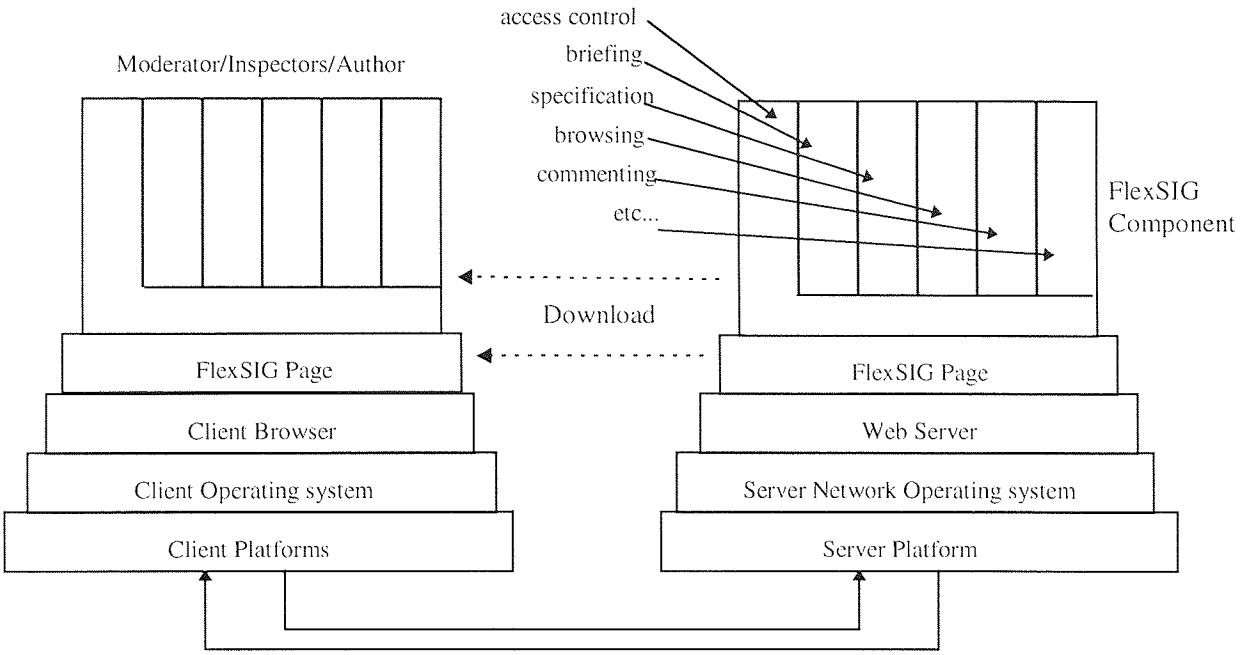

Figure 6 - 6 : *Client-Server Interaction in FlexSIG*

The system architecture presented provides the support for the flexibility required. The client, regardless of the computing platform, can download the system and run it on their machine, as long the browser on the user machine support java. The applet based component also provide the unobtrusive capability to the component. The applet is downloaded only when needed and did not interfere with other functions. The critical mass factor also is taken care of by utilising the world wide web.

6.4.3 Database Storage Architecture

All data files are stored in the web server. No information is kept on the client side. The database is accessible only through the java server program (see *Figure 6 - 7*). The client programs, in the form of java applets, have no direct access to the database. The data from the client is passed to the server program along with the operations that need to be performed on the data. The

java server program then accesses the database and performs the requested operation using the data and instruction passed from the client.

Each java server program can access certain files only. No server program has the right to access all files. For example, the user authentication server program only has access rights to user password and role files. Access to files depends on the clients role. The moderator can access all files. The inspectors have no access to system configuration files. The author cannot modify any file in the database except the password file. The access right to the database implementation reflects the policy on the information accessibility discussed in Chapter Five.

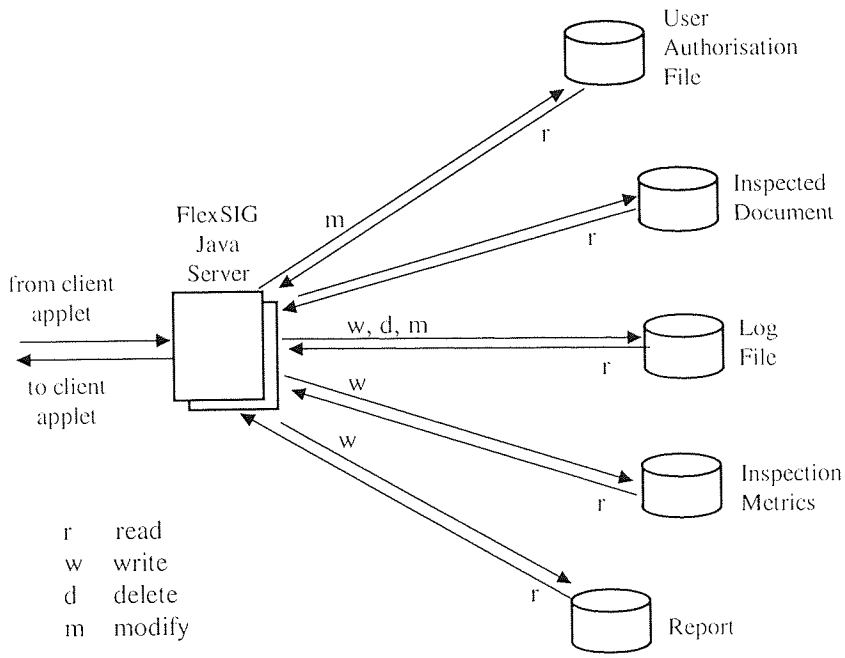

Figure 6 - 7 : *Database Storage Architecture*

6.4.4 User Interface Architecture

The FlexSIG user interface is based on the web browser's interface. Regardless of the client's platform or the type of browsers the client uses, the user inter-

face should look identical. The basic interface is a browser window divided into three frames (see *Figure 6 - 8*) (Tatters, 1996). One frame is reserved for displaying the options available to the user. This is located on the left of the window. The bottom frame is use to display this author's information. The biggest frame is used to display current information, further options from the menu, or an applet. This is referred to as the information frame. The frames provide the basic graphical user interface which requires a menu and a window to display the information (Marcus, 1992). The menu frame provides the menu and the information frame provides the window to display further command or information.

The windows are arranged this way in order to provide the different menu according to the role without affecting the information frame. The division of the menu and the information frame also results in the menu being always visible and the changes in the information menu not having any effect on the menu frame.

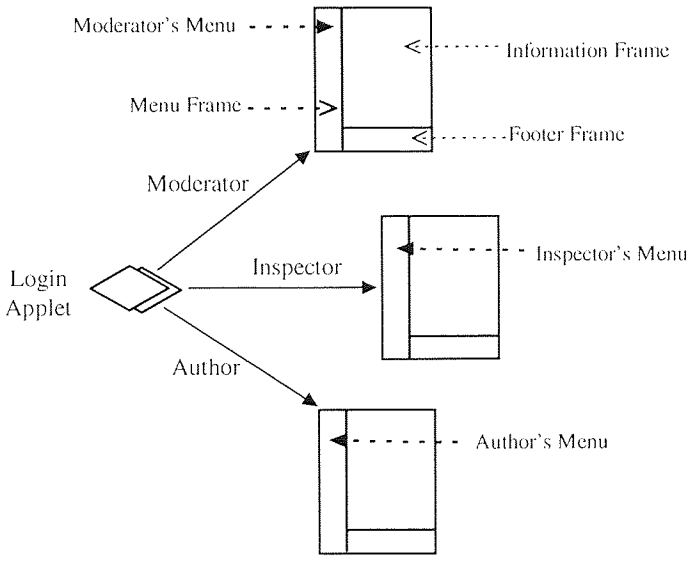

Figure 6 - 8 : *User Interface Architecture*

The role of the user determines the type of menu they see. When the user does a login into the system, the user authentication server determines the role of the user and displays the appropriate user interface (see *Figure 6 - 8*). The moderator will have access to all options. The inspector menus are the same as the moderator menus, minus the system configuration option, metrics and report option, and log compilation option. The author menu is the same as the inspector, except that the author cannot enter any log and cannot participate in the voting process.

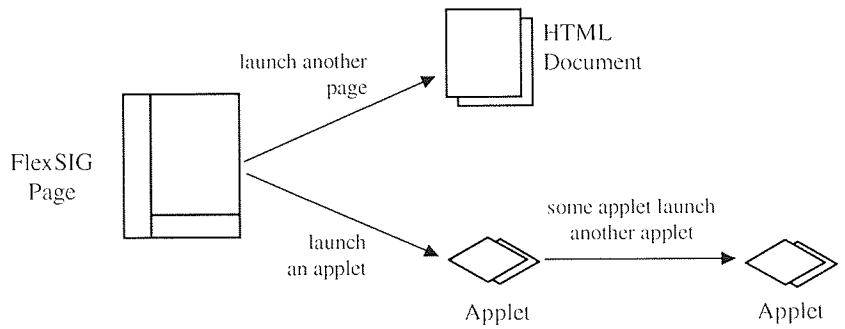

Figure 6 - 9 : *Type of User Interface Response*

Further actions depend on the options chosen by the user. As a general rule, the information frame is always used to display sub-menus. For other than the sub-menu, a new browser window is launched and may contain another HTML document or an applet component. This is to allow more than one component to be used at the same time and also allow the component to be closed without affecting other component. There are four possibilities that might arise (see *Figure 6 - 9*).

- First is that by choosing a sub-menu option, information in the form of HTML document is displayed in the information frame. An example of this is a sub-menu for the help option.

- The second possibility is that an applet is displayed in the information frame. The send mail applet selection is one example.
- The third possibility that can arise is that another browser window is launched and displays information in the form of HTML document. One example from this kind of response is when the user chooses one of the briefing materials from the briefing sub-menu.
- The fourth type of response is that another browser window is launched and an applet is displayed in it. Choosing an option to browse a document to be inspected produces this kind of response.

An applet displayed in another browser may launch another applet. An applet that handles the pop-up message spawns another message applet on another client machine as well as the initiating machine.

6.4.5 Security Access Architecture

Security is a major issue for a system that is based on the world wide web. In FlexSIG, there are basically two issues that need to be addressed. The first is on the issue of access control of the system. The second is the issue of the security of information transferred between the client and the user through the world wide web infrastructure.

The access control issue is handled by the authorisation of the client during login into the system. This can be divided into two problems. The first is in the login process of the client into the system (see *Figure 6 - 10*). In this problem, any attempt to access the FlexSIG homepage will result in a login applet being displayed on the client's browser (Step 1,2). The client is compelled to enter a username and password. The information is passed back to the server program on the web server (Step 3). The username and the pass-

word are then compared against the authorisation file (Step 4). The FlexSIG homepage is displayed when the username and the password are confirmed valid (Step 5a, 6). Otherwise a warning message is generated and displayed to the client (Step 5b). This is not the only way to implement access control. Another methods that can be used is the Common Gateway Interface form. The java applet implementation is chosen in order to be consistent with the rest of architecture.

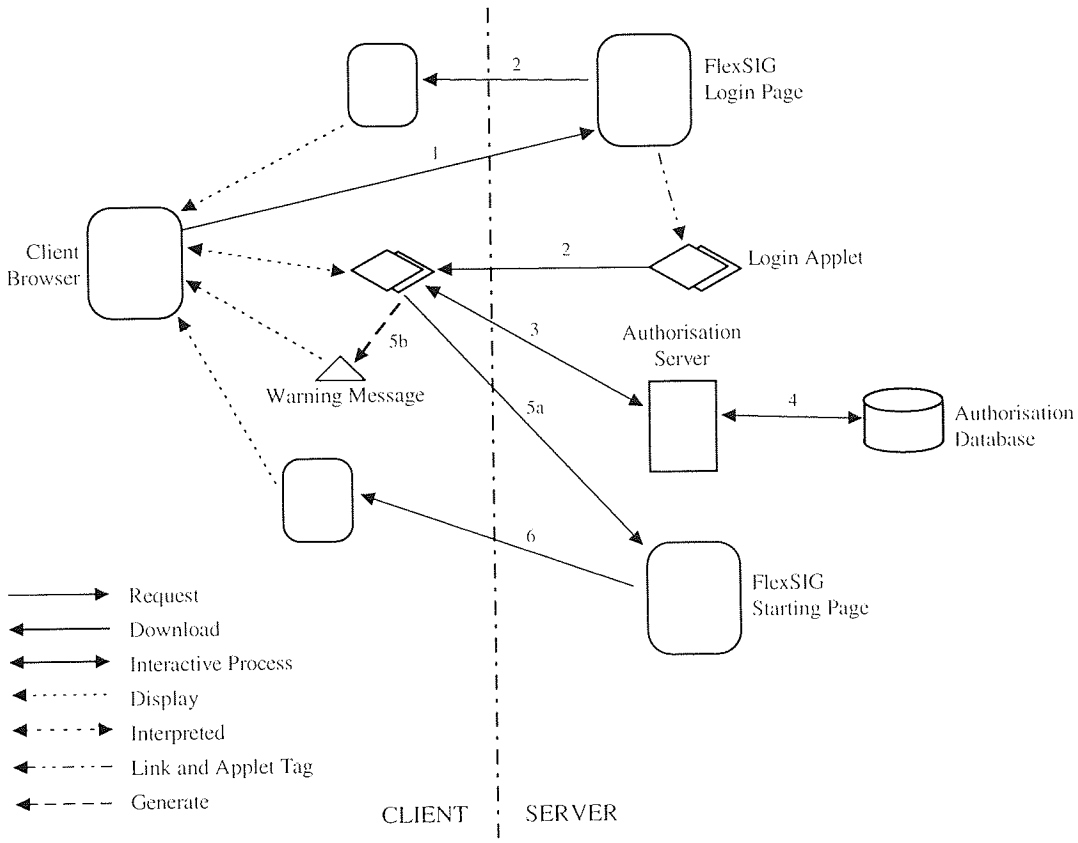

Figure 6 - 10 : *Security Access Architecture*

The second problem is determining the level of access for users with regard to the database files and actions that can be taken by the users. This can be dealt by checking the roles of the client from the authorisation file as explained in Chapter Five (summarised in *Table 6 - 2*). The moderator has the highest level of access, followed by the inspectors and the author. The access

rights of each roles are shown in *Table 6 - 2*. The rights are assigned in line with the framework of each of the roles defined in Chapter Five.

Role	Moderator			Inspector			Author		
Doc. Type	Read	Write	Delete	Read	Write	Delete	Read	Write	Delete
Inspected Doc.	√			√			√		
Briefing Document	√	√	√	√			√		
Requirement Doc.	√	√	√	√			√		
Help Document	√	√	√	√			√		
Log File	√		√	√ *	√ *	√ *	√		
Voting Right	√		√ **	√	√	√	√		
System Config	√	√	√	√ ***	√ ***		√ ***	√ ***	
Metrics & Reports	√	√	√						

* Personal entry only
** Override option only
*** Change personal password only

Table 6 - 2 : *Access Control Depending on the Role*

As mentioned previously, the security of information transferred through the world wide web is a concern. Thus, one of the security requirements of FlexSIG is the existence of some form of firewall. The firewall will divide the Intranet from the Internet. Another security issue is the problem of the organisation sensitive document, namely the program code and the specification, sent across the network. There is also an issue on authentication between clients and server. These security issues are however, beyond the scope of this research.

6.5 FLEXSIG COMPONENTS

This section describes the components of FlexSIG briefly and relates the component with the design issues discussed in Chapter 4. The components are described according to their function. Details are provided in Appendix B.

6.5.1 Briefing

The briefing component is implemented in order to support the kick-off meeting phase or the briefing phase of FlexSIG model and thus provides one aspect of the flexibility. This component is implemented by utilising the HTML documents. The document is prepared by the moderator and placed in the server. The location of the HTML documents is made known to the users during the initiation phase.

6.5.2 Specification

The specification component contains requirement specification or design specification material or both, depending on the document inspected. The component is provided in order to support timely on-line access to the supporting document. The implementation of the specification component is also based on the HTML documents, but the contents are not prepared by the moderator. In this case, the specification is provided to the moderator by the software project team. The moderator's role is to convert it into HTML format and publicise the location to team members.

6.5.3 Document Inspection

The browsing components allow the team member to browse the documents that need to be inspected. There are two types of browsing method. The first type allows the team member to select documents from a given list. The second type permits the user to select the document that another user is currently inspecting. These components require interaction between the java applet and java server responsible for document handling. The applets are responsible for gathering the filename or the username and the server will obtained the appropriate document based on the information from the applet. The document is then passed back to the applet.

The document viewer component provides the support for document handling by providing a browser to browse the inspected document (see *Figure 6 - 11*). The function of the remote browsing component is to provide a discussion medium during the group synchronous inspection.

6.5.4 Comment Log

The comment log component provides another support for the document handling requirement (see *Figure 6 - 12*). The component provides the flexibility to be used in different working modes. The log file also acts as a shared state mechanism. The comment log component is handled by a java applet. The information gathered is sent back to the server, and the information is stored in the appropriate database. The applet is also responsible for displaying existing log sent by the server during the editing process.

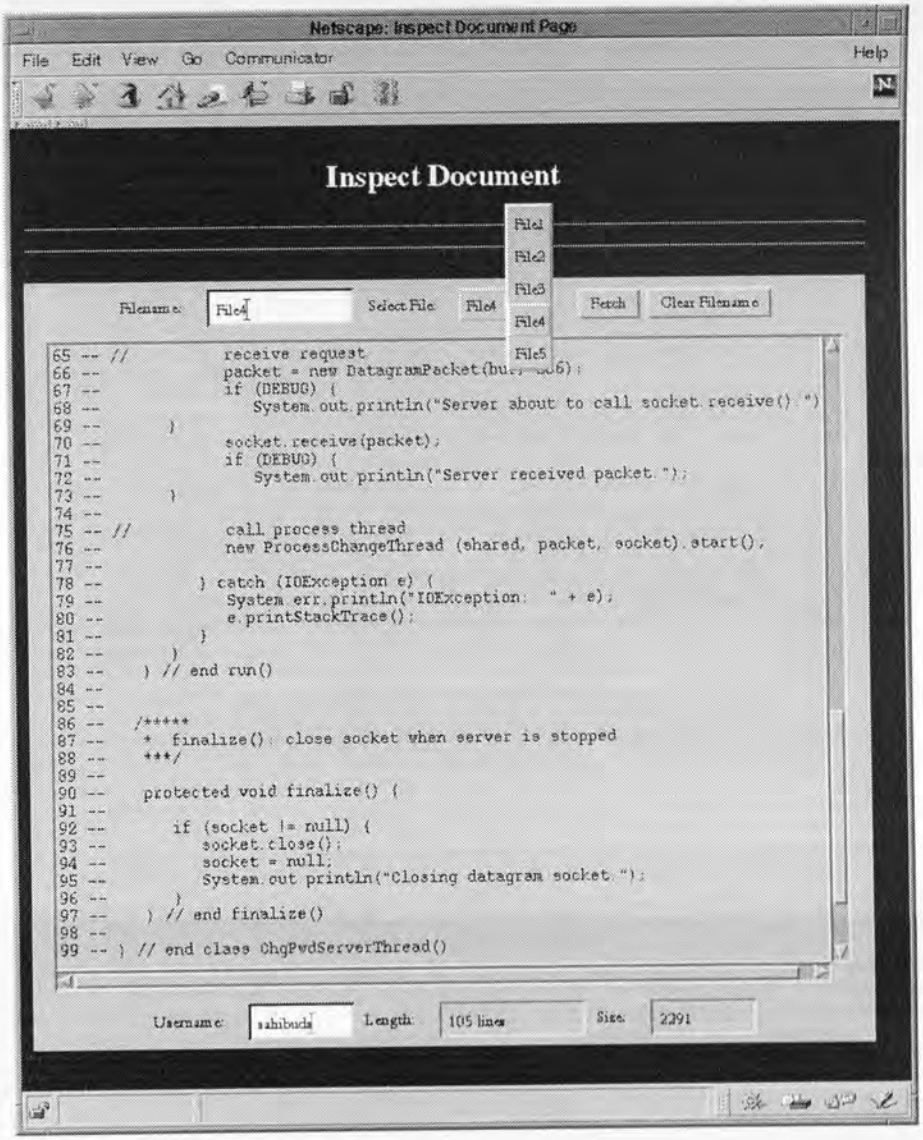

Figure 6 - 11 : *Document Browsing Applet*

6.5.5 Communication

The communication suite can be divided into two major sections. The first category deals with the synchronous mode of communication. The second category deals with the asynchronous mode of communication. The communication component supports both the synchronous group inspection and asynchronous group inspection.

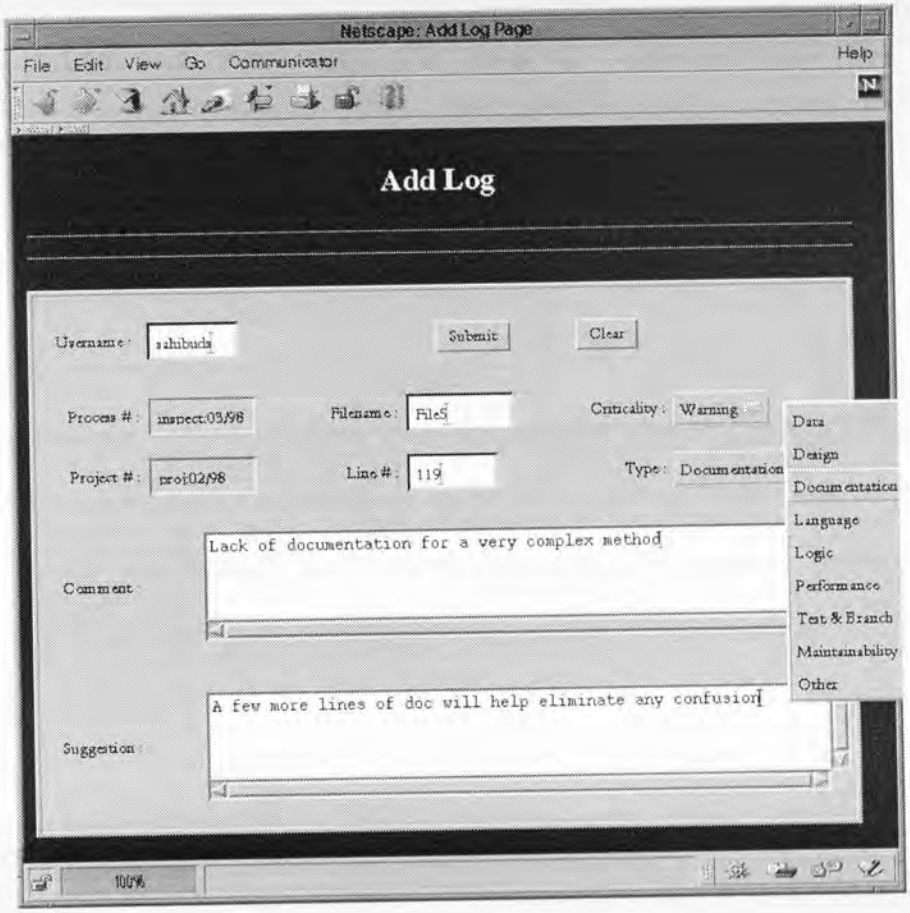

Figure 6 - 12 : *Entering Comment into Log File*

Synchronous Communication

The synchronous communication component provides the support for the synchronous group inspection. The component offer three facilities, namely group chat, private chat, the pop-up message facility. By providing more than one facility, further flexibility can be supported in synchronous communication. The synchronous communication facilities also provide awareness information in terms of knowledge of group and individual activity, and co-ordination. Indirectly, the remote browser also provides awareness information about another individual's activity. The synchronous communication component is served by one server. The server is responsible for maintaining the list of current users.

The server keeps track of the user by prompting for username when the user activates the chat or pop-message applet. The message from the applet is sent to the server, which will then redirect the message to the intended receiver.

Asynchronous Communication

The asynchronous mode of communication can be divided into two categories, namely a direct communication and an indirect communication. In a direct asynchronous communication component, two methods are used - e-mail and bulletin board - to support the asynchronous group inspection. Local flexibility is supported by providing more than one facility. The e-mail and the bulletin board are implemented using an applet and server program. When an e-mail or a bulletin board applet is activated, the server will check the appropriate database for any mail for the user or for any message on the bulletin board. The information is sent to the applet to be displayed. If the user needs to send any e-mail, the mail is sent to the server, which will then forward it to the appropriate mailbox. In an indirect asynchronous communication, the log file itself can be viewed as a shared state, as mentioned earlier. Information on a voting entry in the vote file for any logged item can also act as another shared state.

6.5.6 Help

The help component, which consists of help facilities on using FlexSIG, help on the syntax of the programming language, and help on the operating system usage, is part of the supporting documentation. Similar to the specification component, the help component is provided in order to support timely on-line access to the supporting documents. The implementation of help is based on HTML documents. The contents are based on the standard material available

for that particular language or operating system. The location is then made known to the server program.

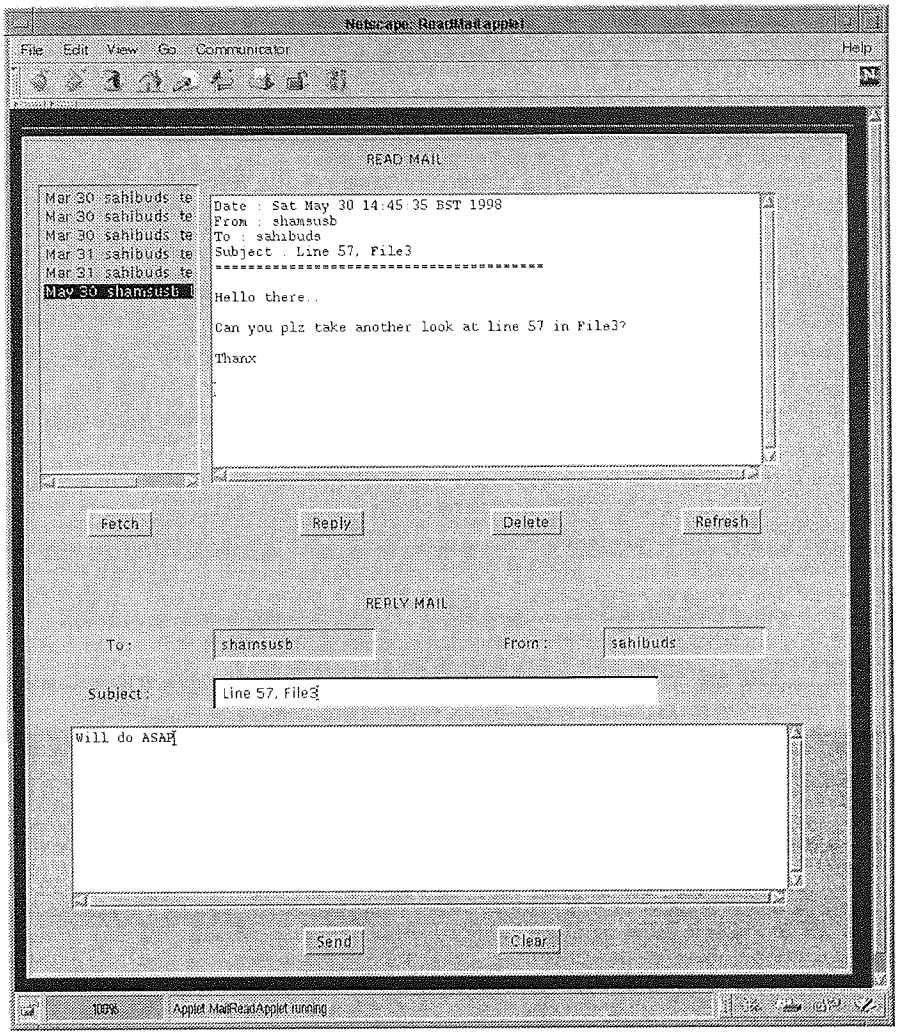

Figure 6 - 13 : *Read e-mail Applet*

6.5.7 Metrics and Reports

The data collection requirement is supported by the metrics and reports component. FlexSIG collects a number of metrics from the software inspection process. Reports produced in FlexSIG are based on the log file and the metrics collected. These components are activated during the consolidation phase and only available to the moderator. The server program responsible for collecting

the metrics will access the appropriate file and collect the metrics. The metrics are stored in another file. The server program for the report generation will access the metrics and the log file and print the report directly to the printer.

6.5.8 Access Control

The access control component is to implement the security architecture and access to the system is handled by the login applet. This component allows the user to modify their own password. The username and password are collected by the applet and passed back to the server. The server will verify the information from the authorisation file and sent the confirmation back to the applet. The modification of username and password information is also processed through the applet which will send the information the server. The server is responsible to make the changes to the authorisation file.

6.5.9 System Configuration

The system configuration component allows the moderator or the system administrator to set-up and maintain the system. The components that need to be set-up before the inspection process are as follows: the briefing, specification, help, documents to be inspected, log file, access control component, and inspection process identification. The implementation is also based on an interaction between the applet on the client side and the server program in the server. The information with regards to the configuration is collected by the applet and sent to the server program for the appropriate action to be taken.

6.6 SUMMARY

This chapter started with the discussion on the implementation issues. The functional architecture of a web-based groupware system which was based on the world wide web and the client-server architecture was presented. The FlexSIG architecture, which includes the outline, system, database, user interface and security architecture was presented. Finally, a discussion on the components of FlexSIG prototype itself was furnished. This includes explanation on the briefing, specification, document inspection, comment log, communication, help, metrics and reports, access control, and system configuration component in relation with the design issue of the prototype.

Discussion on the evaluation of the prototype forms the next chapter.

Chapter 7

USER EVALUATION OF THE PROTOTYPE

7.0 INTRODUCTION

This chapter discusses the third phase questionnaire which is conducted as part of the evaluation of the prototype.

- **Section 7.1** discusses the third phase data collection procedure.
- **Section 7.2** presents the analysis of the questionnaire.

7.1 PHASE THREE DATA COLLECTION

The data collected in phase three is related to the evaluation of the prototype developed in phase two. The analysis on the data collected is used to get some conclusion on the model and prototype development of FlexSIG.

7.1.1 Outline of Phase Three

The third phase is divided into two stages. In the first stage, the sample is asked to evaluate the prototype based on the guidelines given. After the evaluation, the sample was asked to complete a questionnaire which served as a means to gather feedback. The questions ranged from personal background

to their evaluation on FlexSIG. The participants in the third phase were a group of post-graduate students from Division of Electronic Engineering and Computer Science at Aston University.

The data collected from the sample was analysed using statistical tools and used to draw conclusions about the acceptability and flexibility of the model and prototype.

7.1.2 Evaluation of Prototype

The sample was first asked to evaluate the prototype using the guideline given. The evaluation is divided into four sections. The first section deals with the general features of the prototype such as the briefing component, specification component, help component, and accessing the prototype. The second section deals mostly with the asynchronous mode of communication of the prototype such as e-mail. It also asks the sample to evaluate the document viewer and the log entry component. The third section dwells on the synchronous communication of the prototype. Among the evaluations that were carried out was the remote document viewer component, the chat component, and the pop message component. The last section was on miscellaneous components such as changing the password. The complete evaluation guideline is given in Appendix F.

7.1.3 Questionnaire

There are thirty nine questions in the second phase questionnaire given to the subject. Ten questions, similar to the first phase questionnaire, relate to the personal background of the participant. Examples from this type of question are the participants name, occupation, and academic background. Eight ques-

tions relate to the subject experience in software development, software quality, and inspection process. Examples from this category of question are how many years of experience in computer related fields, how many years of experience in software quality, and how many software projects they have been involved in. The final twenty one questions deal with the participant's evaluation of the FlexSIG prototype. Examples from this class of question are do they think that FlexSIG is suitable for asynchronous mode of process, is the help component sufficient for the inspection process, and is FlexSIG is flexible enough for asynchronous mode of process and synchronous mode of process. A blank response is considered as an invalid response. In the questionnaire, some of the questions use a scale of 1 to 4. The value of 1 is a positive response to the question, and meant most likely or less difficult, and the value 4 is a negative response, which meant less likely or most difficult. The complete questionnaire can be found in Appendix F.

7.2 ANALYSIS OF QUESTIONNAIRE ON EVALUATION

Ten subjects participated in this phase. Similarly to phase one, the majority of the subjects are male with only twenty percent female. Eighty percent of the participants were below thirty years of age. The samples are the post-graduate students at the Department of Computer Science and Applied Mathematics, Aston University. Most of them have a degree in computer science. Eighty percent of the valid respondents have a bachelor or a master degree in computer science or related fields.

Ninety percent have more than three years experience working in information technology. All of them have experience in programming. Thirty percent have experience in system analysis. But only twenty percent have ex-

perience in software project management and none have experience in quality management. In the questionnaire, the participants can choose more than one category of experience. The complete listing is in *Figure 7 - 1*. Ninety percent of the subjects have more than three years experience in software development. Seventy percent have had involvement in more than three software projects in their software development experience. The response shows that a large number of the participants have experience in programming and software development.

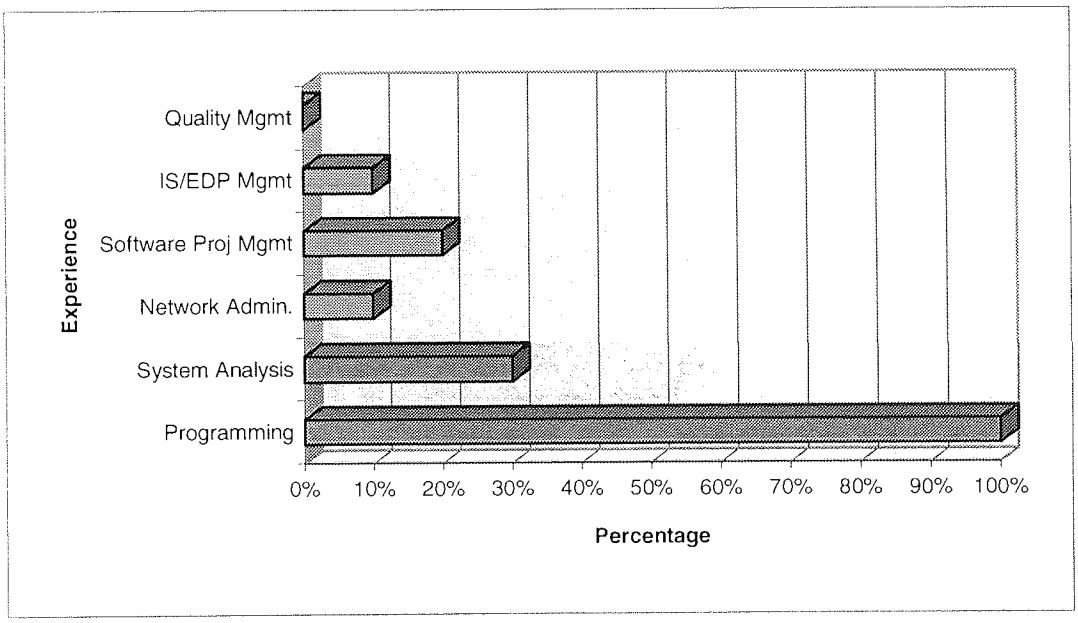

Figure 7 - 1 : *Respondent IT Experience*

Only ten percent said that they received any training or experience in software quality. Ten percent of the respondents have two years or less experience in software quality. Notably, thirty percent of the sample says that they have used some form of inspection or review techniques in software development. This indicates that they used the review or inspection technique without any training. The most common technique used by the respondent was the walkthrough and code inspection with thirty percent saying they have used it. Ten percent of the sample said they have used software audit and formal technical review.

7.2.1 Asynchronous Mode of Inspection

With regard to the sufficiency of the briefing concept in replacing the kick-off meeting in an asynchronous meeting, thirty percent choose 1, sixty percent choose 2, and ten percent choose 3 (*Figure 7 - 2*). The percentage and the mean from *Table 7 - 1* figure shows that in an asynchronous meeting, the briefing concept is sufficient in replacing the kick-off meeting. One of the respondents commented that although the briefing concept may replace the kick-off meeting, a kick-off meeting is still important for face-to-face meeting.

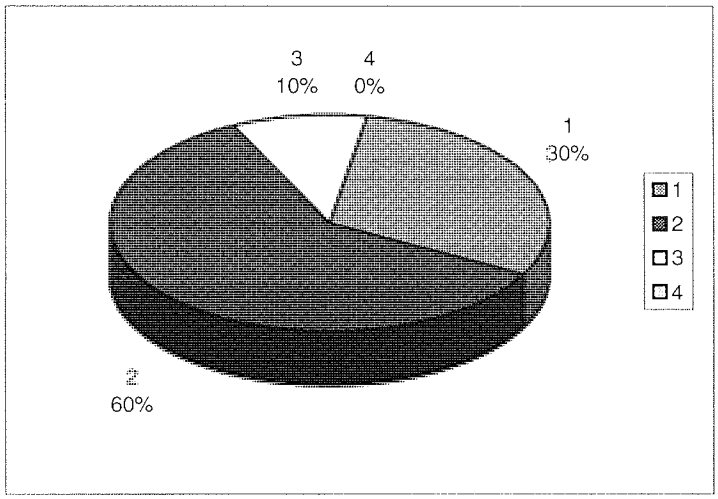

Figure 7 - 2 : *The Suitability of the Briefing Concept in Replacing Kick-Off*

The majority of the respondents also agreed that the HTML is a better option than a paper version for distributing the specification document (see *Table 7 - 1*). Fifty percent choose 1, forty percent choose 2, and ten percent choose 3. The respondents stated that HTML is a better option because of speedier delivery and timeliness.

On the question of the sufficiency of the help concept for the inspection process, fifty six percent choose 1, thirty three percent choose 2, and eleven percent choose 3. The figure from *Table 7 - 1* shows that the help concept is accepted to be satisfactory for the software inspection process.

Questionnaire	Mean	Standard Error	Standard Deviation
Suitability of Briefing in Replacing Kick-Off	1.80	0.20	0.63
HTML a Better Option than Paper	1.60	0.22	0.70
Help Concept Sufficient for Asynchronous	1.56	0.24	0.73
Adequacy of Info. Collected in Log Entry	1.60	0.16	0.52
Suitability of FlexSIG in Asynchronous Process	1.60	0.22	0.70

Table 7 - 1 : *Mean for Questionnaire on Asynchronous Mode*

Forty percent of the respondents choose 1 and sixty percent choose 2 when they were asked on the adequacy of the information collected in the log entry. The statistic confirmed that the information collected is adequate (see *Table 7 - 1*).

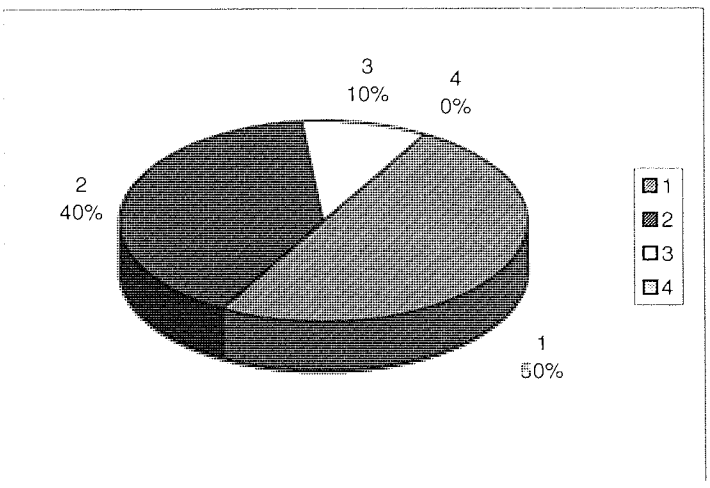

Figure 7 - 3 : *The Suitability of FlexSIG in Asynchronous Process*

Overall, the majority of the respondents agreed that the FlexSIG system is suitable for asynchronous mode of process (see *Table 7 - 1* and *Figure 7 - 3*). Fifty percent choose 1, forty percent choose 2, and ten percent choose 3. Among the suggestions that were given is that the FlexSIG should be implemented over a more secure channel.

7.2.2 Synchronous Mode of Inspection

With regard to the complementary aspect of the briefing concept in relation to the kick-off meeting in a synchronous meeting, seventy percent choose 1 and thirty percent choose 2. The number from *Table 7 - 2* shows that in a synchronous meeting, the briefing concept is a good complement to the kick-off meeting.

Questionnaire	Mean	Standard Error	Standard Deviation
Suitability of Briefing as Complement to Kick-Off	1.30	0.15	0.48
Remote Viewer Swiftness in Accessing File	1.30	0.15	0.48
Ease of Using Chat as a Synch. Communication	1.30	0.15	0.48
Ease of Using Pop Message as a Synch. Comm.	1.30	0.15	0.48
Suitability of FlexSIG in Synchronous Process	1.30	0.15	0.48

Table 7 - 2 : *Mean for Questionnaire on Synchronous Mode*

The respondents gave a positive answer to the question of the remote viewer quickness in accessing the file viewed by another user in the group (see *Table 7 - 2* and *Figure 7 - 4*). Seventy percent choose 1 and thirty percent choose 2.

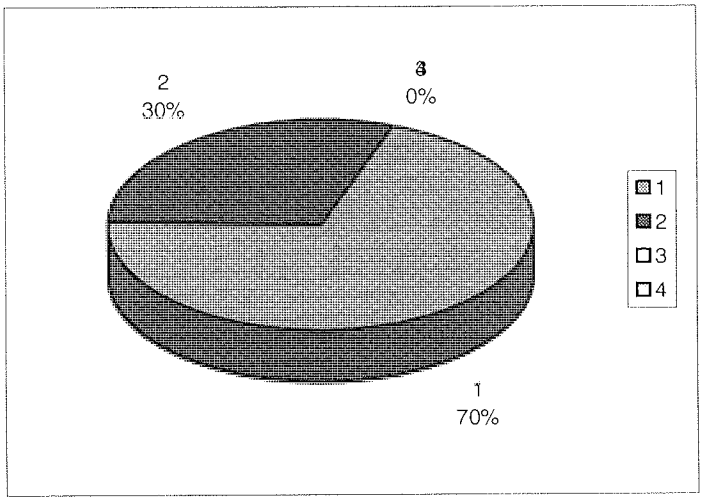

Figure 7 - 4 : *Accesing File using Remote Viewer*

On the ease of communicating with other group members synchronously using chat, seventy percent choose 1 and thirty percent choose 2. The figures from *Table 7 - 2* shows that the chat does provide an easy way to communicate synchronously with other group members.

The same pattern also emerged when asked about the pop message facility. Seventy percent of the respondents choose 1 and thirty percent choose 2 when they were asked about the pop message facility as a synchronous communication tools. The statistic confirmed the ease of using the pop message facility as a synchronous communication tool (see *Table 7 - 2*).

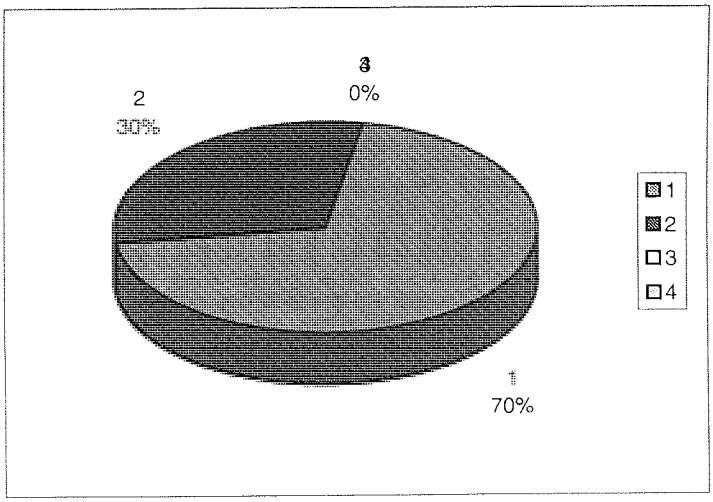

Figure 7 - 5 : *The Suitability of FlexSIG in Synchronous Process*

As a whole, the majority of the respondents agreed that the FlexSIG system is suitable for synchronous mode of processing (see *Table 7 - 2* and *Figure 7 - 5*). Seventy percent choose 1 and thirty percent choose 2.

The respondents suggested video-audio as an additional component to the synchronous communication facilities.

7.2.3 Flexibility of FlexSIG

With reference to the question of the flexibility of FlexSIG in supporting the synchronous and asynchronous mode of process, fifty percent choose 1, thirty percent choose 2, and twenty percent choose 3 (see *Figure 7 - 6*). The figure from *Table 7 - 3* shows that the flexibility offered by the system is sufficient in supporting both the synchronous and the asynchronous mode of work.

On the question of the choice of FlexSIG as a better choice over manual inspection process, seventy percent choose 1 and thirty percent choose 2 (see *Figure 7 - 7*). The number shows that the FlexSIG is a better choice (see *Table 7 - 3*).

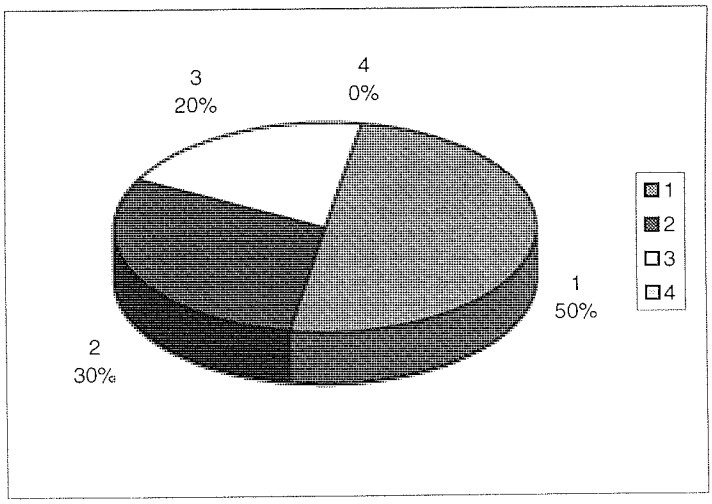

Figure 7 - 6 : *The Flexibility of FlexSIG*

The bulk of the respondents agreed that the FlexSIG system offered better choice than a system that offer only either the synchronous support or the asynchronous support (see *Table 7 - 3*). Eighty percent choose 1 and twenty percent choose 2.

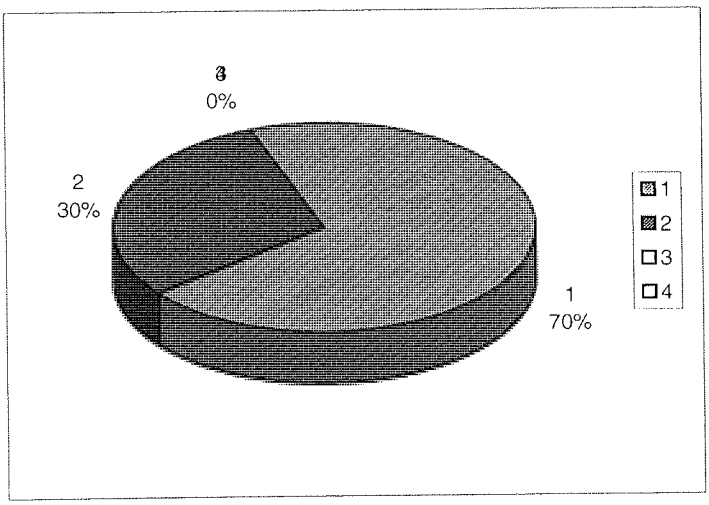

Figure 7 - 7 : *The Acceptability of FlexSIG over Manual Inspection Process*

The majority of the respondents also agreed that the FlexSIG system offers better flexibility by being implemented on the web (see *Table 7 - 3*). Ninety percent choose 1 and ten percent choose 2.

Questionnaire	Mean	Standard Error	Standard Deviation
Suitability of FlexSIG supporting Synch and Asynch	1.70	0.26	0.82
FlexSIG a Better Choice over Manual Inspection	1.30	0.15	0.48
FlexSIG a Better Choice over System with 1 Mode	1.20	0.13	0.42
Better Flexibility over the Web	1.10	0.1	0.32

Table 7 - 3 : *Mean for Flexibility of FlexSIG*

Among the suggestions that were given is that FlexSIG could strengthen the quality of software over manual inspection. There was also a suggestion to test it in a real-life situation to better measure of its benefits and facilities. There was also a concern from the respondents about the information flowing freely without any protection, for example, the source code, which is an asset for a software company, should be protected.

7.2.4 Relationship Between the Variables

Further study was conducted to analyse the relationship between certain variables. Similarly to phase one, crosstabulation and boxplots were utilised. Crosstabulation was used first to identify any relationship between two variables. Boxplot graph was put to use later to visualise the relationship.

There are three types of relationship that this research would like to look into. The first is the effect of the suitability of FlexSIG for an asynchronous process on the acceptability of FlexSIG. The second is the effect of the suitability of FlexSIG for a synchronous process on the acceptability of Flex-

SIG. And the third is the effect of the flexibility of FlexSIG on the acceptability of the prototype.

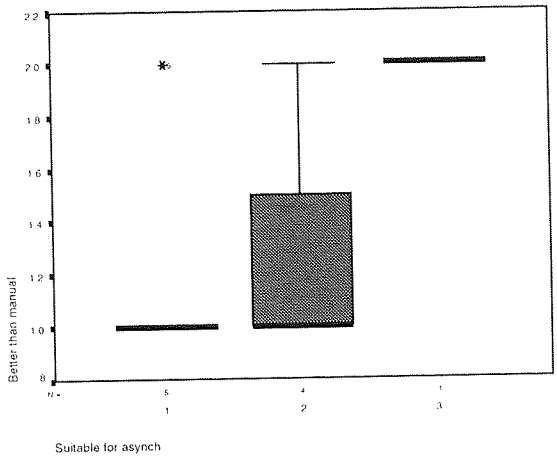

Figure 7 - 8 : *The Suitability of FlexSIG in Supporting Asynchronous Work versus the Acceptability of FlexSIG over Manual Inspection*

Whilst examining the relationship between the suitability of FlexSIG for asynchronous work and the acceptability of FlexSIG over the manual inspection process, *Figure 7 - 8* reveals that fifty percent of the sample that choose value 1 and 2 for the suitability for an asynchronous process is at 1.0 for the acceptability over manual inspection process. But the 25th percentile for the sample that choose 1 for the suitability for an asynchronous process is at 1.0 and for the sample that choose 2 is at 1.5. Thus, it can be said that the more suitable the prototype is for the asynchronous process in the view of the respondent, the more acceptable the prototype over the manual inspection process.

In the analysis of the suitability of FlexSIG for synchronous work versus the acceptability of FlexSIG over manual inspection process, the 50th percentile for the sample that choose 1 for the suitability for the synchronous process is at 1.0 and for the sample that choose 2 is at 2.0 (see *Figure 7 - 9*). The 25th percentile for the sample that choose 1 is also lower than the sample that choose 2, at 1.0 and 2.0 respectively. This means that the more suitable the

prototype is for the synchronous process in the view of the respondent, the more acceptable the prototype over the manual inspection process.

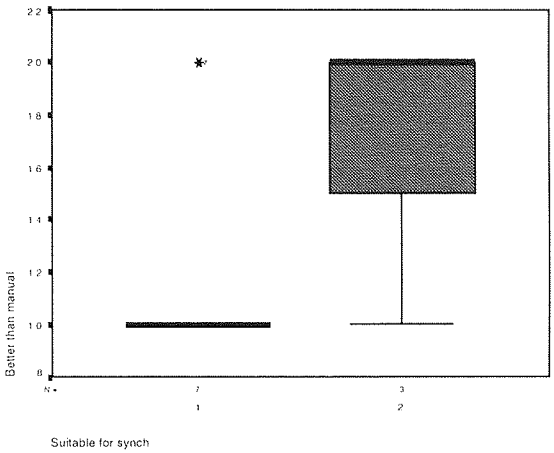

Figure 7 - 9 : *The Suitability of FlexSIG in Supporting Synchronous Work versus the Acceptability of FlexSIG over Manual Inspection*

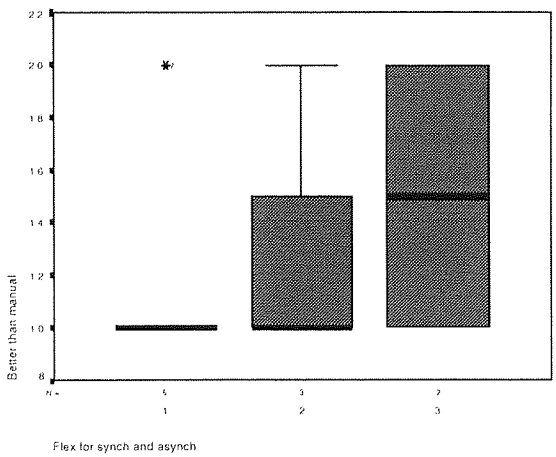

Figure 7 - 10 : *The Flexibility of FlexSIG versus the Acceptability of FlexSIG over Manual Inspection*

On the relationship of the flexibility of FlexSIG versus the acceptability of FlexSIG over the manual inspection process, referring to *Figure 7 - 10,* at 50th percentile, the sample that choose value 1 for the flexibility of the prototype is at 1.0 for the acceptability over manual inspection process. For the sample that choose 2, the acceptability level is at 1.0, and for the sample that

choose 3, the acceptability level is at 1.5. At the 25th percentile, the prototype acceptability level for the sample that choose flexibility value 1 is still 1.0, for the sample that choose 2 the level is 1.5, and for the sample that choose 3 the level is 2.0. Hence, a similar pattern occurred in this relationship as in the previous two. The more flexible the prototype in supporting both the synchronous and the asynchronous process in the view of the respondent, the more acceptable the prototype over the manual inspection process.

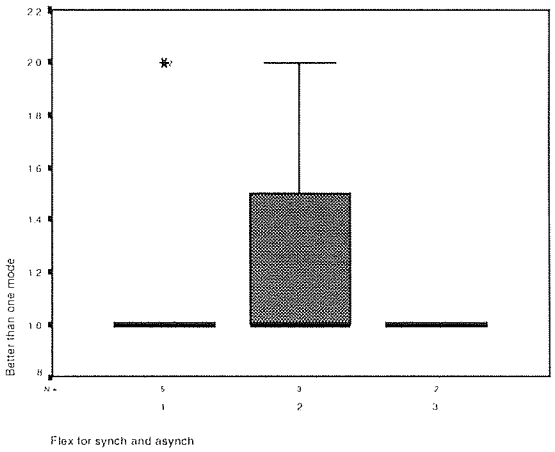

Figure 7 - 11 : *The Flexibility of FlexSIG versus the Acceptability of FlexSIG over System that Support One Process Mode*

From the observation of the dependency in *Figure 7 - 11*, the 50th percentile of the sample that choose value 1, 2, and 3 for the flexibility of FlexSIG is at level 1.0 for the acceptability of FlexSIG over system that support one mode of process only. The 25th percentile for the sample that choose 1 and 3 remained at level 1.0. For the sample that choose 2, the 25th percentile is at level 1.4. Thus, it can be concluded that there is a weak relationship between the variables. Regardless of what the respondent thought about the flexibility of the system, most of them considered that the system is better than other system that support only one mode of working process.

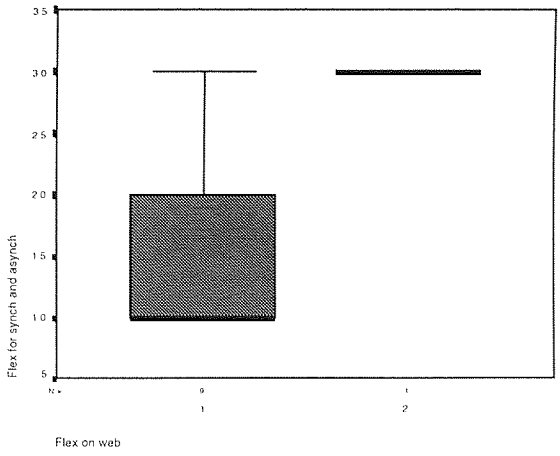

Figure 7 - 12 : *The Implementation on the Web versus the Flexibility of FlexSIG*

The boxplot graph in *Figure 7 - 12,* where the x-axis is the implementation of FlexSIG on the Web to provide better flexibility in terms of multi-platform support and the y-axis is the flexibility of the prototype, show that at the 50th percentile, when the x value is at 1, y is at level 1.0. But when the x value is at 2, the y level is at 3.0. At the 25th percentile, the same pattern occurred as when x is at 1, y is at level 2.0, and when x is at 2, y is at level 3.0. Hence, it can be concluded that if the prototype is perceived to provide better flexibility in terms of multi-platform support by implementing it on the web, then the prototype is accepted as being more flexible.

7.3 SUMMARY

To sum up this chapter, the analysis and the evaluation of the prototype constructed were presented. The prototype developed was evaluated by a group of post-graduate students in computer science. Data was gathered after the evaluation using a questionnaire. The questionnaire in phase three is aimed at measuring the acceptability of the prototype in a synchronous and asynchronous environment, and flexibility in both environments.

Among the methods employed are percentage, mean, standard deviation, and crosstabulation. The percentage, mean and standard deviation are used to show the tendency or the inclination of the sample. The crosstabulation is used to identify any relationship between two variables in the sample. The respondents agreed that the prototype is suitable for the synchronous inspection and also for asynchronous inspection. The respondents also agreed that the prototype is flexible enough to support both mode of inspection.

The next chapter will discuss the research finding and evaluation of this research.

Chapter 8

RESEARCH FINDING AND EVALUATION

8.0 INTRODUCTION

This chapter presents the data collection analysis, result and evaluation of this research.

- **Section 8.1** presents the research finding from the analysis.
- **Section 8.2** spells out the recommendations based on the findings.
- **Section 8.3** provides a critique on this research.

8.1 RESEARCH FINDINGS

This section reveals the findings made in this research. The findings are categorised into three area: namely, the first phase finding, the second phase, and the third phase finding.

8.1.1 Phase One

The first phase concerns with the development of the FlexSIG model. In this phase, two major activities were conducted, namely, the development of the

FlexSIG model and the questionnaire concerning software inspection practice to a group of software engineers. The questionnaire served as an input to the development of the model along with the material from the literature review.

Questionnaire

The phase one questionnaire serves as an additional input to the development of the model of FlexSIG. This section discusses how the findings from the questionnaire help to construct the model.

The reasons why the respondents have had difficulties in using the inspection and review process were difficulties in categorising the errors and defects, no training given on how to conduct the process, not being familiar with the inspection process itself, the inspection process is time consuming, the process is a tedious job, there are quite a number of documents to handle in the process, and also difficulty in accessing the design document during the inspection process. The findings also bring to light the difficulty in using the formal code inspection process because of inadequate experience and exposure to the process and the tedious nature of the process.

The findings showed that it is difficult to get hold of the design and requirement document during the inspection or review process. The data also showed that it is not easy to conduct a meeting in a formal inspection process. The questionnaire reveals that both methods of communication, the synchronous and asynchronous mode, were used in software development projects. The methods include face-to-face meeting, telephone, e-mail, memos, electronic chat, and also informal conversation and informal meeting.

Among the features that the user would like to have in a CASE tools are direct access to specification documents such as process flow and relationships, and support for specification, design, and code inspection. The importance of a

graphical user interface, good report generation features and also support for testing, such as an automatic test generator were put forward. The known facts from the questionnaire showed that a distributed CASE tool for inspection process is a better choice than a stand-alone tool.

The indication from the finding was very clearly that the respondents are positive towards using a software inspection process in the future.

From the analysis, it was found that exposure to training in software quality didn't seem to contribute more than no exposure to software quality training in the likelihood of using inspection in the future. This is probably because in training, the full benefit of the inspection process could not be visualised and thus does not effect the likelihood of using inspection in the future. On the other hand, the experience in using the inspection process contributes positively towards the likelihood of using the inspection process in the future. The benefit of using inspection in real world situations probably influences the likelihood of using inspection in the future in a positive way.

The analysis also indicates that experience of having difficulties in using any inspection or review process in general does not discourage the likelihood of using inspection in the future. However, experiencing difficulty in using the formal code inspection does contribute negatively towards the usage of an inspection process in the future. Explanation for the former is that it includes other less formal inspection methods such as walkthrough, and the respondent most probably will continue to use the method as it involves less preparation. The explanation for the latter is that because of the rigorous and time consuming nature of formal code inspection, any difficulties will bring a negative effect on the future use of the process.

The figures from the analysis show the importance of providing easy access to the documents and its influenced on the acceptability of the inspection

process. The crosstabulation analysis showed that in order to improve the acceptability of the inspection process, the process of conducting a meeting have to be made less difficult.

As a whole, the questionnaire was successful in extracting the information needed from the respondents. Nevertheless, the contents of the questionnaire could be improved further if a pilot study were conducted in order to check any bias in the design of the questions.

FlexSIG Model

The model provides flexibility (Cockburn & Jones, 1995) at two stages of the process, during the initial stage where the information and material with regard to the process are distributed and during the group inspection phase. In the other phases of the model, namely the initiation, individual inspection, consolidation, and follow-up, work is done on an individual basis. It is also important to provide a flexible meeting environment for the inspection process in order to improve the ease of conducting a meeting, as showed by the questionnaire. The model in fact, allows both distributed synchronous and distributed asynchronous inspection to take place.

The FlexSIG model approach to minimising constraints (Cockburn & Jones, 1995) is by providing a flexible working mode as explained in Chapter Five. The capability to be implemented on different platforms also minimises constraints. The external integration strategy (Cockburn & Jones, 1995) aims to solve the problem of differing applications or communications environments. The provision of support for heterogeneous environments and the ability to easily integrate new external components attends to this problem.

The FlexSIG model supports the implementation of inspection process on different platforms across the internet with a consistent graphical user in-

terface. The issue here is to not force changes on the user. Fewer changes for the user result in models being more acceptable to the user (Cockburn & Jones, 1995). The questionnaire's findings also point towards the requirement of a consistent graphical user interface.

The findings of the questionnaire point to the need to support not only the synchronous and asynchronous mode of communication, but also the capability to support informal conversation and informal meetings. The FlexSIG system provides a shared state (Shim *et al.*, 1997) in the form of the log file. The changes to the log file can be viewed remotely by other group members and act as an asynchronous communication object. The model also provides awareness in terms of information sharing, knowledge of group and individual activity, and co-ordination (Dourish & Bellotti, 1992), which is central to a successful collaboration. The information sharing can be achieved through the log file, the inspected documents, and the rest of the supporting material. The knowledge of group and individual activity can be tracked using the communication suite, namely, the synchronous communication and the asynchronous communication. The co-ordination can be achieved using the same communication components and the remote browsing capability.

The availability of the supporting document on-line was in response to the findings in the questionnaire where it was suggested that the tools support direct access to supporting documents. The facility was provided also as a solution to another point raised by the findings which mentioned that there are quite a number of documents to handle in the process. Among the other findings from the questionnaire which were also supported in the model is support for specification, design, and code inspection processes, report generation features, and the facility to categorise the errors and defects.

To summarise, the model is successful in providing a flexible model of the inspection process by offering alternatives in the process flow. However,

the model of the process effect on other aspects of software inspection is not measured. For example, the effect on software quality and the length of the meeting is not measured.

8.1.2 Phase Two

This phase saw the development of the FlexSIG prototype. The prototype was constructed based on the model from phase one. The prototype was designed to be flexible and to be able to accommodate several other models of the software inspection process, specifically the Fagan, Gilb & Graham, and Humphrey models.

By implementing the prototype on the web, the problem of critical mass (Grudin, 1994) in FlexSIG is attended to. The FlexSIG prototype also allows flexibility in terms of different working modes and also in terms of different models supported, the chances of disrupting the existing inspection process is less and thus minimising the disruption of social process (Grudin, 1994). The utilisation of java applet in the prototype allows unobtrusive accessibility (Grudin, 1994) to the components that support group process.

The prototype supports the FlexSIG model and other inspection models, implementing the strategy of minimising constraints (Cockburn & Jones, 1995) is implemented. The flexibility (Cockburn & Jones, 1995) of supporting multiple models even allows users the opportunity to develop their own protocols governing inspection work as they see fit. The implementation of the prototype using the web and java also results in easier external integration (Cockburn & Jones, 1995). The multiplicity of platforms that support the web allow the prototype to run on different platforms in heterogeneous environments, and hence achieve improved external integration.

As mentioned in Chapter Two, CASE tools for software inspection can be classified as support tools. By supporting groupwork in the inspection process, FlexSIG prototype was able to provide the support tools proposed up by Vessey & Sravanapudi (1995) and Beaudouin-Lafon (1990).

The prototype implementation is successful in providing the tools to implement the CSCW design issue and support the model. There is still a lack of customisation components to make it more flexible. The prototype is still not flexible enough to support inspection models, other than the one mentioned.

8.1.3 Phase Three

In phase three, the user evaluation of the prototype was conducted. The questionnaire's findings are discussed in this section. The questionnaire was designed to establish whether the prototype was adequate for an asynchronous process, a synchronous process, and also whether the prototype is flexible enough to handle both working modes.

In order to support the asynchronous inspection process, the prototype provided the briefing component as a replacement for the kick-off meeting. The findings from the questionnaire in phase three shows that the briefing concept is sufficient in replacing the kick-off meeting. One of the respondents cautioned that although the briefing concept may replace the kick-off meeting, the kick-off meeting is still important for face-to-face meeting.

Based on the data in phase one, it is important to provide easy accessibility to the supporting documents. The prototype provided the documents in the form of HTML and the respondents agreed that the HTML is a better option than a paper version for distributing the specification document. The respondents stated that HTML is a better option because of speedier delivery and

timeliness. The prototype also provided another form of support document, namely the on-line help component. On the question of the sufficiency of the help concept for the inspection process, the numbers show that the help concept is accepted as satisfactory for the software inspection process.

As a whole, the respondents agreed that the FlexSIG system is suitable for asynchronous mode of process. Among the suggestion that was given is that the FlexSIG should be implemented over a more secure channel. The findings also showed that the information collected in the log entry is sufficient.

Concerning the complementary aspect of the briefing concept in relation to the kick-off meeting in a synchronous process, the figures show that the briefing concept is a good complement to the kick-off meeting. This validated the aim of the prototype that in a synchronous process, the briefing will behave as complementary component.

To support the synchronous aspect of the inspection process, the Flex-SIG prototype provided several components. The findings from the questionnaire revealed that the remote viewer, chat, and pop-up message do provide an easy way to communicate synchronously with other group members.

With reference to the specific question of the acceptability of the prototype for synchronous software inspection process, the finding was that it is positive. Overall, from the set of questions regarding the synchronous aspect of the inspection process, the respondents agreed that the FlexSIG system is suitable for synchronous mode of process. The respondents suggested video-audio as an additional component to the synchronous communication facilities.

The prototype was developed to support both the synchronous and asynchronous mode of work. It also supported the heterogeneous aspect of the users computer platform and was tested on Unix, Macintosh and PC environments. In regard to the question of the flexibility of FlexSIG in supporting both

the synchronous and asynchronous mode of process, the finding also shows that the flexibility offered by the system is sufficient in supporting both the synchronous and the asynchronous modes of working. On the question of the choice of FlexSIG as a better option over the manual inspection process, the number shows that the FlexSIG is a better choice. The respondents also agreed that the FlexSIG system offered better choice than a system that offered only either synchronous support or asynchronous support. The FlexSIG system implementation on the web as a way to provide flexibility for multi-platform client was also viewed positively by the respondents.

The analysis revealed that the more suitable the prototype is for the asynchronous process in the view of the respondents, the more acceptable the prototype over the manual inspection process. In the analysis, it is also shown that the more suitable the prototype is for the synchronous process in the view of the respondents, the more acceptable the prototype over the manual inspection process. A similar pattern occurred in another analysis where the more flexible the prototype in supporting both the synchronous and the asynchronous process in the view of the respondent, the more acceptable the prototype over the manual inspection process. This is not surprising considering that one of the key issues of the acceptability of a CSCW system mentioned in Chapter Three was the flexibility.

The findings from the analysis of the relationship between the flexibility of the system and the acceptability of the prototype over other system that support only one mode of the working process reveal only a weak relationship. Regardless of what the respondent thought about the flexibility of the system, most of them rated the system as better than other systems that support only one mode of working process. The analysis also discloses that if the prototype is perceived to provides better flexibility in terms of multi-platform support by implementing it on the web, then the prototype is accepted as being more flexi-

ble. Therefore, this finding confirmed the assumption that providing multi-platform support is one of the important factor in determining flexibility.

The evaluation results produced was as expected. Nevertheless, the problem in this phase is the same as in phase one. The contents of the questionnaire could be improved further if a pilot study is conducted in order to check any bias in the design of the question.

8.1.4 Original Objectives Revisited

Looking back at the objectives of the research, the following have been achieved:

- The requirements of a flexible software inspection model and prototype have been identified. The requirements are extracted from the literature and the questionnaire given to a group of software engineers (see Chapter 4, Chapter 5, Appendix H, and Appendix I)
- The flexible model of the software inspection process model has been successfully developed. The flexibility is supported by allowing options to the user during the software inspection process (see Chapter 5)
- The design consideration was successfully integrated into the development of the working software inspection prototype (see Chapter 6 and Appendix B)
- The prototype developed is based on the functional model of FlexSIG. The components implemented in the prototype supports the FlexSIG model. The prototype developed also supports other models and can be implemented on multiple platforms. The choices provided by the prototype allow different software inspection models to be followed. The implementation on

the WWW and using java provided the heterogeneous support (see Chapter 6 and Appendix B)

- The evaluation of the acceptability of the prototype was conducted. The analysis of the questionnaire confirmed the acceptability of the prototype and the success of the model (see Chapter 7 and Appendix J)

8.2 RECOMMENDATIONS

The recommendations presented in this section are based on the user evaluations in Chapter Seven and findings furnished in the previous section. The recommendations are as follows:

- A model of a working process should be flexible and not restrictive. The model should be able to provide alternatives and options along the process flow to the users. The choices allow the model to be flexible within a set framework. From the FlexSIG model and the questionnaire, it is established that a model that allows the user to make choices is more acceptable.

- A groupware system should support multiple models rather than just follow one model. The system that supports more than one model stands a better chance of being accepted. It is impossible for any one system to support all models in existence for that particular process, but if the system can at least support the major models, then the acceptance is much higher. The FlexSIG prototypes showed that the ability to support more than one model is an important factor.

- In the meeting phase of the software inspection process, the team should be provided with other alternatives other than the formal face-to-face meeting structure. Among the alternatives is an informal meeting structure and a

distributed meeting structure. The difficulty in conducting a meeting was pointed out as one of the factors faced by the user. The flexibility provided in the meeting structure will enhanced the acceptability of the process. The toolset that supports an inspection process must support both the synchronous and the asynchronous mode of communication to facilitate the discussion.

- In relation to the point above, the software inspection process as a whole should support both the distributed synchronous mode and the distributed asynchronous mode of the software inspection process. The support for the distributed aspect should be available throughout the inspection process, not only the groupwork aspect.

- An informal or an indirect communication structure should be supported. This is because, the findings showed that the synchronous and asynchronous communication is not the only one that take place, but also other form of informal or indirect communication. The implementation of shared state and awareness capability served this purpose.

- Variation in software inspection should be allowed. This is because, this research revealed that the more flexible the process, the more acceptable it is to the user. The variation is also consistent with the points put forward by Freedman & Weinberg (1990) who stated that different external requirements, different internal organisations, and continuity with past practices influenced the way the inspection is set-up. The variations also make it less likely for the problem of disruption of social process to occur.

8.3 CRITIQUE

This section presents the evaluation of this research, which includes the literature survey, the formulation of the research problem, the FlexSIG model, the prototype, the user evaluation, and the overall critique of this thesis.

8.3.1 Literature Survey

An extensive survey of the literature was successfully completed. A large number of references, over one hundred and fifty, were found in refereed journals, theses, books, and conference publications. The type of literature found was spread evenly, covering the general issue of software engineering, narrowing down to software quality, before focusing to software inspection. The literature survey also covered the design issues of CSCW and the enabling technologies, namely, the world wide web, java, the object oriented paradigm, and hypertext.

The survey found most of the related literature, especially in the area of software inspection. This observation is based on the index of thesis, proceedings of the latest conference related to the topic, and cross-reference from literature on hand. Overall, the literature survey produce a good coverage of published work.

8.3.2 Formulation of the Research Problem

The literature survey of the software inspection process and the design issues of CSCW pointed to a gap. Although there are quite a number of papers on the software inspection process, the topics discussed were either on the drawbacks

of software inspection process or on groupware support for inspection in one quadrant of time-space matrix only. Thus, there is a gap in the literature where the design issues of CSCW in software inspection have not been addressed.

The design strategy from the literature provided some of the solutions to the design issues of CSCW in software inspection. The other method used was a questionnaire. The research method that was decided on was to develop a model, followed by a prototype, and perform user evaluation of the prototype. A one-shot case study strategy was used for the user evaluation. The decision on the research design strategy was based on the literature. The one-shot case study is not the best experimental method, but is chosen because of limited availability of time, participants, and relatively small number.

The design strategy was generally successful because it was based on the design issue of general CSCW systems which was extensively researched. However, the strategy could have been improved if an observation on a real-life inspection was conducted. The observation could have confirmed the strategy devised or provide the opportunity to change it.

8.3.3 The Model

The model was constructed based on the literature surveyed and the questionnaire. Based on the strengths and the limitations of existing models, an extended model was proposed. The model also takes into consideration of the questionnaire's finding during the development. Addressing the design issues of CSCW for software inspection was the main goals of the model. The model addresses the issues of critical mass, disruption of social processes, unobtrusive accessibility, integration of different tools, and lack of flexibility in the software inspection process.

The design of the model could be improved further if further analysis and observation were conducted on a real-life inspection process. The observation could be used to verify the decision taken based on existing models and the questionnaire. The time constraints did not permit this extension. Overall, the model provides a good basis for supporting flexibility in the process. The choices in the process flow and the components offered enough flexibility.

8.3.4 The Prototype

The construction of the prototype was aimed at implementing the model and the goals of the design issue of CSCW for software inspection process. The existing tools for software inspection were analysed and their limitations and strengths presented. Based on this information, an extended tool was proposed.

The prototype was constructed using the world wide web, java, the object oriented paradigm, and hypertext. Although this technology does provide flexibility, the system is slow and depends too much on the internet infrastructure. Utilising the internet infrastructure may provide security problem to the system.

The flexibility provided by the tools met the aim of the design, but it can be improved further. A customisation facility could be provided in order to add, delete, and redefine roles in the inspection process. This facility should include the role definition and access rights. This is because the job specification of the different roles is changed depending on which model is being used.

Nevertheless, the tool is successful in providing the flexibility missing from existing software inspection tools.

8.3.5 User Evaluation

The user evaluation on the prototype was conducted in order to measure the acceptability of the prototype. The evaluation showed that the prototype met the aims outlined earlier. The result of the questionnaire's analysis is valid within the standard error. The user evaluation could have involved more people. The background of the participant also should include more people directly involved in software quality. Because of time and personnel constraints, a one-shot experiment was conducted. The experiment also measured the acceptability issue and not the effectiveness of the tools. The experiment could have been done better with a control group with pre and post-testing. This could resulted in a better statistical conclusion.

8.3.6 Overall Critique

Despite the findings and the recommendations discussed previously, there are problems and limitations inherent in this research. Some of the problems and limitations arose from the need to make the research more manageable and some are caused by limited availability of resources.

- In the development of the FlexSIG model, the basis for the development of the flexible model of the software inspection process is review of literature of existing practices. Additional data was also gathered from a group of software engineers to assist the construction of the model. In an ideal situation, the development of the model should have more data in the form of an observation of a real working inspection process on top of review of existing practices and the questionnaire.

- The implementation of the prototype only concentrated on a smaller aspect of software inspection, namely, the code inspection. The model developed

was meant to be used as a general software inspection model that supported requirement and design specification inspection as well.

- The scope of the enabling technology usage is also limited to text only. The inspected document cannot be of the graphical type. No visual or audio aspect was used in this research. There is also a lack of intelligence and automation in detecting common errors and defects during the inspections. The prototype relied on a file system created specifically for its usage, there is a lack of common database integration in the implementation.

- A more comprehensive programme of experiment, for example, a pre-test and post-test control group experiment should be conducted. This allows more confidence in the statistical conclusion, for example, the analysis of variance can be used. In the evaluation of the prototype, only a one-shot case study design is employed. The evaluation is conducted using a small group of computer scientist in an educational environment. Ideally, the testing should be conducted in a real-life working environment.

- The testing and the evaluation of the model could be conducted explicitly. Currently, the conclusion on the model is implied by the evaluation of the prototype, which is based on the model. A more extensive evaluation, not only on the prototype, but also on the model of the process is desired.

- The evaluation in this research is only on the acceptability aspect of the process. No study has been conducted on the effectiveness of the flexible inspection process on the quality of the software produced. Further testing is needed in order to determined whether the flexible approach affects the quality.

- The implementation of the prototype on the web using java does have it strengths and its weaknesses. One of the problems is because of the interpreted nature of java and the reliance on the world wide web infrastructure.

This combination made the system slow in terms of loading time and response to the user command.

- The different version of java releases also caused an inconsistency in the feature supported. The capability of the web browsers to support java applet also caused a problem. The majority of web browsers do support java applets, but the version of java supported may be different.
- Last but not least, there is a lack of security features in the implementation. Although there is some rudimentary access control feature built into the system, there exist a number of other security questions. Ideally, the FlexSIG architecture would be constructed in an Intranet environment in which the FlexSIG is implemented behind a firewall for protection against unauthorised access from the Internet, but in the implementation, no such firewall exist. The free-flowing nature of the information across the internet also created another security issue. The information needs to be protected from unauthorised access.

8.4 SUMMARY

To sum up this chapter, the finding, which is based on the analysis and the evaluations were compared against the original aim. It can be concluded that this research has met the aims outlined in the fourth chapter. The result based on the findings are that, this research has:

- developed a flexible software inspection process model
- implemented a prototype based on the model
- evaluated the prototype and found it acceptable and flexible.

The end result also showed that the system's flexibility met the definition of the flexibility outlined in Chapter Four. The prototype was confirmed to be flexible in terms of the following:

- Allowed the software inspection process to use all four quadrants of the space-time matrix
- Available across the internet on different computer platform running different operating system
- Allowed the user to choose the software inspection process model that fits their requirement and style.

The recommendations by this research was also put forward along with the critique, which evaluated every aspect of this research.

The next chapter will discuss some future development that can be carried forward from this research.

Chapter 9

CONCLUSION

9.0 INTRODUCTION

This chapter presents the overall conclusion of this thesis. The information presented here is the culmination of the work described in the previous chapters.

- **Section 9.1** covers the summary of the previous chapters and presents the general conclusion.
- **Section 9.2** presents the future development of this research topic.

9.1 RESEARCH CONCLUSIONS

This section re-examines the information presented in all the previous chapters and presents the general conclusion of this research.

9.1.1 Literature Background

This research covered two main areas: software quality and CSCW. This research started with an introduction and definition of software engineering, including software quality. Definition of CSCW from Wilson's point of view was

given. The idea that software development is a group activity was also introduced.

The topic of software engineering was expanded further with issues on software engineering methodology, process models, software engineering management and CASE tools being presented. One of the primary goals of software engineering is to improve the quality of software products (Fairley, 1985). The discussion on software quality was concentrated on the description of the techniques and methods to achieve high software quality. A proper management system is needed in order to achieved quality software. Software quality also depends on Capability Maturity Model, Verification & Validation, software testing, and evaluation techniques. One aspect that all these techniques have in common is that they make use of the software inspection process as part of each technique.

With this background, the topic was focused further on the issue of software inspection, including its costs and benefits. Software inspections aim to analyse a product or document to detect defects (Fagan, 1976). Several major variants of the software inspection process model were presented including Fagan's (1976, 1986), Gilb and Graham's (1993), and Humphrey's (1989). Fagan's, Gilb and Graham's, and Humphrey's model of the software inspection process is a formal manual task. Two of the earliest CASE tools that support software inspection process were presented, ICICLE (Brothers *et al.*, 1990) and InspeQ (Knight & Myers, 1991, 1993).

Further work on the classification of CSCW and early developments were reviewed. The classification of CSCW by Ellis *et al.* (1991), which includes the idea of classifying groupware according to a time-space taxonomy was furnished. The CSCW reviews were narrowed to the problem in design, design strategies, and issue of flexibility in the acceptability and design issues

of CSCW (Cockburn & Jones, 1995; Roger, 1994; Cockburn & Thimbleby, 1991; Grudin, 1994; Glance *et al.,* 1996; Robinson, 1993; Haake and Wilson, 1992).

Software development is a group activity (Jones, 1986). Thus, there is a need for software tools that support group activity in a software process. Groupware support for the software development process was discussed with special attention given to the topic of groupware support for software inspection. Among the groupware tools that support software inspection are: CSRS (Johnson & Tjahjono, 1993), CSI (Mashayekhi *et al.*, 1993), Scrutiny (Gintel *et al.,* 1993), CAIS (Mashayekhi *et al.*, 1994), and AISA (Stein *et al.*, 1997). CSRS supports distributed asynchronous inspection, CSI only supports distributed synchronous inspection, CAIS, like CSRS, supports distributed asynchronous inspection, AISA supports distributed asynchronous inspection on the Web, and finally, Scrutiny supports distributed synchronous inspection.

To summarise, the goal of software engineering is to improve the quality of software product. One of the techniques to improve quality is software inspection. The software inspection process is an example of groupwork and it is affected by the acceptability problem of all groupware. The design issues for CSCW in software inspection were presented. Current inspection models and tools do not address these issues. This research addresses to the design issue, proposing a flexible software inspection model and implementing a prototype system.

9.1.2 The Aim

Taking into consideration the background presented in the previous section, the objective of this research is to design and build a groupware system which will allow members of a distributed group more flexibility in performing software inspection. The groupware system aims to provide a system that will im-

prove acceptability of groupware by providing a software inspection tool that is flexible and adaptable.

Specifically, the development of the Flexible Software Inspection Groupware (FlexSIG) system includes:

- The development of a flexible software inspection process model
- An implementation of a prototype based on the model of the system
- The evaluation of the prototype in order to measure the acceptability of the system.

The flexibility that this research is looking for in the model and the prototype, as defined in Chapter Four, is as follows:

- The ability to allow the software inspection process to use all four quadrants of the Johansen (1988) space-time matrix or the four categories of the time and space taxonomy proposed by Ellis *et al.* (1991).
- The ability to use the tool across the internet on different computer platforms running different operating systems.
- The ability to let the user choose the software inspection process model that fits their requirement and style.

This research was designed to be experimental and used the one-shot case study design. There are three phases in this research, each phase corresponds to a specific aim of the FlexSIG development outlined above. The first phase is to develop a model for the flexible software inspection process, the second phase is to develop the flexible software inspection groupware prototype, and the third phase is to evaluate the prototype. Simple statistical methods were used in the analysis, which includes, mean, standard error, standard deviation, crosstabulations, and boxplots.

9.1.3 FlexSIG Model

The development of the FlexSIG model is an extension and a combination of the existing models of inspection proposed by Fagan, Gilb & Graham, and Humphrey. The model aims to support one aspect of the flexibility mentioned in the previous section, that is, the ability to allow the software inspection process to use all four quadrant of the Johansen (1988) space-time matrix or the four categories of the time and space taxonomy proposed by Ellis *et al.* (1991).

The construction of the model started with the conceptual framework of FlexSIG which comprised the assumptions, considerations, and high-level concepts. The conceptual framework serves as a basis for the functional model of FlexSIG. The functional model includes the description on the components and services available, the data model, and the process model. The process model of FlexSIG comprises an initiation phase, followed by either a kick-off meeting phase or a briefing phase. The process flow merges again for the individual inspection phase. For the group inspection phase, there is a choice between synchronous or asynchronous group inspection. The consolidation phase merged the process flow again, and is followed by the follow-up phase. In this model, there are three roles, namely, the moderator, the author, and several inspectors.

9.1.4 FlexSIG Prototype

The FlexSIG model is the basis for the prototype. The development of the prototype is targeted to solve the other two issues in flexibility, namely: the ability to use the tool developed across the internet on top of different computer platform running different operating system, and the ability to let the user choose the software inspection process model that fits their requirement and style.

The functional architecture of a web-based groupware system, based on the world wide web and the client-server architecture, was presented. The FlexSIG architecture consists of several viewpoints of the system. The architecture of the prototype is described from the system, database, user interface and security architecture perspective. The components of FlexSIG prototype includes the briefing, specification, document inspection, comment log, communication, help, metrics and reports, access control, and system configuration components.

Among the technologies that were used in order to achieve the aim are: the world wide web, java, the object oriented paradigm, and hypertext.

9.1.5 Research Findings and Evaluations

From the analysis and user evaluation, it can be concluded that this research has met the aim outlined in the fourth chapter. The findings, based on the analysis and the evaluations, were compared against the original aim. It can be determined that this research has developed a flexible software inspection process model, implemented a prototype based on the model, evaluated the prototype and found it acceptable and flexible. The end result also showed that the system's flexibility met the definition of the flexibility outlined in Chapter Four. The recommendations by this research were also put forward along with the critique of this research.

9.2 FUTURE DEVELOPMENT

There are a number of areas that can be expanded further for future development in this area. The list is provided below:

- The system can be expanded further to include the following facilities. The system still lacks any intelligent features. There is no automatic error detection for trivial defects or errors. Artificial intelligence could be employed to detect certain type of defects and take some burden away from the inspectors. A knowledge base could also be utilised to detect any trends by certain authors in future inspections.

- There is also no integration with any standard database management system. Currently, the system is utilising it's own file system. The integration with a standard database management system allows easier access and timeliness in accessing documents and the distribution of reports.

- The communication aspects of FlexSIG is still based on text. The system should also be expanded to support graphical documents so that the requirement and design inspection can be conducted.

- The inclusion of audio and video in the communication suite could provide more alternatives to the user and improve the flexibility further. With the inclusion of audio and video, other techniques of group awareness can be incorporated.

- The prototype can be improved further to include more flexibility. A role definition component is one example. This component should allow a new role to be added, or an existing role be removed or redefined. The role definition component should include job specification and access right control.

- The model and the prototype can be expanded further to include more security features. As it is, the only security feature of the system is a simple access control relying on username and password, and using user's role to control access to information. Currently, the information stored in the database is not protected, nor is the information that flows across the network. The conversation between users using the communication suite is

also open to attack. The model also did not take the security issue as one of the important criteria.

- A more comprehensive experiment can be conducted on the acceptability of the prototype to provide better statistical confidence. A pre-test and post-test control group experiment in which there are two group of inspection teams is one example. In this type of experiment, the initial data is collected before the 'treatment' is conducted. The experimental group then uses the FlexSIG system and the control group might use a manual inspection process. Data again is collected after the 'treatment'. Data collected before and after the 'treatment' can be compared in order to conclude the experiment. The data from the experiment group can be compared to the control group to check the validity of the result.

- A complete experiment on the model and the system can be conducted. Further testing of the model and the prototype in a real-life working environment over a period of time is needed in order to examine the acceptability and to improve the model and the system. This experimented structure could be the same but is more thorough than the previous point. The experiment begins with the planning followed by every phase of the inspection process in the model. There should be different group representing different working mode. Each group will use different aspect of the model, namely the face-to-face, distributed synchronous, and distributed asynchronous inspection. The results from each group will determined the acceptability of the model in that particular working mode.

- Further experiments also need to be conducted in order to measure the effectiveness of the flexible inspection process on the quality of the software produced. Again, the same structure as in the previous two points can be used. Preferably, two groups are used. One group will use the FlexSIG model of inspection, the second group will use a manual inspection model.

The effectiveness can be measured by the number of defects found in the process. The data on the quality aspect collected can be compared with the data from the manual process or from the usage of other type of inspection tools.

- The work can also be expanded further into other aspects of software engineering that require groupwork support. For example, the design technique of this research can equally be applied to the requirement and design document preparation which require groupwork support.

9.3 SUMMARY

To conclude, this research has demonstrated that a flexible model and groupware system to support software inspection process is more acceptable to the user. A flexible model has been developed. The model supports working modes in all four quadrants of the space-time matrix. A prototype was also developed and supported multiple software inspection models and can be used in a heterogeneous environment.

As a whole, this research has extended current technology of software inspection groupware by providing a flexible software inspection structure resulting from using the system. The main contributions of this research are as follows:

- A flexible software inspection groupware system that is not limited to any one of the four categories of the time and space taxonomy.
- A flexible software inspection groupware system that is able to accommodate different software inspection process models.

- A flexible software inspection groupware system that is not limited to any computer platform.
- A flexible software inspection structure based on the flexible software inspection groupware.
- A model of a flexible software inspection groupware.

The results and the findings obtained have indicated that the objectives outlined have all been met.

GLOSSARY

Author : A person or team who has written something (product) using a set of source documents, in accordance with a set of rules.

Asynchronous : Occurring at different times.

CSCW : Computer Supported Cooperative Work - A term which combines the understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services, and techniques.

Checklist : A specialised set of questions designed to help inspectors find more defects.

Defect : An error made in writing a document or code which violates a rule.

Document : A written set of information, which can be the subject for inspection.

Entry Criteria : The set of generic and specific conditions for permitting a process to go forward with the defined task.

Exit Criteria : The set of generic and specific conditions for permitting a process to be officially completed.

Experimental Research : A type of research where the researchers look at the effects of at least one independent variable on one or more dependent variables.

Group Activity Support Systems : Provide computer support for specific group process or activities.

Groupware : A generic term for specialised computer aids that are designed for the use of collaborative work.

Hypertext : A type of database which enable information to be held in chunks, and for each chunk to be link to other chunks.

Inspector : A person who examines a set of related documents with the primary objective of finding potential defects as defined by written rules and checklists. All inspectors are part of the larger process of Inspection.

Moderator : The person who leads the inspection process.

Object-Oriented : Object-oriented development is characterised by object oriented analysis, object oriented design, and implementation in an object oriented language which provides facilities for encapsulation and inheritance.

Query : An item logged during the inspection process, which requires explanation from the author.

Software Engineering : The technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates.

Software Inspection : An evaluation technique to find defects and problems in a product.

Software Quality : Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

Suggestion : A proposal recorded during the inspection process regarding a change to any software engineering process.

Synchronous : Occurring at the same time.

Treatment : A manipulation on the independent variable in an experimental research.

Workgroup Computing : Activities undertaken on a network using software application programmes designed to support the members of a group.

World Wide Web : A distributed heterogeneous collaborative multimedia information system.

REFERENCES

Abbot, Joe (1986), *Software Testing Technique,* Manchester: NCC Publications, The National Computing Centre Limited.

Achmatowitz, Richard (1994), "Object Groups for Groupware Applications: Application Requirements and Design Issues", *Technical Report No. 685,* London: Dept. Of Computer Science, Queen Mary and Westfield College, September.

Ackerman, A.F. (1984), "Software Inspections and the Industrial Production of Software", in Hause, H.L. (ed.), *Software Validation,* Amsterdam: Elsevier Science Publishers.

Arthur, Lowell Jay (1993), *Improving Software Quality: An Insider's Guide to TQM,* New York: John Wiley & Sons, Inc..

Arthur, Lowell Jay (1985), *Measuring Programmer Productivity and Software Quality,* New York: John Wiley & sons, Inc..

Babich, Wayne A. (1986), *Software Configuration Management: Coordination for Team Productivity*, Reading, Massachusetts: Addison-Wesley Publishing Company.

Baker Jr., Richard A. (1997), "Code Reviews Enhance Software Quality", *Proceeding of the 19th International Conference on Software Engineering,* New York: ACM Press, pp. 570-571.

Bannon, L. & Schmidt, K (1991), "CSCW: Four characters in search of a context", in Bowers, J.M. & Benford, S.D. (eds.), *Studies in Computer Supported Cooperative Work*, Amsterdam: North Holland, pp. 3-16.

Barghouti, Naser S. (1992), "Supporting Cooperation in MARVEL Process-Centered SDE", *Software Engineering Notes,* New York: ACM SigSoft 92, December, 17(2), pp. 21-31.

Bauer, Fritz L. (1976), "Programming as an Evolutionary Process", *Proceedings of the 2nd International Conference on Software Engineering*, January, New York: IEEE Computer Society, pp. 222-234.

Beaudouin-Lafon, Michel (1990), "Collaborative Development of Software", in Gibbs, S. & Verrijn-Stuart, A.A. (eds.), *Multi-User Interfaces and Applications,* Amsterdam: Elsevier Science Publishers B.V., pp. 103-114.

Bentley, R. *et al.* (1995), "Supporting Collaborative Information Sharing with the World Wide Web: The BSCW Shared Workspace System", *Proceeding of the 4th International World Wide Web Conference,* Boston, December.

Berners-Lee, Tim, *et al.* (1994), "The World Wide Web", *Communications of the ACM,* New York: ACM Press, August, 37(8), pp. 76-82.

Berners-Lee, Tim, *et al.* (1992), "World Wide Web: The Information Universe", *Electronic Networking: Research, Applications and Policy,* Spring, pp. 52-58.

Blair, Gordon S. & Rodden, Tom (1994), *The Challenges of CSCW for Open Distributed Processing,* Bailrigg, Lancaster: Department of Computing, Lancaster University.

Boehm, Barry W. (1988), "A Spiral Model of Software Development and Enhancement", *Computer,* New York: IEEE Press, May, pp 61-72.

Boehm, Barry W. (1982), *Software Engineering Economics,* Englewood Cliffs, New Jersey: Prentice Hall.

Boehm, Barry, W. (1976), "Software Engineering", *IEEE Transactions on Computers,* New York: IEEE Press, December, C-25(12), pp. 1226-1241.

Booch, G. (1990), *On the Concepts of Object-Oriented Design,* New York: Van Nostrand-Reinhold.

Brodman, Judith G. & Johnson, Donna L. (1997), "A Software Process Improvement Approach Tailored for Small Organisations and Small Projects", *Proceeding of the 19th International Conference on Software Engineering,* New York: ACM Press, pp. 661-662.

Brooks Jr., Frederick P. (1986), "No Silver Bullet - Essence and Accident of Software Engineering", *Information Processing 86,* Amsterdam: Elsevier Science Publishers B.V. (North-Holland), pp 1069-1076.

Brooks Jr., Frederick P. (1979), *The Mythical Man-Month: Essays on Software Engineering,* Reading, Massachusetts: Addison-Wesley Publishing Company.

Brothers, L., Sembugamoorthy, V. & Muller, M. (1990), "ICICLE: Groupware for Code Inspection", *CSCW 90 Proceedings*, New York: ACM Press, October, pp. 169-181.

Brown, Marc H. & Najork, Marc A. (1996), "Distributed Active Objects", *Proceedings of the 5th International World Wide Web Conference,* Paris, May.

Bush, Vannevar (1945), "As We May Think", *Atlantic Monthly*, July, 176(1), pp. 101-108

Ciancarini, P. *et al.* (1996), "PageSpace: An Architecture to Coordinate Distributed Applications on the Web", *Proceedings of the 5th International World Wide Web Conference,* Paris, France, May.

Coad, P. & Yourdon, E. (1991a), *Object Oriented Analysis,* Englewood Cliffs, New Jersey: Prentice-Hall.

Coad, P. & Yourdon, E. (1991b), *Object Oriented Design,* Englewood Cliffs, New Jersey: Prentice-Hall.

Cockburn, Andy & Jones, Steve (1995), "Four Principles of Groupware Design", *Interacting with Computers,* Amsterdam: Elsevier Science Ltd., 7(2), pp. 195-210.

Cockburn, Andrew J.G. & Thimbleby, Harold (1991), "A Reflexive Perspective of CSCW", *SIGCHI Bulletin,* New York: ACM Press, July, 23(3), pp. 63-67.

Conte, S.D., Dunsmore, H.E. & Shen, V.Y. (1986), *Software Engineering Metrics and Model,* Menlo Park,California: The Benjamin/Cummings Publishing Co.

Cornel, G. & Horstmann, C.S. (1996), *core Java,* Mountain View, California: SunSoft Press, A Prentice Hall Title.

Cousin, Larry (1993), "Improving Software Quality through Practical CMM Level Progression", *Proceeding of the 11th Annual Pacific Northwest Software Quality Conference,* Oregon, October, pp. 240-256.

Cox, Brad J. (1986), *Object-Oriented Programming: An Evolutionary Approach,* Reading, Massachusetts: Addison-Wesley Publishing Company.

Curtis, Bill (1992), "The Software Process Movement: Current Status & Future Directions", *Singapore Computer Society Silver Jubilee Conference,* Singapore, October.

Daily, Kevin (1992), *Quality Management for Software,* Oxford, England: NCC Blackwell Limited.

Davies, M., Pettersen, K., Srinivasen, S. & Kinsman, K. (1994), "The Challenges of Designing Groupware: A Case Study", *Computer*, New York: IEEE Press, December, pp. 58-62.

DeJesus, Edmund X. (1996), "Control Quality", *Byte,* New York: McGraw-Hill Company, September, pp. 82-83.

DeMarco, T. & Lister, T. (1987), *Peopleware,* New York: Dorset House.

Dix A. (1997), "Challenges for Cooperative Work on the Web: An Analytic Approach", *Computer Supported Cooperative Work: The Journal of Collaborative Computing: Special Issue on CSCW and the Web,* Dordrecht: Kluwer Academic Publishers, 6(2-3).

Dobbins, James H. (1987), "Inspections as an Up-Front Quality Technique", in Schulmeyer, G. Gordon & McManus, James I. (eds.), *Handbook of Software Quality Assurance*, New York: Van Nostrand Reinhold, pp. 137-177.

Dobbins, James H. & Buck, Robert D. (1987), "The Cost of Software Quality", in Schulmeyer, G. Gordon & McManus, James I. (eds.), *Handbook of Software Quality Assurance*, New York: Van Nostrand Reinhold, pp. 119-136.

Dourish, Paul & Bellotti, Victoria (1992), "Awareness and Coordination in Shared Workspaces", in Turner, J. & Kraut, R. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 107-114.

Dunn, Robert H. (1987), "The Quest for Software Reliability", in Schulmeyer, G. Gordon & McManus, James I. (eds.), *Handbook of Software Quality Assurance*, New York: Van Nostrand Reinhold, pp. 342-384.

Ellis, C.A., Gibbs, S.J. & Rein, G.L. (1991), "Groupware: Some Issues and Experiences", *Communications of the ACM*, New York: ACM Press, January, 34(1), pp. 39-58

Fafchamps, Danielle & Garg, Pankaj (1994), "Computing Environments for Flexible Teams", in Taylor, Richard N. (ed.), *Proceedings of ICSE Workshop on Software Engineering and Human-Computer Interaction,* Berlin: Springer-Verlag, pp. 174-184.

Fagan, M.E. (1986), "Advances in Software Inspection", *IEEE Transactions on software engineering*, New York: IEEE Press, July, 12(7), pp. 744-751.

Fagan, M.E. (1976), "Design and Code Inspections to Reduce Errors in Program Development", *IBM System Journal,* International Business Machines, 15(1), pp. 182-211.

Fairley, R. (1985), *Software Engineering Concepts,* New York: McGraw-Hill.

Ferrans, James C. *et al.* (1992), "HyperWeb: A Framework for Hypermedia-Based Environments", *Software Engineering Notes,* New York: ACM SigSoft 92, December, 17(2), pp. 1-10.

Fisher, Alan S. (1991), *CASE: Using Software Development Tools,* New York: John Wiley & Sons, Inc..

Fletton, Nigel T. (1990), "A Hypertext Approach to Browsing and Documenting Software", in McAleese, R. & Green, C. (eds.), *Hypertext: State of the Art,* Oxford: Blackwell Scientific Pubs Ltd., pp. 193-204.

Flynn, J. & Clarke, B. (1995), "The World Wakes Up to Java", *Computer Technology Review,* Fall/Winter, pp. 34-37.

Fowler, J. *et al.* (1994), "Experience with the Virtual Notebook System: Abstraction in Hypertext", in Furuta, R. & Neuwirth, C. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 133-143.

Freedman, Daniel P. & Weinberg, Gerald M. (1990), *Handbook of Walkthroughs, Inspections, and Technical Reviews,* New York: Dorset House Publishing.

Gilb, Tom & Graham, Dorothy (1993), *Software Inspection,* Wokingham, England: Addison-Wesley Publishing Company.

Gintell, J.W., Arnold, J., Houde, M., Kruszelnicki, J., McKenney, R., & Memmi, G. (1993), "Srutiny: A Collaborative Inspection and Review System", *Proceedings of the 4th European Software Engineering Conference,*

Glance, Natalie S., Pagani, Daniele S., & Pareschi, Remo (1996), "Generalized Process Structure Grammars (GPSG) for Flexible Representations of Work", in Ackerman, Mark S. (ed.), *Proceeding of the ACM 1996 Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 180-189.

Goldfarb, Charles F. (1990), *The SGML Handbook,* Oxford: Oxford University Press.

Grady, Robert B. (1993), "Practical Results From Measuring Software Quality", *Communication of the ACM,* New York: ACM Press, November, 36(11), pp. 62-68.

Greif, Irene (1988), "CSCW: What Does It Mean?", *Proceedings of the CSCW 88*, September, New York: ACM Press.

Grinter, Rebecca E. (1997), "Doing Software Development: Occasions for Automation and Formalisation", in Hughes, J.A. *et al.* (eds.), *Proceeding of the 5th European Computer Supported Cooperative Work,* Dordrecht: Kluwer Academic Publishers, pp. 173-188.

Grudin, J. (1994), "Groupware and Social Dynamics: Eight Challenges for Developers", *Communications of the ACM*, New York: ACM Press, January, 37(1), pp. 92-105.

Grudin, J. (1993), "CSCW: History and Focus", Information and Computer Science Department, Irvine: University of California.

Grudin, J. (1991), "A Tale of Two Cities: Reflections on CSCW in Europe and the United States", *SIGCHI Bulletin,* New York: ACM Press, July, 23(3), pp. 22-24.

Grudin, J. & Poltrock, S. (1994), "Software Engineering and the CHI & CSCW Communities", *Proceeding of ICSE Workshop on Software Engineering and HCI,* Berlin: Springer-Verlag, pp. 93-112.

Haake, Jorg M. & Wilson, Brian (1992), "Supporting Collaborative Writing of Hyperdocuments in SEPIA", in Turner, J. & Kraut, R. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 138-146.

Hahn, Udo *et al.* (1991), "CoOUTHOR - A Hypermedia Group Authoring Environment", in Bowers, J.M. & Benford, S.D. (eds.), *Studies in Computer Supported Cooperative Work,* Amsterdam: Elsevier Science Publishers B.V., pp. 79-100.

Hedrick, terry E. (1993), *Applied Research Design, a Practical Guide,* London, Sage.

Hennessy, Phillippa (1990), *Modelling Group Communication,* PhD Thesis, Nottingham: Computer Science Department, University of Nottingham, October.

Higuera, Ronald P. & Gluch, David P. (1993), "Risk Management and Quality in Software Development", *Proceeding of the 11th Annual Pacific Northwest Software Quality Conference,* October, pp. 58-73.

Hiltz, Starr Roxanne & Turoff, Murray (1978), *The Network Nation,* Reading, Massachusetts: Addison-Wesley.

Hoffman, Douglas (1993), "The Role of the Software Quality Group in Software Development", *Proceeding of the 11th Annual Pacific Northwest Software Quality Conference",* Oregon, October, pp. 293-300.

Horn, R.E. (1989), *Mapping Hypertext: Analysis, Linkage, and Display of Knowledge for the Next Generation of On-Line Text and Graphics,* Massachusetts: The Lexington Institute.

Horton W., Taylor L., Ignacio A. & Hoft N.L., (1996), *The Web Page Design Cookbook: All the Ingredients You Need to Create 5-Star Web Pages,* New York: John Wiley & Sons Inc..

Huff, Sid L. (1993), "Object-Oriented Programming", *Business Quarterly,* Winter, 58(2), pp. 85-89.

Humphrey, Watts S. (1991), "CASE Planning and the Software Process", in Ng. P.A., Ramamoorthy, C.V., Seifert, L.C., & Yeh, R.T. (eds.), *Journal of System Integration,* Boston: Kluwer Academic Publisher.

Humphrey, Watts S. (1989), *Managing the Software Process,* Reading, Massachusetts: Addison-Wesley Publishing Company.

Humphrey, Watts S. (1988), "Characterizing the Software Process: A Maturity Framework, *IEEE Software,* New York: IEEE Press, March, pp. 73-79.

IEEE (1993), *IEEE Standard Glossary of Software Engineering Terminology,* New York: IEEE Press.

Jackson, Michael (1985), "The World and the Machine", *Proceeding of the International Conference on Software Engineering '85,* New York: ACM Press, pp 283-292.

Jacobson, Ivar *et.al.* (1992), *Object-Oriented Software Engineering,* Wokingham, England: ACM Press, Addison-Wesley Publishing Company.

Johansen, R. (1988), *Groupware: Computer Supported for Business Teams,* New York: The Free Press, Macmillan Inc.

Johnson, Philip M. (1994), "Supporting Technology Transfer of Formal Technical Review Through a Computer Supported Collaborative Review System", *Proceeding of the 16th International Conference on Software Engineering,* New York: ACM Press.

Johnson, Philip M. & Tjahjono, Danu (1997), "Assessing Software Review Meetings: A Controlled Experimental Study Using CSRS", *Proceeding of the 19th International Conference of Software Engineering,* New York: ACM Press, pp. 118-127.

Johnson, Philip M. & Tjahjono, Danu (1993), "Improving Software Quality through Computer Supported Collaborative Review", *Proceedings of the 3rd European Conference on Computer Supported Cooperative Work,* Dordrecht: Kluwer Academic Publisher, pp. 61-76.

Johnson, Philip M. *et al.* (1993), "Experiences with CSRS: An Instrumented Software Review Environment", *Proceedings of Pacific Northwest Software Quality Conference,* Portland, Oregon, pp. 301-316.

Jonassen, D.H., (1990), "Semantic Network Elicitation: Tools for Structuring Hypertext", in McAleese, R. & Green, C. (eds.), *Hypertext: State of the Art,* Oxford: Blackwell Scientific Pubs Ltd.

Jones, T.C. (1986), *Programming Productivity,* New York: McGraw-Hill.

Kaiser, Gail E. *et al.* (1997), "An Architecture for WWW-based Hypercode Environments", *Proceeding of the 19th International Conference on Software Engineering,* New York: ACM Press, pp. 3-13.

Kaplan, Simon M. *et al.* (1992a), "Flexible, Active Support for Collaborative Work with ConversationBuilder", in Turner, J. & Kraut, R. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 378-385.

Kaplan, Simon M. *et al.* (1992b), "Supporting Collaborative Software Development with ConversationBuilder", *Software Engineering Notes,* New York: ACM SigSoft 92, December, 17(2), pp. 11-20.

Kennedy A.J., (1995), *The Internet and World Wide Web: The Rough Guide,* London: Rough Guides Ltd..

Kitchenham, B.A., Kitchenham, A.P., & Fellows, J.P. (1986), "The Effects of Inspections on Software Quality and Productivity", *ICL Technical Journal,* Staffordshire, England: ICL, May, pp. 112-122.

King, Steven & Cohn, Robert (1993), "Software Configuration Management Tools: The Next Generation", *Proceeding of the 11th Annual Pacific Northwest Software Quality Conference,* October, pp. 411-426.

Knight, John C. & Myers, E. Ann (1993), "An Improved Inspection Technique", *Communications of the ACM,* New York: ACM Press, 36(11), pp. 50-61.

Knight, John C. & Myers, E. Ann (1991), "Phased Inspections and their Implementation", *Software Engineering Notes,* New York: ACM SIGSOFT, July, 16(3), pp. 29-35.

Knutson, Charles D. (1996), "Developing Quality", *Byte,* New York: McGraw-Hill Company, September, pp. 93-96.

Kraut, Robert (1990), "How Can We Make Groupware Practical?", Ensor, R. (moderator), *Proceedings of CHI 90,* April, New York: ACM Press.

Lai, Robert C.T. (1993), "The Move to Mature Processes", *IEEE Software,* New York: IEEE Press, July, 10(4), pp. 14-17.

Lee, Jang Ho *et al.* (1996), "Supporting Multi-User, Multi-Applet Workspaces in CBE", in Ackerman, Mark S. (ed.), *Proceeding of the ACM 1996 Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 344-353.

Leonhardt, Ulf *et al.* (1995), "Decentralised Process Enactment in a Multi-Perspective Development Environment", *Proceeding of the 17th International Conference on Software Engineering,* New York IEEE Computer Society Press, pp. 255-264.

Linden, Peter van der (1996), *just Java,* Mountain View, California: SunSoft Press, A Prentice Hall Title.

Lunt, Graeme Arnold (1990), *A Software Framework for Representing Organisational Communication,* PhD Thesis, Nottingham: University of Nottingham, May.

Macdonald, F.M., Miller, J., Brooks, A., Roper, M. & Wood, M. (1996), "Automating the Software Inspection Process", *Automated Software Engineering,* Boston: Kluwer Academic Publisher, Vol. 3, pp. 193-218.

Malone, Thomas W., Lai, Kum-Yew & Fry, Christopher (1992), "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work", in Turner, J. & Kraut, R. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 289-297.

Marcus, A. (1992), *Graphic Design for Electronic Documents and User Interfaces,* Massachusetts: Addison-Wesley Publishing Company.

Martin, Johnny & Tsai, W.T. (1990), "*N*-Fold Inspection: A Requirements Analysis Technique", *Communications of the ACM,* New York: ACM Press, February, 33(2), pp. 225-232.

Mashayekhi, V., Feulner, C., & Riedl, J. (1994), "CAIS: Collaborative Asynchronous Inspection of Software", *Proceeding of the 2nd ACM SIGSOFT Symposium on the Foundations of Software Engineering,* New York: ACM Press, pp. 21-34.

Mashayekhi, V., Drake, J.M., Tsai, W.T., & Reidl, J. (1993), "Distributed, Collaborative Software Inspection", *IEEE Software,* New York: IEEE Press, 10(5), pp. 66-75.

McKnight, C., Dillon, A. & Richardson, J. (1991), *Hypertext in Context,* Cambridge: Cambridge University Press.

Mills, Harlan, D., Dyer, Michael & Linger, Richard C. (1987), "Cleanroom Software Engineering", *IEEE Software,* New York: IEEE Press, September, pp. 19-24.

Myers, Glenford J. (1979), *The Art of Software Testing,* New York: Wiley-Interscience Publication.

Narayanaswamy, K. & Goldman, Neil (1992), ""Lazy" Consistency: A Basis for Cooperative Software Development", in Turner, J. & Kraut, R. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 257-264.

Nelson, Theodor (1981), *Literary Machines*, Sausalito, California: Mindful Press.

Nunamaker, J.F. *et al.* (1991), "Electronic Meeting Systems to Support Group Work", *Communications of the ACM,* New York: ACM Press, July, 34(7), pp. 40-61.

Nielson, J. (1990), *Hypertext and Hypermedia,* London: Academic Press Ltd.

Offutt, A. Jeffereson (1991), "An Integrated Automatic Test Data Generation System", in Ng. P.A., Ramamoorthy, C.V., Seifert, L.C., & Yeh, R.T. (eds.), *Journal of System Integration,* Boston: Kluwer Academic Publisher, pp. 129-147.

Ohno, Yutaka (1989), "Background and Current View of Software Engineering", in Matsumoto, Yoshiro & Ohno, Yutaka (eds.), *Japanese Perspectives in Software Engineering,* Singapore: Addison-Wesley Publishing Company, pp 1-18.

Olsen, Neil C. (1993), "The Software Rush Hour", *IEEE Software,* New York: IEEE Press, September, 10(5), pp. 29-37.

Olson, Judith S. & Teasley, Stephanie (1996), "Groupware in the Wild: Lessons Learned from a Year of Virtual Collocation", in Ackerman, Mark S. (ed.), *Proceeding of the ACM 1996 Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 419-427.

Olson, Judith *et al.* (1993), "Computer Supported Cooperative Work: Research Issues for the 90s", *Behaviour & Information Technology,* USA: Taylor & Francis Ltd., 12(2), pp. 115-129.

Opper, Susanna & Fersko-Weiss, Henry (1992), *Technology for Teams: Enhancing Productivity in Networked Organizations,* New York: Van Norstrand Reinhold.

Palfreyman, Kevin & Rodden, Tom (1996), "A Protocol for User Awareness on the World Wide Web", in Ackerman, Mark S. (ed.), *Proceeding of the ACM 1996 Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 130-139.

Paulk, Mark C., Curtis, Bill, Chrissis, Mary B., & Weber, Charles V. (1993), "Capability Maturity Model, Version 1.1", *IEEE Software,* New York: IEEE Press, July, 10(4), pp. 18-27.

Peek, J.D. (1991), *MH & xmh: E-Mail for Users and Programmers,* O'Reilly & Associates Inc.

Perpich, J.M. *et al.*, (1997), "Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development", *Proceeding of the 19th International Conference on Software Engineering,* New York: ACM Press, pp. 14-21.

Porter, Adam A., Siy, Harvey P., & Votta Jr., Lawrence G. (1997), "Understanding the Effects of Developer Activities on Inspection Interval", *Proceeding of the 19th International Conference on Software Engineering,* New York: ACM Press, pp. 128-138.

Pressman, Roger S. (1992), *Software Engineering: A Practitioner's Approach (3rd edition),* New York: McGraw-Hill Inc.

Prinz, Wolfgang (1994), "Object-Oriented Organization Modeling for the Support of CSCW", *Proceeding of the 27th Hawaii International Conference on System Sciences,* New York: IEEE Computer Society Press, Vol. IV, pp. 797-806.

Reese, Jon D. & Leveson, Nancy G. (1997), "Software Deviation Analysis", *Proceeding of the 19th International Conference of Software Engineering,* New York: ACM Press, pp. 250-260.

Rein, G.L., McCue, L., & Slein, J.A. (1997), "A Case for Document Management Functions on the Web", *Communications of the ACM,* New York: ACM Press, September, 40(9), pp. 81-89.

Roberts, Bill (1996), "Groupwar Strategies", *Byte,* New York: McGraw-Hill Company, July, pp. 68-78.

Robinson, Mike (1993), "Design for Unanticipated Use.....", in De Michelis, G., Simone, C., & Schmidt, K. (eds.), *Proceeding of the 3rd European Conference on Computer Supported Cooperative Work,* Dordrecht: Kluwer Academic Publishers, pp. 187-202.

Rodden, Thomas (1990), *Supporting Cooperation in Software Engineering Environments,* PhD Thesis, Lancaster: Department of Computing, University of Lancaster.

Rodden, Tom & Blair, Gordon S. (1992), "Distributed Systems Support for Computer Supported Cooperative Work", *Computer Communication,* Butterworth-Heinemann Ltd., October, 15(8), pp. 527-538.

Rogers, A.S. (1994), "An Introduction to Groupware and CSCW", *BT Technology Journal,* July, 12(3), pp. 7-11.

Roth, Tina (1994), "Hypermedia Support for Software Development: A Retrospective Assessment", *Hypermedia,* London: Taylor Graham Publishing, 6(3), pp. 149-173.

Royce, W.W. (1970), "Managing the Development of Large Software Systems", *Proceedings of IEEE WESCON*, New York: IEEE Press., pp. 1-9.

Rumbaugh, J. *et al.* (1991), *Object-Oriented Modelling and Design,* New Jersey: Prentice Hall.

Russell, Glen W. (1991), "Experience with Inspection in Ultralarge-Scale Developments", *IEEE Software,* New York: IEEE Press, January, pp. 25-31.

Schah, S.R. (1993), *Software Engineering,* Illinois: Irwin and Aksen Associates

Schulmeyer, G. Gordon (1990), *Zero Defect Software,* New York: McGraw-Hill

Schulmeyer, G. Gordon (1987a), "Software Quality Assurance - Coming to Terms", in Schulmeyer, G. Gordon & McManus, James I. (eds.), *Handbook of Software Quality Assurance*, New York: Van Nostrand Reinhold, pp. 1-13.

Schulmeyer, G. Gordon (1987b), "Standardisation of Software Quality Assurance", in Schulmeyer, G. Gordon & McManus, James I. (eds.), *Handbook of Software Quality Assurance*, New York: Van Nostrand Reinhold, pp. 79-103.

Sellars, P. (1987), "IPSEs in Support of Teams", in Benyom, D. & Skidmore, S. (eds.), *Automating Systems Development,* New York: Plenum Press.

Shen, HongHai & Dewan, Prasun (1992), "Access Control for Collaborative Environments", in Turner, J. & Kraut, R. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work,* New York: ACM Press, pp. 51-58.

Shim, H., Hall, R.W., Prakash, A., & Jahaniam, F. (1997), "Providing Flexible Services for Management Shared State in Collaborative Systems", in Hughes J. *et al.* (eds.), *Proceeding of the 5th European Conference on Computer Supported Cooperative Work,* Dordrecht: Kluwer Academic Publishers, pp. 237-252.

Shneiderman, B. & Kearsley, G. (1989), *Hypertext Hands-On: An Introduction to a New Way of Organizing and Accessing Information Reading,* Reading, Massachusetts: Addison-Wesley Publishing Company.

Smith, David J. & Wood, Kenneth B. (1989), *Engineering Quality Software: A Review of Current Practices, Standards and Guidelines including New Methods and Development Tools,* London: Elsevier Science Publishers.

Sneed, Harry, M. (1989), *Software Engineering Management*, Chichester, England: Ellis Horwood Limited.

Sodhi, Jag (1991), *Software Engineering: Methods, Management, and CASE Tools,* Blue Ridge Summit, Pennsylvania: TAB Books.

Sproull, Lee (1990), "How Can We Make Groupware Practical?", Ensor, R. (moderator), *Proceedings of CHI 90*, April, New York: ACM Press.

Stein, Michael *et al.* (1997), "A Case Study of Distributed, Asynchronous Software Inspection", *Proceeding of the 19th International Conference on Software Engineering,* New York: ACM Press, pp. 107-117.

Strange, Michael (1993), "Management Report: Metrics to Evaluate the Quality of Your Software", *Proceeding of the 11th Annual Pacific Northwest Software Quality Conference,* Oregon, October, pp. 328-342.

Suchman, Lucy A. (1983), "Office Procedures as Practical Action: Models of Work and System Design", *ACM Transactions on Office Information Systems,* New York: ACM Press, 1(4), pp. 320-328.

Sun (1995), *The Java Language Environment: A White Paper,* Mountain View, California: Sun Microsystems Computer Company, October.

Tanenbaum, A.S. (1981), *Computer Networks*, New Jersey: Prentice-Hall.

Tatters, Wes (1996), *Teach Yourself Netscape Web Publishing in a Week,* Indianapolis, Indiana: Sams.net Publishing.

Trevor, Jonathan, Koch, Thomas, & Woetzel, Gerd (1997), "MetaWeb: Bringing Synchronous Groupware to the World Wide Web", in Hughes, J.A. *et al.* (eds.), *Proceeding of the 5th European Computer Supported Cooperative Work,* Dordrecht: Kluwer Academic Publishers, pp. 65-80.

Trevor, Jonathan, Rodden, Tom, & Blair, Gordon (1993), "COLA: A Lightweight Platform for CSCW", in De Michelis, G., Simone, C. & Schmidt, K. (eds.), *Proceeding of the 3rd European Conference on Computer Supported Cooperative Work,* Dordrecht: Kluwer Academic Publishers, pp. 15-30.

Twidale, M., Rodden, T., & Sommerville, I. (1993), "The Designers' Notepad: Supporting and Understanding Cooperative Design", in De Michelis, G., Simone, C. & Schmidt, K. (eds.), *Proceeding of the 3rd European Conference on Computer Supported Cooperative Work,* Dordrecht: Kluwer Academic Publishers, pp. 93-108.

UTM (1994), *Research Design: Lets Go for It!,* Johor Bahru, Malaysia: University Technology of Malaysia.

Vaughan-Nichols, Steven J. (1997), *Intranets,* Boston: AP Professional, A Division of Academic Press, Inc.

Vessey, I. & Sravanapudi, A.P. (1995), "CASE Tools as Collaborative Support Technologies", *Communications of the ACM*, New York: ACM Press, January, 38(1), pp. 83-95.

Wang, Bing (1993), *Integrating Database and Hypertext to Support Document Environments,* PhD Thesis, Department of Computer Science, University of York, March.

Weinberg, Gerald M. & Freedman, Daniel P. (1984), "Reviews, Walkthroughs, & Inspections", *IEEE Transaction on Software Engineering,* New York: IEEE Press, January, SE-10(1), pp. 68-72.

Weller, Edward F. (1993), "Lessons from Three Years of Inspection Data", *IEEE Software,* New York: IEEE Press, September, 10(5), pp. 38-45.

Wilson, P. (1991), *Computer Supported Cooperative Work: An Introduction,* Oxford: Intellect Books.

Wilson, P. (1990), "Computer Supported Cooperative Work: An Overview", *Intelligent Tutoring Media*, 1(3).

Winograd, T. & Flores, F. (1986), *Understanding Computers and Cognition, a New Foundation for Design,* Norwood, New Jersey: Abblex.

Yasuda, K. (1989), "Software Quality Assurance Activities in Japan", in Matsumoto, Y. & Ohno, Y. (eds.), *Japanese Perspective in Software Engineering,* Singapore: Addison-Wesley Publishing Company, pp.

Yourdon, Edward (1996), "When Good Enough is Best", *Byte,* New York: McGraw-Hill Company, September, pp. 85-90.

Yourdon, Edward (1989), *Structured Walkthrough,* Englewood Cliffs, New Jersey: Yourdon Press, A Prentice-Hall Company.

Zeb, Akmal Jahan (1993), *Browsing versus Searching in a Collaborative Authoring Hypertext,* PhD Thesis, Department of Computer Science, University of Liverpool, March, 1993.

BIBLIOGRAPHY

Button, Graham & Sharrock, Wes (1996), "Project Work: The Organisation of Collaborative Design and Development in Software Engineering", *Computer Supported Cooperative Work: The Journal of Collaborative Computing,* Dordrecht: Kluwer Academic Publishers, 5, pp. 369-386.

Choo, C.K. (1987), "Statistical Methods Applied to Software Quality Control", in Schulmeyer, G. Gordon & McManus, James I. (eds.), *Handbook of Software Quality Assurance*, New York: Van Nostrand Reinhold, pp. 285-341.

Dossick, S.E. & Kaiser, G.E. (1996), "WWW Access to Legacy Client/Server Applications", *Proceeding of the 5th International World Wide Web Conference,* Paris, France, May.

Ellis, C. & Wainer, J. (1994), "A Conceptual Model of Groupware", in Furuta, R. & Neuwirth, C. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work 94,* New York: ACM Press, pp. 79-88.

Grudin, J. & Poltrock, S. (1994), "Software Engineering and the CHI & CSCW Communities", *Proceeding of ICSE Workshop on Software Engineering and HCI,* Berlin: Springer-Verlag, pp. 93-112.

Lott, C.T. & Rombach, H.D. (1996), "Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques", *Empirical Software Engineering,* Boston, Kluwer Academic Publishers, 1, pp. 241-277.

Rogers, Y. (1994), "Exploring Obstacles: Integrating CSCW in Evolving Organizations", in Furuta, R. & Neuwirth, C. (eds.), *Proceedings of the Conference on Computer Supported Cooperative Work 94,* New York: ACM Press, pp. 67-77.

Seaman, Carolyn B. & Basili, Victor R. (1997), "An Empirical Study of Communication in Code Inspections", *Proceeding of the 19th International Conference on Software Engineering,* New York: ACM Press, pp. 96-106.

Sikkel, Klaas (1997), "A Group-based Authorization Model for Cooperative Systems", in Hughes, J. *et. al.* (eds.), *Proceedings of the 5th European Conference on Computer Supported Cooperative Work,* Dordrecht: Kluwer Academic Publisher, pp. 345-360.

Smith, H.T., Hennessy, P.A., & Lunt, G.A. (1991), "An Object-Oriented Framework for Modelling Organisational Communication", in Bowers, J.M. & Benford, S.D. (eds.), *Studies in Computer Supported Cooperative Work,* Dordrecht: Elsevier Science Publishers B.V.

Vessey, I., Jarvenpaa, S., & Tractinsky, N. (1992), "CASE Tools as Methodology Companion", *Communications of the ACM*, New York: ACM Press, January, 35(4), pp. 90-105.

Fusaro, P. & Lanubile, F. & Visaggio, G. (1997), "A Replicated Experiment to Assess Requirements Inspection Techniques", *Empirical Software Engineering,* Boston, Kluwer Academic Publishers, 2, pp. 39-57.

LIST OF PAPERS PRODUCED

Doherty, Bernard S. & Sahibuddin, Shamsul (1997), "Software Quality through Distributed Code Inspection Groupware", in Tasso, S., Adey, R.A., & Pighin, M. (eds.), *Software Quality Engineering,* Southampton: Computational Mechanics Publications, pp. 159-168.

Doherty, Bernard S. & Sahibuddin, Shamsul (1996), "Modelling Distributed Code Inspection Groupware", *Proceedings of the REDECS '96,* Serdang, Malaysia: Universiti Putra Malaysia Publications, June, pp. 69-73.

Appendix A

FLEXSIG USER MANUAL

A.0 INTRODUCTION

This manual provides information on how to use the FlexSIG prototype. It describes the inspection process according to FlexSIG in general. It also provides detailed information on each component of the prototype. The manual is intended for the members of the inspection team.

A.1 OVERVIEW OF FLEXSIG

The FlexSIG prototype is based on the FlexSIG model. In the FlexSIG model, the inspection process starts with the initiation phase, followed by the kick-off meeting or the briefing phase, the individual inspection phase, the synchronous group inspection or the asynchronous group inspection, the consolidation, and the follow-up phase. The FlexSIG system is based on the FlexSIG model of software inspection. Nevertheless, it is flexible enough to be used as a toolset to support software inspection based on Fagan's, Gilb & Graham's, and Humphrey's model of software inspection.

A.2 ACCESSING AND EXITING FLEXSIG

This section describes how to access and exit the system. It also describes how to change the password. The URL of the system, the username, and the password should have been distributed in advance by the moderator.

A.2.1 Accessing the System

To access the system, follow these instructions :

1. Open a web browser
2. Open the login page using the URL given by the moderator.
3. Enter your username and the password in the space allocated.
4. To proceed, click

   Submit

5. To clear the field, click

   Reset

A.2.2 Changing the Password

To change the password :

1. Click the following option from the main menu of FlexSIG

   Change Password

2. The change password applet is displayed. Type in the username and old password. Enter the new password and reconfirm it by typing the new password again.

3. To proceed with the changes, click

   Change

4. To clear the information entered, click

   Clear

A.2.3 Exiting the System

To exit from the system :

1. Click the following button from the main menu of FlexSIG

   Quit

2. A confirmation window will be displayed, to proceed, click

   Proceed

3. To cancel, and go back to FlexSIG system, click

   Cancel

A.3 SUPPORTING DOCUMENTS

There are three types of supporting documents available, namely, briefing, specification, and help components. Briefing components contain information related to the organisational matters of the inspection process. The specification components include specification documents related to the document being

inspected. The help component consist of help information with regard to using the FlexSIG system and additional information on the implementation language and the operating system of the document inspected.

A.3.1 Briefing

To access the briefing material from the system, do the following :

1. Click the following option from the menu

   Briefing

2. The briefing menu will be displayed in the information frame. From the briefing menu, choose any of the six briefing documents.

3. The document will be displayed in a new browser window. The windows can be left open or closed down after finishing with the material.

A.3.2 Specification

The specification document availability depends on the type of the document inspected. If the document inspected is program code, then the specification component contains the requirement and design specification. If the document inspected is a design specification document, then the specification component contains the requirement specification only. In the case of requirement specification inspection, the specification component is empty.

To access the specification material from the system :

1. Click the following option from the menu

   Specification

2. The specification menu will be displayed in the information frame. From the specification menu, choose any of the documents.

3. The document will be displayed in a new browser window. The windows can be left open or closed down after finishing reading the material.

A.3.3 Help

The help component consist of three types of information, namely, the help information on FlexSIG, help information on the programming language, and help information on the operating system. The information on the programming language and the operating system are only available for the code inspection process.

To access the help information from the system :

1. Click the following option from the menu

   Help

2. The help menu will be displayed in the information frame. From the help menu, choose any of the help information.

3. The information will be displayed in a new browser window. The windows can be left open or closed down after finish reading the material.

A.4 BROWSING THE INSPECTED DOCUMENT

The document browsing module consist of two components, specifically, the document viewer and the remote viewer. The document viewer is used to view

the inspected document. Meanwhile, the remote viewer is used to view another user's inspection document.

A.4.1 Document Viewer

The document viewer can be accessed using the following step :

1. Click the following option from the menu

   Code Viewer

2. A new web browser with the viewing applet will be opened.
3. Select the filename of the document to be inspected by typing in the name or choosing it from the list.
4. To proceed with the process, click

   Fetch

The document will then be loaded into the applet.

5. To clear the filename entered, click

   Clear Filename

6. The web browser windows should be closed down after finish inspecting the document.

A.4.2 Remote Viewer

The remote viewer can be accessed using the following step :

1. Click the following option from the menu

   View Remote

2. A new web browser with the remote viewing applet will be opened.

3. Select the name of another user from the list of user currently using the system.

4. To proceed with the process, click

   Fetch

The document viewed by the particular user selected will then be loaded into the applet.

5. To update the list of user currently using the system, click

   Refresh

6. The web browser windows should be closed down after finish inspecting the document.

A.5 DEFECTS AND LOG FILE

The defects and queries found during the individual and group inspection is stored in the log file. The log file is entered and can be modified by the inspectors. The moderator can delete any entry. The author can only view the log file.

A.5.1 Add Log Entry

To add a log entry into the log file :

1. Click the following option from the menu

   Log Entry

2. The log entry menu will be displayed in the information frame. Click the following option

   Add Log

3. A new web browser with the add log applet will be open.

4. Enter the defect information into the applet.

5. To submit the entry, click the following button

   Submit

6. To clear the entry, click the following button

   Clear

A.5.2 Modify Log Entry

To modify or delete a log entry from the log file :

1. Click the following option from the menu

   Log Entry

2. The log entry menu will be displayed in the information frame. Click the following option

   Modify Log

3. A new web browser with the modify log applet will be open.

4. Enter the filename of the document which the log relates to and click

   Fetch List

5. Select the log entry from the list and click

   Fetch Log Entry

6. The information displayed in the log applet can be modified accordingly.

7. To submit the modified entry, click the following button

Modify

or to delete the log entry, click the following button

Delete

A.5.3 View Overall Log Entry

The overall log for each document inspected can be viewed by every member of the inspection team. This component also serves as the voting mechanism to accept or reject any log entry that is in dispute. Only the inspectors are allowed to vote on the log entry.

To view the overall log entry for each document inspected :

1. Click the following option from the menu

Log Entry

2. The log entry menu will be displayed in the information frame. Click the following option

View Overall Log

3. A new web browser with the view overall log applet will be open.

4. Enter the filename of the document which the log relates to and click

Fetch List

5. To view the individual log entry, select the log entry from the list and click

Fetch Log Entry

6. The information displayed in the log applet can only be viewed.

7. If there is any dispute on any log entry during the discussion, the voting mechanism can be employed. Voting is for the inspectors only, and can be done by clicking the following button to accept the log entry

   Accept

or to reject the log entry, click

   Reject

8. The moderator can mark any log entry to be deleted during consolidation if there is any duplication by clicking

   Delete

A.6 COMMUNICATIONS

There are two major communication modules, namely, the synchronous communication and the asynchronous communication. The synchronous communication consists of the electronic chat component and the pop message component. The asynchronous communication consists of the e-mail component and the bulletin board component.

A.6.1 Chat

There are two types of chat available, namely, the group chat and the personal chat. The group chat involves the whole inspection team in a room set-up by the system. The personal chat can be used by two or more people. The participants need to agree the name of the room beforehand.

The chat component can be accessed using the following steps :

1. Click the following option from the menu

   Chat

2. The chat menu will be displayed in the information frame. Click the following option to start chat with the whole group

   Group Chat

or to start the personal chat, click

Personal Chat

and type the personal room name.

3. A new web browser with chat applet will be opened.

4. Type in your username and proceed with the discussion

A.6.2 Pop Message

The pop message causes a short message to pop-up on another user's screen. The pop message component can be accessed using the following steps :

1. Click the following option from the menu

   Pop Message

2. The pop message applet will be displayed in the information frame.

3. Type in your message in the area given. To send the message to any particular user, click from the list

   username

or to send the message to everybody in the inspection group, click

*

4. To proceed with process, click

   Send

5. To clear the message composition area, click

   Clear

A.6.3 E-mail

The e-mail component consist of two applets, namely, the send e-mail and the read e-mail applets.

Send E-mail

To send an e-mail :

1. Click the following option from the menu

   E-mail

2. The e-mail menu will be displayed in the information frame, click

   Send E-mail

3. The send e-mail applet will be displayed in a new browser window. Select the recipient from the list, and type in the subject and the contents of the e-mail.

4. To send the e-mail, click

   Send

5. To clear the information entered, click

Clear

Read E-mail

To read an e-mail :

1. Click the following option from the menu

   E-mail

2. The e-mail menu will be displayed in the information frame, click

   Read E-mail

3. The read e-mail applet will be displayed in a new browser window. The list of e-mail in the mailbox is displayed

4. To fetch the contents of the e-mail, select the e-mail from the list, click

   Fetch

5. Or to delete any e-mail entry, select the e-mail, click

   Delete

6. Or to reply any e-mail message, select the e-mail, click

   Reply

Enter the subject and the contents of the reply in the composition area. To send reply, click

Send

To clear information entered, click

Clear

A.6.4 Bulletin Board

To read and post a notice on the electronic bulletin board :

1. Click the following option from the menu

   BBS

2. The bulletin board applet will be displayed in a new browser window. The list of notice on the electronic board is displayed.

3. To fetch the contents of the notice, select the notice from the list, click

   Fetch

4. To reply any notice, select the message in the list, click

   Reply

5. To post a new message on the board, click

   New

6. Enter the subject and the contents of the notice in the composition area. To post the message, click

   Post-It

To clear information entered, click

Clear

A.7 CONSOLIDATION AND REPORTS

These components are accessible only by the moderator. The consolidation of the log file and report generation is conducted at the end of the software inspection process.

A.7.1 Consolidation

The consolidation of log file and collecting the inspection metrics is done using the following step :

1. Click the following option from the menu

   Log Compilation

2. The log compilation menu will be displayed in the information frame. To consolidate the log file and collect the inspection metrics, click

   Consolidate

A.7.1 Report

The report of the inspection which includes the log file and the metrics collected can be generated using the following steps :

1. Click the following option from the menu

   Log Compilation

2. The log compilation menu will be displayed in the information frame. To generate the report, click

   Report Generation

3. The report generation applet will be displayed in a new web browser window. Select the filename of the document to be printed and click

   Print

A.8 CONFIGURING FLEXSIG

These options are available to the moderator only. There are four kinds of configuration that need to be done, namely, setting the inspection information file, the inspection document file, the password file, and resetting the database file.

A.8.1 Process Information

The inspection process information file can be configured using the following steps :

1. Click the following option from the menu

   Configuration

2. The configuration menu will be displayed in the information frame. Select

   Inspection Process Information

3. The inspection process information applet will be displayed in a new browser window. The current information on the process is displayed.

4. The contents can be modified, and to submit changes, click

   Submit

5. To clear the information entered, click

   Clear

A.8.2 Inspected Document List

The inspected document list file can be configured using the following steps :

1. Click the following option from the menu

   Configuration

2. The configuration menu will be displayed in the information frame. Select

   Inspected Document List

3. The inspected document list applet will be displayed in a new browser window with the current information which includes the location of the file, the number of documents, and the name of the documents.

4. The contents can be modified, and to submit changes, click

   Submit

5. To clear the information entered, click

   Clear

A.8.3 Password File

The password file can be configured using the following steps :

1. Click the following option from the menu

   Configuration

2. The configuration menu will be displayed in the information frame. Select

   Password File

3. The password file applet will be displayed in a new browser window with the current list of user information.

4. To add new entry in the password file, type the information in the field given and click

   New

5. The change any information from the password file, select the required username and click

   Fetch

6. The contents can be modified, and to submit changes, click

   Change

7. To clear the information entered, click

   Clear

A.8.3 Resetting Database File

At the conclusion of the software inspection process, the database needs to be reset to its original condition so that a new inspection process can take place. To reset the database :

1. Click the following option from the menu

   Configuration

2. The configuration menu will be displayed in the information frame. Select

   Reset Database

A.9 LIST OF DATABASE FILES

1. idFile.dat
2. usrPasswd.dat
3. fileLocList.dat
4. currentUsr.dat
5. loadedCodeInfo.dat
6. ProjDesc.brief
7. CurrentCodeDesc.brief
8. GroupDesc.brief
9. CurrentInspectDesc.brief
10. SoftwareInspectDesc.brief
11. FlexSIGDesc.brief
12. ProjReq.spec
13. ProjDesign.spec
14. FlexSIG.help
15. Documents to be inspected
16. Log file for each documents inspected
17. Inspection metric files
18. Mailbox files

Appendix B

COMPONENT DESCRIPTION

B.0 FLEXSIG COMPONENTS

This appendix describes the components of FlexSIG. The components are described according to their function. The access control authorises valid users to initiate FlexSIG and assign access level according to their roles. The briefing, specification, and help components assist users in understanding and using FlexSIG in the software inspection process. The document inspection and communication component is used in the individual and group phases of inspection itself. The comment log component is used to enter potential defects, queries, and suggestion. The metrics and reports and system configuration are only accessible by the moderators and system administrator.

B.1 BRIEFING

The briefing section is used differently according to the mode of the software inspection process decided by the moderator. In the face-to-face mode, the briefing component is just support material to the kick-off meeting. In the other modes, namely the distributed synchronous and distributed asynchronous mode, the briefing component replaces the kick-off meeting. The briefing component is divided into six sub-sections. They are: project description,

document description, group information, current process description, software inspection process, and FlexSIG description (see *Table B - 1*).

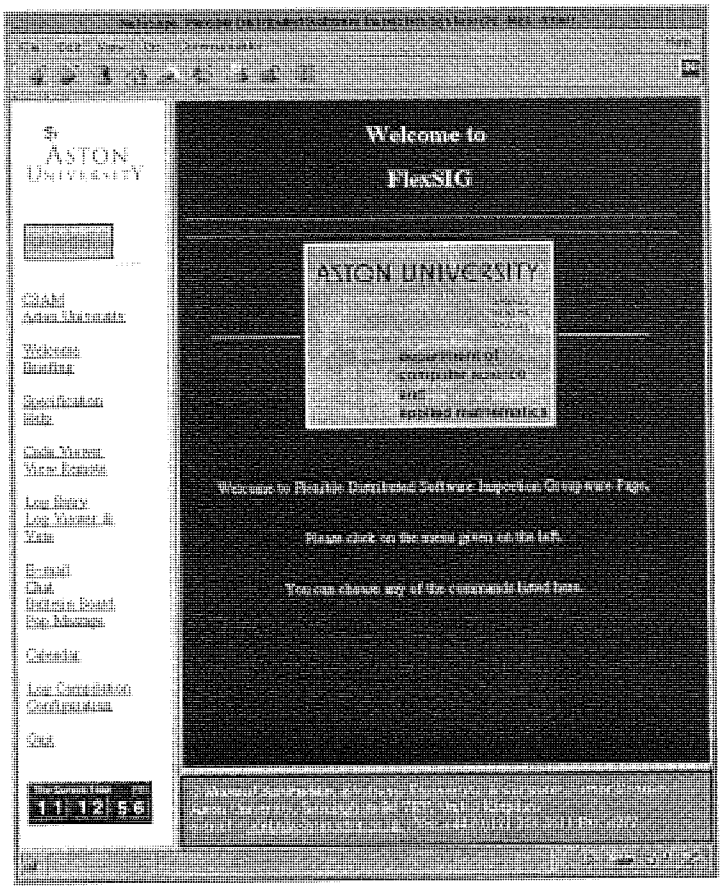

Figure B - 1 : *The Starting Page of FlexSIG*

The project description describes the current project to which the inspected document belongs. The project description includes the background, objectives, phases of the project, and other information that helps the team members for the software inspection process to understand the whole project. This is vital because the information on the project as a whole leads to a better understanding of the document being inspected.

The document description explains the inspected document in relation to the project as a whole. The objective and the specific task of each document is explained in detail. The relationship of the document with other parts of the project is laid out.

Type of Briefing Doc.	Description
Group Information	The composition of the team. Includes name, background, roles.
Current Process Description	Describes the objective and task of current inspection
Project Description	Describes the software project the document belongs to.
Document Description	Explains the objective and the function of the document inspected
Software Inspection Process	Explains the software inspection process in general
FlexSIG Description	Spells out the FlexSIG methods for conducting software inspection.

Table B - 1 : *Type of Briefing Document*

The group information relates to the background information and the structure of the software inspection team. Some background information on team members is included in order to introduce one team member to another. The role of each team member is specified here. The user name and password is distributed in a more discreet way, namely using the e-mail facility in the organisation. Each member's role objective is described in detail.

The objective and the task of the team is specified in the description of the current software inspection process. Specific tasks for certain individual, if any, are outlined here. In this briefing sub-section, the moderator explains the mode of the software inspection process, whether it is in distributed synchronous, distributed asynchronous, or face-to-face mode. The time frame and milestone for each phase and also for the whole process is laid down. Any meeting timetable for distributed synchronous meetings is specified.

The software inspection process description explains the process in general. It describes how software inspection in general is conducted. Differences between several major methods are point out. The strengths and weaknesses

of software inspection are presented to the user. This information is useful if there is any member of the team who is not familiar with the process.

The FlexSIG description sub-component spells out the FlexSIG version of the software inspection process. It describes the implementation of FlexSIG in general, not the detailed description of current software inspection process. The general model and the architecture of FlexSIG is presented. The motive here again is to familiarise the team members with process so that the software inspection process can achieve the maximum benefit.

B.2 SPECIFICATION

The contents of specification components depends on the type of document to be inspected. If the document to be inspected is program code, then the specification contains requirement specification and design specification material. If the document to be inspected is a design specification document, then the specification contains only the requirement specification. The requirement specification and the design specification are provided by the software project development team. The information is provided on the basis of a read-only policy by the development team. The aim of providing this information is to instil better understanding of the software project as a whole. The information also serves as a reference point if there is any confusion with regard to the inspected document.

B.3 DOCUMENT INSPECTION

The browsing components allows the team member to browse the documents that need to be inspected. There are two types of browsing method. The first

type allows the team member to select documents from a given list. The second type permits the user to select the document that another user is currently inspecting.

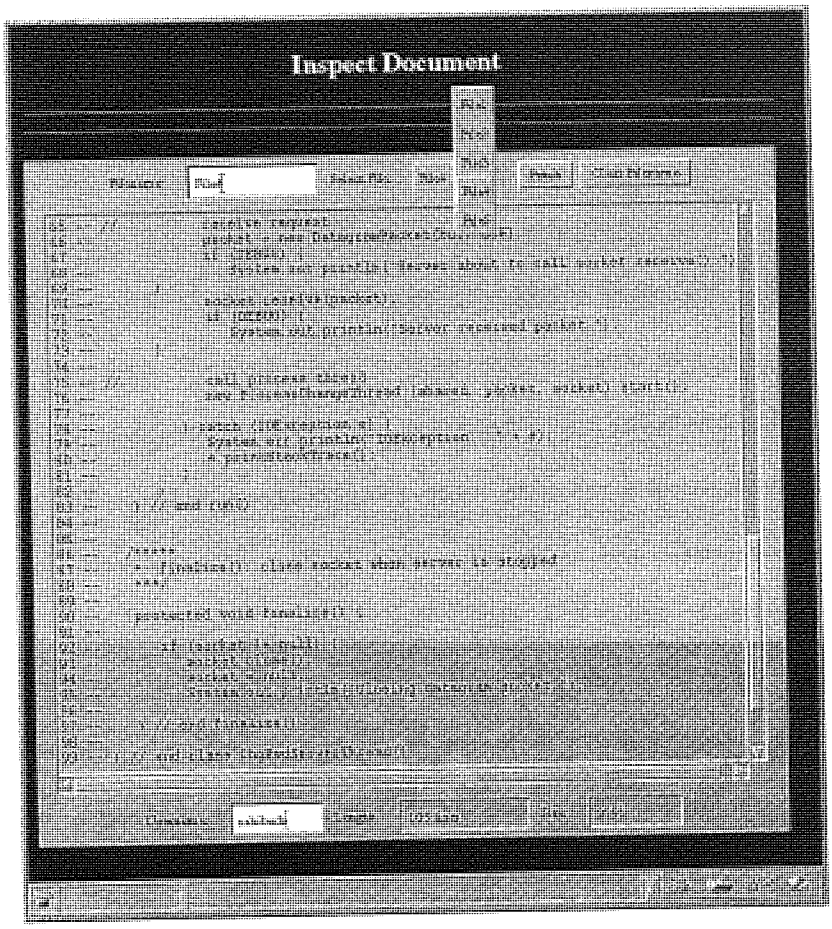

Figure B - 2 : *Document Browsing Applet*

B.3.1 Document Viewer

The document viewer allows the user to type the filename of the document to be inspected. Alternatively, they can select the document name from a pull-down list (see *Figure B - 2*). The filename selected is sent to the server. The server program opens the appropriate document. The contents of the document are then sent back to the client machine and displayed in the viewer applet.

Line numbers are added automatically to assist team members in the inspection process. The user can browse the document displayed in the applet. The document viewer is used both during the individual inspection phase and during the group inspection phase.

B.3.2 Remote Viewer

The remote browsing component provides the capability for a team member to view the same document that the another member is inspecting. This facility is to provide a short-cut for the team member when there is a need for a discussion over the same document. Rather than sending a message asking the other party the filename of the inspected document, the first user can just select the second party's username from the list (see *Figure B - 3*). The list of names provides information of current users using the document viewing component. When a username is selected from the list, the name is sent to the server. Based on a look-up table, the server program then reads the name of the document inspected by the second party. The document is then fetched and sent back to the first user. Both members of the inspection team can now proceed with their discussion using the communication component provided in FlexSIG, referring to the same document. This component is used during the distributed synchronous group inspection phase.

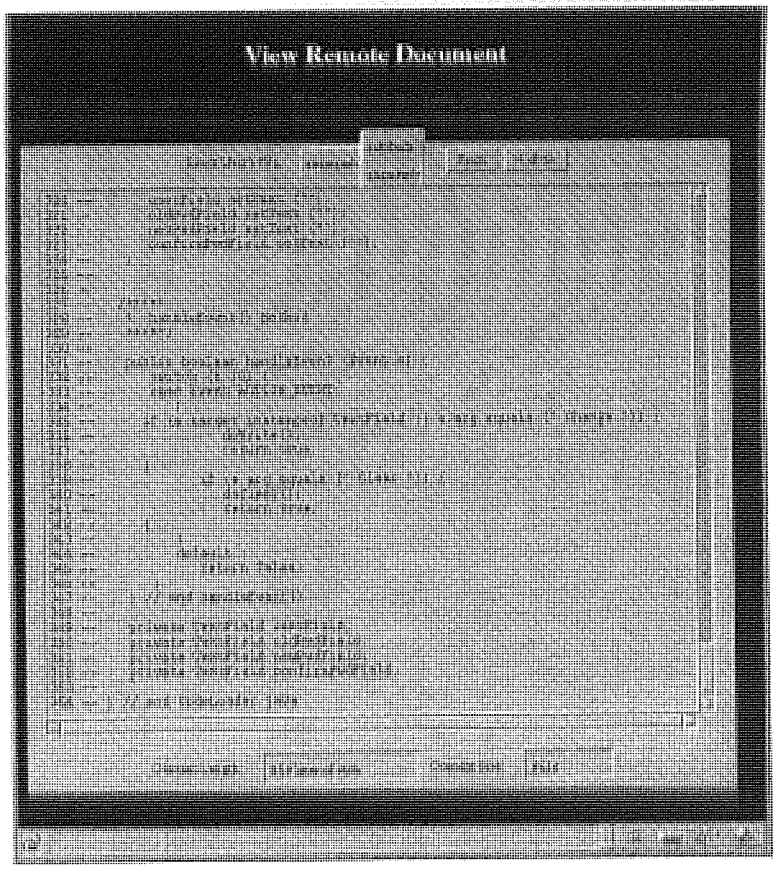

Figure B - 3 : *Remote Browsing Applet*

B.4 COMMENT LOG

This component is responsible for collecting team members comments, namely the defects, the queries, and the suggestions. Comment can be logged during the individual inspection phase or during the group inspection phase. Whenever a potential defect is identified in the document inspected, the log entry applet can be activated (see *Figure B - 4*). The log entry applet will create the process and project number automatically. The inspector is required to enter the filename of the document inspected and the line number. The criticality, which was explained in the fifth chapter, of the log item can be selected from a list. The type of the log entered is available from a list. The inspector is expected to explain the log entry in the comment windows. Any inquiries or

question of intent should be noted here also. If there is any suggestion on how to improve the defects identified, the inspector can enter the recommendation in the suggestion window.

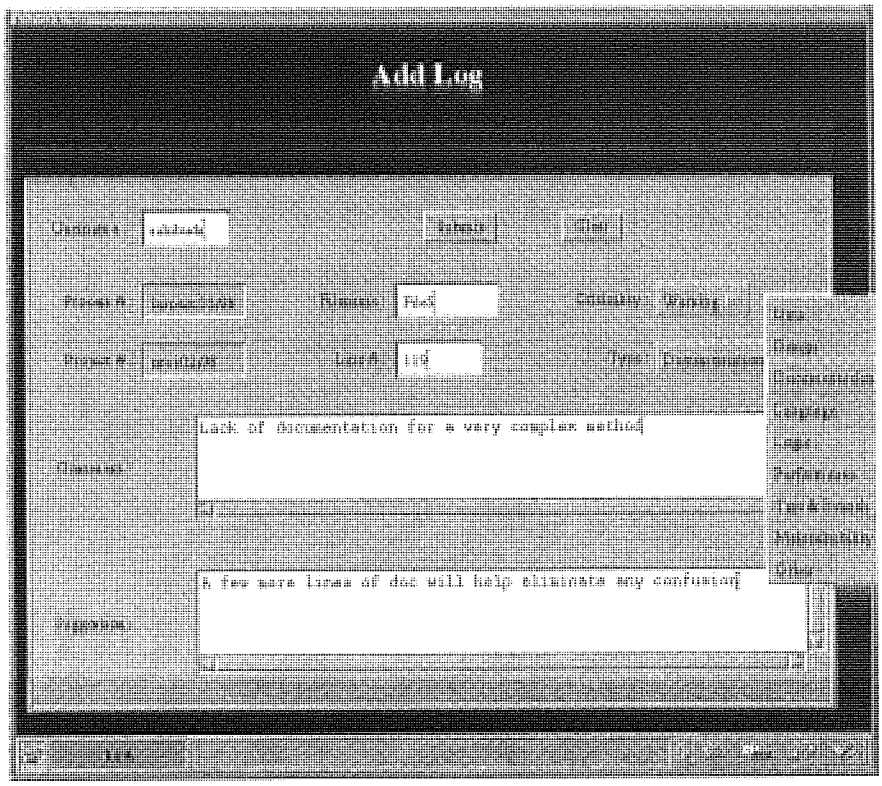

Figure B - 4 : *Entering Comment into Log File*

The comment is sent from the applet to the server to be processed. The server program parses the data entered and stores the information in the appropriate log file. A separate log file is kept for each document inspected. A log number is generated as an identifier. The username is used to identify the owner of the comment. Only the owner can modify their own entry. The author can only view the log file. The moderator can view the log file during the individual and group inspection phases but cannot enter any comment. However, the moderator has the right to override and delete any entry during consolidation to resolve any outstanding problem.

B.5 COMMUNICATION

The communication suite can be divided into two major sections. The first category deals with the synchronous mode of communication. The second category deals with asynchronous mode of communication. The synchronous communication is used mainly in the pre-arranged time distributed group inspection. The asynchronous communication is used during the individual inspection phase and during the group inspection phase of the distributed asynchronous mode of software inspection process.

B.5.1 Synchronous Communication

In the synchronous mode of communication, there are two ways to communicate. The first is using the chat method and the second is using the pop-up message method. The chat method can further be divided into two subcategories, namely the group chat and the private chat. The group chat involves the whole software inspection team and the configuration is done beforehand by the moderator. By selecting the group chat, the user can communicate with the other team member, provided that the other team member activate the group chat applet on his screen (see *Figure B - 5*). The user only needs to enter his name and then proceed with the message.

The private chat is similar in operation to the group chat except that users must specify their own room name. Users are prompted to enter the room name before the chat applet is displayed (see *Figure B - 6*). The room name must be agreed beforehand with the other party. The number of participants in the private chat can be more than two person. There is no authorisation procedure for the private chat. Users only need to know the chat room name. The chat room is destroyed when all participants have left the room.

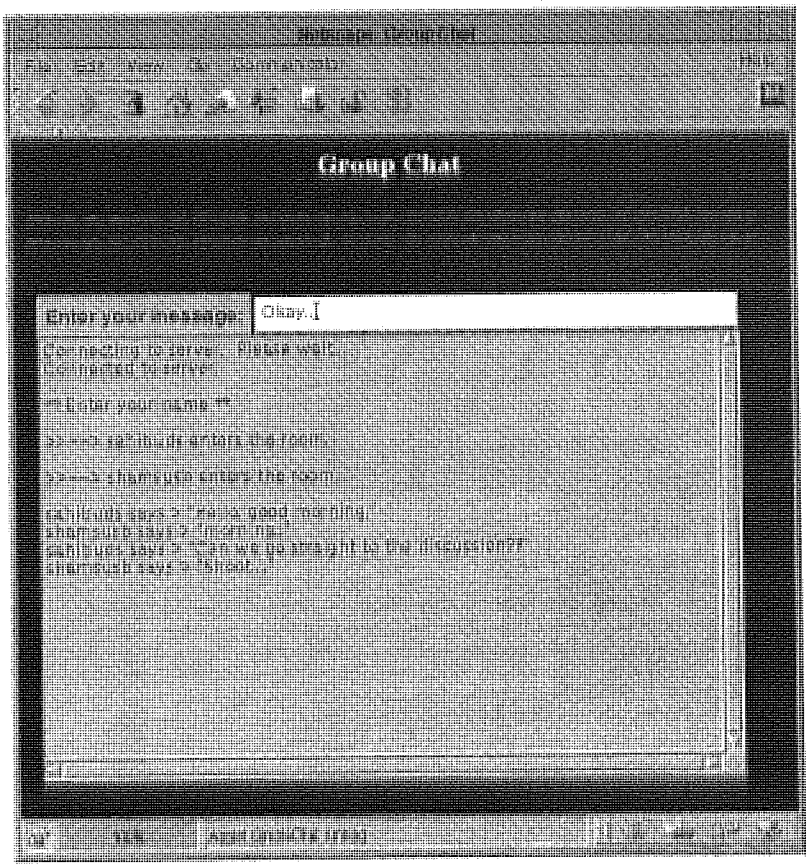

Figure B - 5 : *Group Chat Applet*

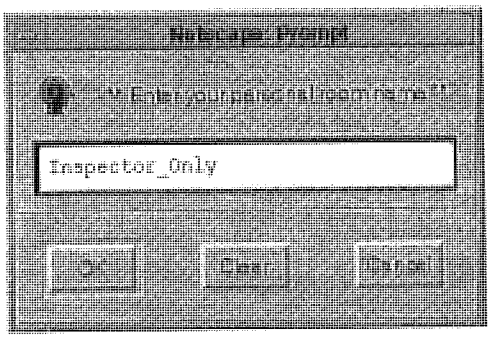

Figure B - 6 : *Specifying Room Name in Private Chat*

The pop-up message is another alternative way to communicate synchronously with the other members of the software inspection team. Similarly to the chat facility, pop-up messages can be split into two sub-categories. Pop-

up messages can be used for sending a message to the whole group or to certain individual only (see *Figure B - 7*). Users can select the whole group as the recipient. The message entered is then sent and will pop-up on the screens of the entire inspection team. If the message is private in nature, then a user can select any name from the list. The message sent will pop-up on the intended users screen only. The user needs to enter their name when they initiate the pop-up message applet.

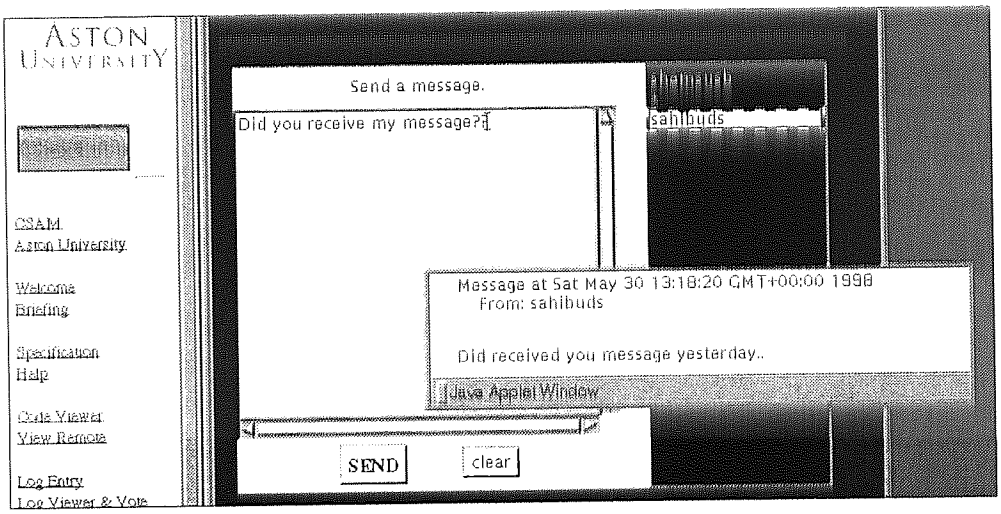

Figure B - 7 : *Pop-up Message Applet*

B.5.2 Asynchronous Communication

The asynchronous mode of communication can be divided into two categories, namely a direct communication and an indirect communication. In a direct asynchronous communication, two methods are used, namely e-mail and bulletin board. The e-mail facility is similar to other e-mail packages except that this is an enclosed system dedicated to this system only (Peek, 1991). A user needs to enter the recipient's e-mail address. The user can enter subject followed by the message. The mail is sent to the server program which will then store it in the appropriated mailbox of the recipient. In the read mail facility,

the user can read any mail sent to them (see *Figure B - 8*). They can also delete and reply to any mail that is in their mailbox. The bulletin board is another way of sending a message for the other team members. A user is required to enter their username, the subject and the message. The message is then posted on the group bulletin board and can be viewed by every member of the inspection team. The message posted is kept according to the date posted, with the latest message at the bottom. All messages posted are kept until the software inspection process has ended.

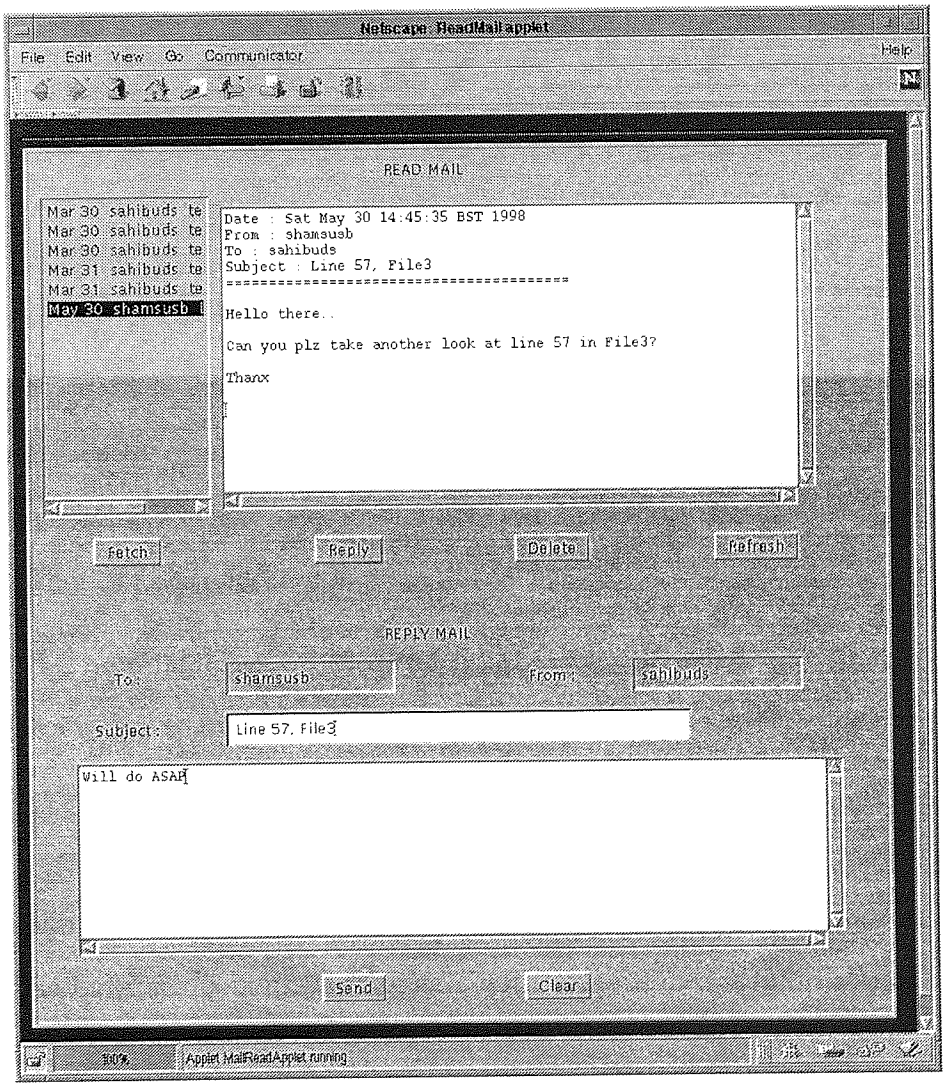

Figure B - 8 : *Read e-mail Applet*

In an indirect asynchronous communication, the log file itself can be viewed as one method of communication. Since every team member can view the log file, regardless of whether the entry is owned by them of by another member, any changes can be tracked easily. Additions to the log file, or changes to an existing entry can be considered an indirect form of asynchronous communication. Information on voting entry in the vote file for any logged item discussed can also act as a communication medium. All the changes will update the other users on the current status of the inspection process. Shared state as a communication medium was discussed in Chapter Three (Shim *et. al.*, 1997; Ellis *et. al.*, 1991).

B.6 HELP

This component is divided into three sub-components. They are help facilities on using FlexSIG, help on the syntax of the programming language, and help on the operating system usage. The last two help facilities are available when the document to be inspected is program code.

The help facilities on FlexSIG is intended to answer any question with regards to using the FlexSIG system. This sub-component explains the function of every components and operation in FlexSIG. It includes step-by-step instruction on using the system. Information on the general concept of software inspection according to FlexSIG is covered in the briefing component, and not in the help component which deals with a more detail information on how to use the system.

The following help facilities are available only for the code inspection process only: help on the syntax of the programming language and on the operating system. This facility provides information on the programming language

and web links to other sites that provide further information. The help on the operating system again depends on the implementation platform of the code inspected. The help facility provides information and links to other web sites.

B.7 METRICS AND REPORTS

Metrics and reports are generated by the moderator towards the end of the software inspection process. The reports and metrics produced in the inspection process can be used by the author and the software project team in improving the software. The reports and metrics can also be used by the software inspection process team for further improvement in the process, such as measuring the productivity of the inspectors. The quality management team also use the reports and the metrics to further improve the quality management process. Only metrics related to defects are collected. Metrics related to queries are not tabulated because it is not a defect and it is just a point the inspectors might want the author of the document to pay attention to.

B.7.1 Metrics

FlexSIG collects a number of metrics from the software inspection process. There are four types of metrics collected. All the metrics collected are from the log file entry. The first type of metric is the breakdown of the log entry according to the criticality and type of the issue (see *Table B - 2*). The metrics are divided according to the document inspected. For each document inspected, the criticality and type of defect entered is tabulated according to the inspector. C1 refers to the number of defect of criticality type 1, C2 refers to number of defect of criticality type 2, T1 refers to number of defect of type 1, so on and so forth.

The total number of log entered per inspector per document is also added up. This metric is collected before any redundancy of issue between the inspectors is removed by the moderator. The explanation on the type of criticality and the type of defects was given in Chapter Five.

As an example, for Document 1, Inspector 1 total number of log entry in the inspection process is twenty six. Out of that, seven is criticality type 1, which is a major defect, fifteen is criticality type 2, which is minor defect, and the rest is criticality type 3, which is just a warning. The twenty six defects identified is further break down into types of defects, for example three defects is from defect type 1, which is data defect, four defects is from defect type 2, which is design defect, so on and so forth. This type of metrics is useful in giving the overall picture of the inspection process.

Document 1												
	Inspector 1									Total # of Entry		
	C1	C2	C3	T1	T2	T3	T4	T5	T6	T7	T8	T9
	Inspector 2									Total # of Entry		
	C1	C2	C3	T1	T2	T3	T4	T5	T6	T7	T8	T9
Document 2												
	Inspector 1									Total # of Entry		
	C1	C2	C3	T1	T2	T3	T4	T5	T6	T7	T8	T9
:	:	:	:	:	:	:	:	:	:	:	:	:

Table B - 2 : *The First Type of Metrics Collected*

The second type of metrics collected is related to type one. This metric is a collection of each inspectors scrutiny on all the documents (see *Table B - 3*). The C1 for the first inspector is the total defects of criticality 1 for all the

documents inspected. The T1 for inspector one is the total defects of type 1 for all the documents inspected. The same holds true for the rest of the inspectors. The main reason why the first two types of metrics are collected before any duplication is removed is so that the data will reflect the actual number of defects identified by each inspectors. This information offers information on the productivity of each inspectors in the software inspection process and should reflect the level of accomplishment of this particular software inspection process.

Inspector 1									Total # of Issue		
C1	C2	C3	T1	T2	T3	T4	T5	T6	T7	T8	T9
Inspector 2									Total # of Issue		
C1	C2	C3	T1	T2	T3	T4	T5	T6	T7	T8	T9
:	:	:	:	:	:	:	:	:	:	:	:

Table B - 3 : *Tabulation of Metrics Based on Inspectors*

The third type of metrics is a summary of the data collected according to the document inspected. For each document, each type of metric from each inspector is tallied. For example, C1 for document one is the total number of defects of criticality one found by every inspectors in document one, C1 for document two is the total number of defects of criticality one found by every inspector in document two (see *Table B - 4*). The metrics of this type are collected after any duplication of issues found is removed by the moderator. This metrics reflects the state of the document, unlike the first two types of the metrics mentioned before.

Document 1									Total # of Issue		
C1	C2	C3	T1	T2	T3	T4	T5	T6	T7	T8	T9
Document 2									Total # of Issue		
C1	C2	C3	T1	T2	T3	T4	T5	T6	T7	T8	T9
Document *n*									Total # of Issue		
:	:	:	:	:	:	:	:	:	:	:	:

Table B - 4 : *Metrics According to Document*

The fourth type of metric is the tabulation of type three. This reflects the total number of issue found in the whole software inspection process. In this type, C1 means the total number of issue of criticality one for every document inspected in the software inspection process. Total number of issues here means the total number of issues collected for every document during the inspection process.

B.7.2 Reports

Reports produced in FlexSIG are based on the log file and the metrics collected. Basically, there are three types of reports. The first category is for the author of the document, the second is for the software inspection process management, and the third category is for the quality management team.

The report for the author is for the purpose of correcting the document based on the defects found. The report contains defects and queries logged by the inspector and the metrics collected. The log file report is comprised of the document name, the software project and the inspection process number, and details which include the log number, the criticality and the type of the issue, comment, and suggestion. A separate report is compiled for different docu-

ments inspected. The metrics that are included in a report to the author are from the third category (see *Table B - 4*).

The report for the software inspection process management is for the purpose of improving the software inspection process in the future. The reports of this type include only the metrics explained in the previous section. Only metrics of type one and two are included in this report (see *Table B - 2* and *Table B - 3*).

The quality management report is intended to be used in the total process improvement. The report encompass the log file and all the metrics produced. The log file report is similar to the one produced for the author. The metrics report include all four type of metrics collected.

B.8 ACCESS CONTROL

Access control was explained in detail in *Section 6.4.5*. User authorisation is handled by the login applet which is loaded to the clients machine and interpreted by the browser (see *Figure B - 9*). The clients is expected to enter their username and password into the applet and send the data back to the server to be authenticated.

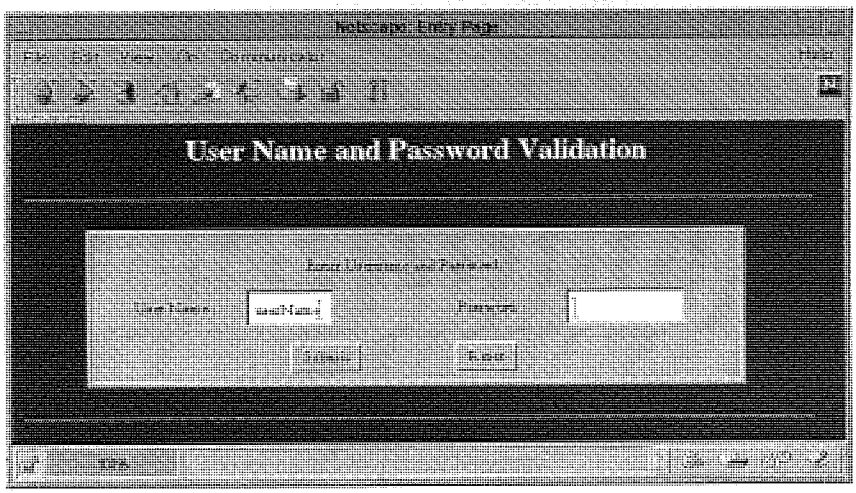

Figure B - 9 : *Login Applet*

On the maintenance aspect, the user authorisation file are managed by the moderator. The moderator has read, write, and delete rights to this file. New team member can be added and current one can be deleted. The moderator can also modify any existing entry in the file. The information that is kept in this file is the username, the password, and their role. The inspectors and the authors can only modify their own password (see *Figure B - 10).*

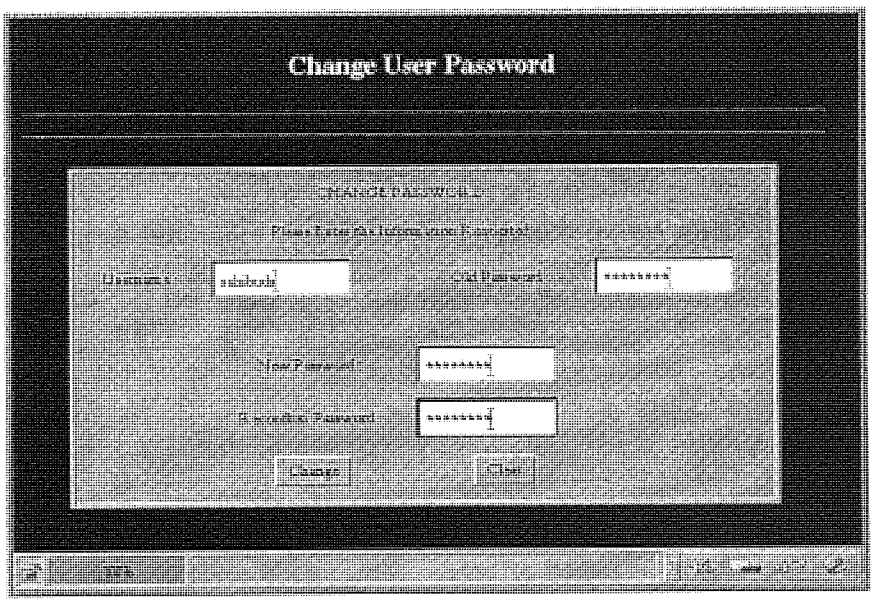

Figure B - 10 : *Changing Password Applet*

B.9 SYSTEM CONFIGURATION

System configuration is only accessible by the moderator and the system administrator. There are a number of components that need to be configured or set-up before the software inspection process takes place. These are the briefing, specification, help, documents to be inspected, log file, access control component, and inspection process identification.

To configure the briefing component, the moderator needs to create the project description, the document description, the group information, current process description, software inspection process and FlexSIG description. The briefing then needs to be made accessible to the rest of the team. The software inspection process and FlexSIG description briefing document will be the same as used in the previous software inspection process, if there was one.

The setting-up of specification briefing documents involves getting the specification from the software development team in electronic form and making it accessible to team members.

The help component on using FlexSIG is the same for every software inspection process. The moderator need only to change the help components on the programming language and operating system usage according to the document inspected. If the document inspected is not program code, then there is no need to include these two sub-components.

In preparing the document to be inspected, the moderator need to obtain the document from the software development team and make it accessible to the inspection team. The document's name is listed in a file accessible by the document viewer applet. The list is also used by the log server program to create the log file. The moderator also need to enter the process number and the

project number into a file. These information is used by the document viewer and the log file.

Lastly, the moderator is required to enter user name, password, and role for every member of the inspection team in the access control file and update the inspection process identification file.

The moderator is also expected to clear all the relevant files after the consolidation phase.

Appendix C

SERVER SOURCE CODE

C.1 EXAMPLES OF A SERVER PROGRAM CODE

```
MailUserInfoThread.java        Fri Mar 27 22:35:20 1998        1

import java.io.*;
import java.net.*;
import java.util.*;

/******
 *
 * Author : Shamsul
 * Date Created : 17/03/98
 * Date Last Modified : 17/03/98
 * Project : FlexSIG System
 * File Name : MailUserInfoThread.java
 *
 * Open a socket and wait for a packet from KunciClientApplet. When
 * request for username and password validation is received, file
 * katakunci.txt is open..
 *
 * Class :
 *   MailUserInfoThread
 * Methods :
 *   MailUserInforThread
 *   run
 *   finalize
 *   openInputFile
 *
 ******/


class MailUserInfoThread extends Thread {
  String userName[];
  String roleID = null;
  byte[] buf = new byte[1024];
  InetAddress address;
  int port;
  boolean DEBUG = true;
  char EOF = (char) -1;
  char EOLN = '\n';


 /******
  * MailUserInfoThread()
  * constructor: open a socket
  ***/

  public MailUserInfoThread (CurrentUser sM, DatagramPacket inPacket,
    DatagramSocket inSocket) {
    sharedMethod = sM;
    packet = inPacket;
    socket = inSocket;
  }


 /*****
  * run()
  * main section of the thread: wait for a packet, open a file and send info
  ***/

  public void run() {

    String returnVal = null;

    address = packet.getAddress();
    port = packet.getPort();
    String received = new String (packet.getData(), 0);
    if (DEBUG) {
      System.out.println("client port : " + port);
      System.out.println("in run - received : " + received);
    }

//  get current user from file
    StringTokenizer tkn = new StringTokenizer (received, "|");
```

```
MailUserInfoThread.java        Fri Mar 27 22:35:20 1998        2
            String roleID = tkn.nextToken();
            String returnVal2 = sharedMethod.getUsername (roleID);
            if (DEBUG)
              System.out.println("returnVal2 : " + returnVal2);
            StringTokenizer tkn2 = new StringTokenizer (returnVal2, "|");
            String readStatus = tkn2.nextToken();

            try {

              if (DEBUG) {
                System.out.println("port : " + port);
                System.out.println("received : " + received);
              }

        //    get loaded file info
              String userInfo = getUserInfo ();

        //    send response
              if (readStatus.equals ("OK")) {
                String fromUser = tkn2.nextToken();
                returnVal = fromUser + "|";
                returnVal = returnVal.concat (userInfo);
              }
              else {
                returnVal = "NotOpenCurrentUser|";
              }
              returnVal.getBytes(0, returnVal.length(), buf, 0);
              packet = new DatagramPacket(buf, buf.length, address, port);
              if (DEBUG) {
                System.out.println("buf.length : "+ buf.length);
                System.out.println("returnVal.length : "+ returnVal.length());
                System.out.println("returnVal : "+ returnVal);
                System.out.println("Server send packet.");
              }
              socket.send(packet);

            } catch (IOException e) {
              System.err.println("IOException:  " + e);
              e.printStackTrace();
            }
          } // end run() method


          public String getUserInfo () {
            String returnVal = null;
            String to = null, from = null;

            try {
              openInputFile();
              if (upfs == null)
                returnVal = "NotOpenInput|";
              else {
                String status = getUsername();
                upfs.close();
                returnVal = status;
              }
            } catch (IOException e) {
              returnVal = "XOK|IOException occurred in server.";
            }
            return returnVal;
          }


          /*****
           * openInputFile()
           ***/

          private void openInputFile() {
```

```
MailUserInfoThread.java        Fri Mar 27 22:35:20 1998        3
            try {
              upfs = new DataInputStream(new FileInputStream("usrPasswd.dat"));
            } catch (java.io.FileNotFoundException e) {
              System.err.println("Could not open password file.." + e);
            }
          } // end openInputFile()


          /*****
           * getUsername()
           * get loaded file info
           ***/

          private String getUsername () {
            String returnValue = null;
            String readValue = null;
            int i = 0, userNum = 0;
            char c = ' ';
            StringTokenizer tkn;

            userName = new String[10];
            char roleTemp = ' ';
            String passwordTemp = null;

            i = 0;
            try {
              while ((c = upfs.readChar()) != EOF) {
                StringBuffer temp = new StringBuffer();
                while (c != EOLN) {
                  temp.append (c);
                  c = upfs.readChar();
                }
                readValue = temp.toString();
                if (DEBUG) {
                  System.out.println ("readValue : " + readValue);
                  System.out.println ("i = " + i);
                }
                roleTemp = readValue.charAt (0);
                tkn = new StringTokenizer (readValue, "|");
                String temp2 = tkn.nextToken();
                userName[i] = new String (tkn.nextToken());
                passwordTemp = new String (tkn.nextToken());

                if (DEBUG)
                  System.out.println ("username[i] : " + userName[i]);
                i++;

              }
              userNum = i;
            } catch (IOException e) {
              returnValue = "XOK|IOException occurred in server.|";
            }
            returnValue = String.valueOf (userNum);
            returnValue = returnValue.concat ("|");
            for (i = 0; i < userNum; i++) {
              returnValue = returnValue.concat (userName[i] + "|");
            }
            return returnValue;

          } // end getUsername()


          private CurrentUser sharedMethod;
          private DatagramSocket socket = null;
          private DatagramPacket packet;
          private DataInputStream upfs = null;
          private DataInputStream kfs = null;

        }
```

```
MailAddShare.java        Sun Mar 29 14:00:29 1998        1
        import java.io.*;
        import java.net.*;
        import java.util.*;

        /******
         *
         * Author : Shamsul
         * Date Created : 17/03/98
         * Date Last Modified : 17/03/98
         * Project : DCIG System
         * File Name : MailAddShare.java
         *
         * Methods shared by all threads in MailAddThread.
         *
         * Class :
         *   MailAddShare
         * Methods :
         *   addMail
         *   sendP
         *   writeMail
         *   openRandomFile
         *
         ******/

        class MailAddShare {
          private RandomAccessFile mailrf = null;
          String returnValue  = null;
          String returnVal = null;
          boolean DEBUG = true;
          char EOF = (char) -1;
          char EOLN = '\n';


          /*****
           * addMail(): process change password
           ***/

          public synchronized String addMail (String inData) {
            StringTokenizer tkn;
            String fileName = null;
            String retrnVal = null;

            tkn = new StringTokenizer (inData, "|");
            fileName = tkn.nextToken();
            fileName = fileName.concat (".mbox");
            int startPos = inData.indexOf ('|');
            inData = inData.substring (startPos + 1);

            try {
              openRandomFile(fileName);
              if (mailrf == null)
                returnVal = "NotOpenRandom";
              else {
                returnVal = writeMail (inData);
                mailrf.close();
              }
            } catch (IOException e) {
              returnVal = "IOException occurred in server.";
            }
            return returnVal;
          }


          /*****
           * sendP(): send packet back to client
           ***/

          public synchronized void sendP (DatagramPacket packet, DatagramSocket
            socket) {
```

MailAddShare.java Sun Mar 29 14:00:29 1998 2

```
        try {
          socket.send (packet);
        } catch (IOException e) {
          System.err.println ("IOException in sending packet back");
        }
      }


      /*****
       * write mail into file
       ***/

      public String writeMail (String mailString) throws java.io.IOException {
        char c;
        long filePtr = 0;
        Date dateStamp = new Date();

      //  read until hit EOF
        while ((mailrf.getFilePointer() < mailrf.length()) &&
          ((c = mailrf.readChar()) != EOF)) {
          filePtr = mailrf.getFilePointer();
        }

      //  skip until EOF
        mailrf.seek (filePtr);
        if (DEBUG) {
          System.out.println ("length : " + mailrf.length());
        }

      //  write mail at end of mbox and add new EOF
        String newMail = null;
        newMail = dateStamp.toString();
        newMail = newMail.concat ("l" + mailString.trim() );
        mailrf.writeChars (newMail);
        mailrf.writeChar (EOLN);
        mailrf.writeChar (EOF);
        if (DEBUG) {
          System.out.println ("length : " + mailrf.length());
        }
        String returnValue = "OK";

        if (DEBUG) {
          System.out.println("returnValue : " + returnValue);
        }
        return returnValue;

      } // end writeMail()


      /*****
       * open loaded code file
       **/

      private void openRandomFile (String fileName) {

        try {
          if (DEBUG) {
            System.out.println("in open filename : " + fileName);
          }
          mailrf = new RandomAccessFile (fileName, "rw");
        } catch (java.io.FileNotFoundException e) {
          System.err.println("Could not open mail file location..");
        } catch (java.io.IOException e) {
          System.err.println("IO Exception when opening mail file ..");
        }

      } // end openInputFile() method

    } // end of class
```

C.2 LIST OF SERVER PROGRAM

1. KunciServer.class
2. KunciServer2Thread.class
3. KunciValidateShare.class
4. KunciValidateThread.class
5. CodeLoader.class
6. CodeLoaderThread.class
7. LoadedCodeServer.class
8. LoadedCodeServerThread.class
9. RemoteLoadedCodeServer.class
10. RemoteLoadedCodeSvrThread.class
11. LogServer.class
12. LogEntryServer.class
13. LogServerThread.class
14. LogChgSvrThread.class
15. LogChgShare.class
16. LogChgThread.class
17. LogViewAllThread.class
18. LogViewShare.class
19. LogViewThread.class
20. ChangePwdServer.class
21. ChangePwdServerThread.class
22. ProcessChangeShare.class
23. ProcessChangeThread.class
24. MailServer.class
25. MailGetUserThread.class
26. CurrentUser.class
27. MailUserInfoThread.class
28. MailSendThread.class
29. MailAddShare.class
30. MailAddThread.class
31. MailReadThread.class
32. MailGetShare.class
33. MailGetThread.class
34. BbsServer.class
35. BbsSvrThread.class
36. BbsGetShare.class
37. BbsGetThread.class
38. MetricServer.class
39. M1Thread.class
40. CompileThread.class
41. M2Thread.class
42. ReportServer.class
43. ReportSvrTread.class
44. ConfigServer.class
45. ConfigSvrThread.class

Appendix D

CLIENT SOURCE CODE

D.1 EXAMPLE OF CLIENT PROGRAM CODE

```
MailSendApplet.java          Fri Mar 27 22:43:06 1998          1

import java.applet.*;
import java.net.*;
import java.awt.*;
import java.io.*;
import java.util.*;

/******
 *
 * Author : Shamsul
 * Date Created : 17/03/98
 * Date Last Modified : 26/03/98
 * Project : FlexSIG System
 * File Name : MailSendApplet.java
 *
 * Send mail to other user
 * Communicate with MailServer by sending socket to
 * port # 16797 and # 16798
 *
 * Class :
 *   MailSendApplet
 * Methods :
 *   init
 *   makeUI
 *   addComp
 *   insets
 *   paint
 *   getUserInfo
 *   doSend
 *   doClear
 *   handleEvent
 *
 ******/

public class MailSendApplet extends java.applet.Applet {
  Choice toChoice, fromChoice;
  InetAddress address;
  int port1 = 16797;
  int port2 = 16798;
  DatagramSocket socket1, socket2;
  DatagramPacket packet1, packet2;
  String msg1 = null, msg2 = null;
  Frame wf1, wf2, wf3;
  String roleID = null;
  boolean DEBUG = true;


 /*****
  * init() Method
  *****/

  public void init() {

//   Get host address
    String host = getCodeBase().getHost();
    try {
      address = InetAddress.getByName (host);
    } catch (UnknownHostException e) {
      System.out.println ("Couldn't get Internet address: Unknown host!");
      return;
    }

    if (getParameter ("ROLEID") != null)
      roleID = getParameter ("ROLEID");

    makeUI();

  } // end init()
```

```
MailSendApplet.java          Fri Mar 27 22:43:06 1998          2
        /*****
         * makeUI () Method
         *****/

        public void makeUI() {


          ConnectInfoMail sm = new ConnectInfoMail ("", "", "", "");

        //  Set file name entry area

          String userInfo = getUserInfo();
          if (userInfo != "XOK") {
            StringTokenizer tkn = new StringTokenizer (userInfo, "|");
            fromChoice = new Choice();
            String userName = tkn.nextToken();
            fromChoice.addItem (userName);
            toChoice = new Choice();
            int userNum = Integer.parseInt (tkn.nextToken());
            for (int i = 0; i < userNum; i++) {
              userName = tkn.nextToken();
              toChoice.addItem (userName);
            }
          }

        //  Create UI
          GridBagLayout gridBag = new GridBagLayout();
          GridBagConstraints gbc = new GridBagConstraints();
          setLayout(gridBag);

          Label h1 = new Label("SEND MAIL", Label.CENTER);
          gbc.anchor = GridBagConstraints.CENTER;
          gbc.weightx = 100;
          gbc.weighty = 50;
          addComp (h1, gridBag, gbc, 3, 0, 1, 1);

          Label l1 = new Label("To : ", Label.CENTER);
          gbc.anchor = GridBagConstraints.CENTER;
          gbc.fill = GridBagConstraints.NONE;
          gbc.weightx = 100;
          gbc.weighty = 100;
          addComp (l1, gridBag, gbc, 0, 1, 1, 1);

          toField = new TextField(sm.toIn,15);
          gbc.anchor = GridBagConstraints.WEST;
          addComp(toField, gridBag, gbc, 1, 1, 1, 1);

          toChoice.select (0);
          gbc.anchor = GridBagConstraints.CENTER;
          addComp(toChoice, gridBag, gbc, 2, 1, 1, 1);

          Label dummy = new Label ("  ");
          gbc.weightx = 100;
          gbc.weighty = 20;
          addComp (dummy, gridBag, gbc, 3, 1, 1, 1);

          Label l2 = new Label("From : ", Label.CENTER);
          gbc.anchor = GridBagConstraints.CENTER;
          addComp (l2, gridBag, gbc, 4, 1, 1, 1);

          fromField = new TextField(sm.fromIn, 15);
          gbc.anchor = GridBagConstraints.WEST;
          addComp(fromField, gridBag, gbc, 5, 1, 1, 1);
          fromField.setEditable (false);
          fromField.setText (fromChoice.getItem(0));

          fromChoice.select (0);
          gbc.anchor = GridBagConstraints.CENTER;
          addComp(fromChoice, gridBag, gbc, 6, 1, 1, 1);
```

MailSendApplet.java Fri Mar 27 22:43:06 1998 3

```
        Label l3 = new Label("Subject : ", Label.CENTER);
        gbc.anchor = GridBagConstraints.CENTER;
        gbc.weightx = 100;
        gbc.weighty = 100;
        addComp (l3, gridBag, gbc, 0, 2, 2, 1);

        subjectField = new TextField(sm.subjectIn,45);
        gbc.anchor = GridBagConstraints.WEST;
        addComp(subjectField, gridBag, gbc, 2, 2, 5, 1);

        messageArea = new TextArea (sm.messageIn, 10, 85);
        Font fcourier = new Font ("Courier", Font.PLAIN, 12);
        messageArea.setFont (fcourier);
        gbc.anchor = GridBagConstraints.WEST;
        addComp(messageArea, gridBag, gbc, 0, 3, 7, 4);

        Button change = new Button(" Send ");
        gbc.anchor = GridBagConstraints.CENTER;
        gbc.weightx = 100;
        gbc.weighty = 150;
        addComp(change, gridBag, gbc, 2, 7, 1, 1);

        Button clear = new Button(" Clear ");
        addComp(clear, gridBag, gbc, 4, 7, 1, 1);

      } // end makeUI()


    //* add component into gridbaglayout
      private void addComp (Component c, GridBagLayout gbl,
       GridBagConstraints gbc, int x, int y, int w, int h) {

        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridwidth = w;
        gbc.gridheight = h;
        gbl.setConstraints (c, gbc);
        add (c);
      } // end addComp()


    //*
      public Insets insets() {

        return new Insets(4,4,5,5);
      } // end insets()


    //*
      public void paint (Graphics g) {
        Dimension d = size();
        Color bg = getBackground();

        g.setColor(bg);
        g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
        g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
      } // end paint()


     /*****
      * getUserInfo method
      *****/

      public String getUserInfo() {
        byte[] sendBuf1 = new byte [1024];

    //  open socket
        try {
```

```
            socket1 = new DatagramSocket();
          } catch (SocketException e) {
            System.out.println ("Couldn't create new DatagramSocket");
            return "Error";
          }

        // send request for username to server
          String initiateRequest = roleID + "I";
          initiateRequest.getBytes (0, initiateRequest.length(), sendBuf1, 0);
          packet1 = new DatagramPacket (sendBuf1, 1024, address, port1);
          try {
            if (DEBUG) {
              System.out.println (" in getFileInfo address : " + address);
              System.out.println ("port : " + port1);
            }
            socket1.send(packet1);
            if (DEBUG)
               System.out.println("Applet sent packet.");
          } catch (IOException e) {
            System.out.println("Applet socket.send failed:");
            e.printStackTrace();
            return "Socket.send failed";
          }

        // check file info from server
          packet1 = new DatagramPacket (sendBuf1, 1024);
          try {
            if (DEBUG)
              System.out.println("Applet about to call socket.receive().");
            socket1.receive(packet1);
            if (DEBUG)
              System.out.println("Applet returned from socket.receive().");
          } catch (IOException e) {
            System.out.println("Applet socket.receive failed:");
            e.printStackTrace();
            return "Socket.receive failed";
          }
          String receivedD = new String(packet1.getData(), 0);
          if (DEBUG)
            System.out.println("File info : " + receivedD);
          StringTokenizer tkn = new StringTokenizer (receivedD, "I");
          String status = tkn.nextToken();

        // problem reading username
          if (status.equals ("XOK")) {
            msg1 = "Error!";
            msg2 = "Problem in reading username!";
            wf1 = new WarningFrame (msg1, msg2);
            wf1.resize (250,150);
            wf1.setBackground (Color.red);
            wf1.show();
            doClear();
            receivedD = "XOK";
          }

          return receivedD;
        }// end getUserInfo()


        /*****
         * doSendM() Method
         *****/

        public void doSendM() {

          ConnectInfoMail mssg = new ConnectInfoMail (toField.getText(),
            fromField.getText(), subjectField.getText(), messageArea.getText());

          MailAppletShare sendM = new MailAppletShare();
          sendM.doSend (mssg, address);
```

```
MailSendApplet.java          Fri Mar 27 22:43:06 1998          5
            doClear();
          }


        /*****
         * doClear() Method
         *****/

         public void doClear () {

           toField.setText ("");
           subjectField.setText ("");
           messageArea.setText ("");
         } // end doClear()


        /*****
         * handleEvent() Method
         *****/

         public boolean handleEvent (Event e) {
           switch (e.id) {
           case Event.ACTION_EVENT:
             {
               if (e.target instanceof TextField || e.arg.equals (" Send ")) {
                 doSendM();
                 return true;
               }
               if (e.arg.equals (" Clear ")) {
                 doClear();
                 return true;
               }
               if (e.target == toChoice) {
                 toField.setText (toChoice.getSelectedItem());
                 return true;
               }
             }
             default :
               return false;
           }
         } // end handleEvent()

         private TextField toField;
         private TextField fromField;
         private TextField subjectField;
         private TextArea messageArea;

        } // end MailSendApplet
```

D.2 LIST OF CLIENT PROGRAM

1. ValidateUserApplet.class
2. ConnectInfo.class
3. WarningFrame.class
4. CodeLoader.class
5. RemoteCodeLoader.class
6. PopMessage.class
7. BbsApplet.class
8. ChangePwdApplet.class
9. ConnectInfoChg.class
10. AddLogApplet.class
11. ChangeLogApplet.class
12. ViewVoteLogApplet.class
13. ConnectInfo2.class
14. MailSendApplet.class
15. ConnectInfoMail.class
16. MailAppletShare.class
17. MailReadApplet.class
18. SimpleChat.class
19. MetricConpileApplet.class
20. ReportApplet.class
21. ConfigApplet.class

D.3 LIST OF HTML DOCUMENT

1. ValidateUserPage.html
2. DCIGPagex.html
3. ColDCIGx.html
4. FootDCIG.html
5. Welcome.html
6. Briefing.html
7. Specification.html
8. Help.html
9. ViewerPagex.html
10. ViewRemote.html
11. LogEntryx.html
12. LogChangex.html
13. LogViewx.html
14. Mailx.html
15. Chat.html
16. Bbs.html
17. PopMessage.html
18. Calendar.html
19. ChangePwdPage.html

20. LogCompile.html
21. Configure.html
22. AddLogx.html
23. ModifyLogx.html
24. RmLogx.html
25. ViewLogx.html
26. MailSendx.html
27. MailReadx.html
28. GroupChat.html
29. PrsnlChat.html
30. Metric1.html
31. Compile.html
32. Metric2.html
33. Report.html
34. CfgPwdFile.html
35. CfgInspectDocFile.html

Appendix E

PHASE ONE QUESTIONNAIRE

A Survey on Inspection and Review Process

1. Name :

_____

2. Gender :

____ Male ____ Female

3. Age :

____ 20 - 24 ____ 25 - 29 ____ 30 -34 ____ 34

4. Occupation :

_____

5. Employer :

_____

6. Employers Address :

_____

_____

_____

7. Office Tel. No. :

_____

8. E-mail Address :

_____

9. Homepage Address :

_____

10. Qualification :

_____

_____

11. How many years in IT/Computer related fields :

____ 1 - 2 ____ 3 - 4 ____ 5 - 6 ____ 7 -

12. Which area of IT/Computer do you have experience in :

____ Programming ____ System Analysis

____ Network Administrator ____ S/W Project Manager

____ IS/EDP Management ____ Quality Management

Other (please specify) :

_____

_____

13. How many years in Software Development area :

____ 1 - 2 ____ 3 - 4 ____ 5 - 6 ____ 7 -

14. How many software projects have you been involved in :

____ 1 - 2 ____ 3 - 4 ____ 5 - 6 ____ 7 -

15. Do you have any experience or training in Software Quality process?

____ Yes ____ No

16. If Yes, how many years in Software Quality process?

____ 1 - 2 ____ 3 - 4 ____ 5 - 6 ____ 7 -

17. Have you used any inspection or review method in software quality process?

____ Yes ____ No

18. If Yes, what type of inspection or review method have you have been using so far?

____ Software Review ____ Software Audit

____ Formal Technical Review ____ Software Inspection

____ Code Inspection ____ Walkthrough

Other (please specify) :

_____

_____

19. Do you have any reservation or difficulty in using any inspection or review process?

____ Yes ____ No

20. If Yes, can you explain why?

_____

_____

_____

_____

21. How likely you will be using any inspection or review process in the future?

(Most Likely) 1 2 3 4(Most Unlikely)

22. Do you have any reservation or difficulty in using the formal code inspection process?

____ Yes ____ No

23. If Yes, can you explain why?

_____

_____

_____

_____

24. What sort of improvement would you like to see in the inspection or review process?

_____

_____

_____

_____

25. Do you think that using CASE tools will help you in the inspection process?

____ Yes ____ No

26. What kind of feature would you like to see in the CASE tools?

_____

_____

_____

_____

27. Do you think that using distributed CASE tools for inspection process will be a better choice?

____ Yes ____ No

28. Why?

_____

_____

_____

_____

29. In a software development project, how do you communicate with other project members?

____ E-mail ____ Formal meeting

____ Memo ____ Chat system

____ Telephone

Other (please specify)

_____

_____

30. How do you distribute project documents among members?

____ E-mail ____ Fax

____ Paper ____ HTML

Other (please specify) :

_____

_____

31. How easy would it be to get a reference for the requirement and design specification and other document when you run into trouble in the coding and testing stage?

(Easy) 1 2 3 4 (Difficult)

32. How easy is it to conduct a formal meeting with project members?

(Easy) 1 2 3 4 (Difficult)

Appendix F

FLEXSIG EVALUATION GUIDELINE

Evaluation of Flexible Software Inspection Groupware

Name : _____

1. Start Flexible Software Inspection Groupware (FlexSIG)
   a. Start web browser
   b. Type http://www.cs.aston.ac.uk/~sahibuds/FlexSIG
   c. Enter Username
   d. Enter Password
   e. Click Submit

2. Select 'Briefing' from menu
   a. Select code inspection process description
   b. Select FlexSIG description
   c. Select inspection group information
   d. Select overall project description
   e. Select current code inspected description

3. Select 'Specification' from menu
   a. Select requirement specification
   b. Select design specification

4. Select 'Help' from menu
   a. Select help on FlexSIG
   b. Select help on inspection process
   c. Select help on programming language
   d. Select help on operating system

Asynchronous Evaluation

5. Select 'Code Viewer' from menu
   a. Enter Username
   b. Select the file to view
   c. Fetch the file

6. Select 'Log Entry' from menu
   a. Enter Username
   b. Enter Filename
   c. Create a comment and enter into log
   d. Log another comment

7. Select 'E-mail' from menu
   a. Send e-mail to yourself
   b. Send e-mail to your colleague
   c. Open mail box and read message sent previously in step 7a

Note : In asynchronous mode, communication is handled by e-mail, log view and rely heavily on HTML documents. Inspectors can go back and forth to briefing, specification and help documents when logging comment on current code inspected.

Synchronous Evaluation

8. Select 'Remote Viewer' from menu
   a. Select file which is currently viewed by other group member
   b. Fetch the file
   c. Refresh the applet in case other users change to other file
   d. Fetch the new file

9. Select 'Chat' from menu
   a. Select group chat
   b. Enter your name
   c. Chat with your colleague

10. Select 'Pop Message' from menu
   a. Enter your name
   b. Pop a message to everybody in the group
   c. Pop a message to one of your colleagues only

11. Select 'Chat' from menu
   a. Select personal chat
   b. Enter your name
   c. Enter a pre-arranged room name
   d. Chat one-to-one with your colleague

Note : In synchronous mode, communication is handled by group chat, personal chat, pop message to group or individual and also by remote viewer. Inspectors can still go back and forth to the briefing, specification and help material.

Miscellaneous

12. Select 'Change Password' from menu
    a. Change your password

13. Select 'Calendar' from menu

14. Quit from browser

End of evaluation. Thank you.

Appendix G

PHASE THREE QUESTIONNAIRE

A Survey on Inspection and Review Process: Phase III

1. Name :

_____

2. Gender :

____ Male ____ Female

3. Age :

____ 20 - 24 ____ 25 - 29 ____ 30 -34 ____ 34-

4. Occupation :

_____

5. Employer :

_____

6. Employers Address :

_____

_____

_____

7. Office Tel. No. :

_____

8. E-mail Address :

_____

9. Personal Homepage Address :

_____

10. Qualification :

_____

_____

_____

11. How many years in IT/Computer related fields :

____ 1 - 2 ____ 3 - 4 ____ 5 - 6 ____ 7 -

12. Which area of IT/Computer do you have experience in :

____ Programming ____ System Analysis

____ Network Administrator ____ S/W Project Manager

____ IS/EDP Management ____ S/W Quality Management

Other (please specify) :

_____

_____

13. How many years in Software Development area :

____ 1 - 2 ____ 3 - 4 ____ 5 - 6 ____ 7 -

14. How many software projects have you been involved in :

____ 1 - 2 ____ 3 - 4 ____ 5 - 6 ____ 7 -

15. Do you have any training in Software Quality process?

____ Yes ____ No

16. How many years in Software Quality process?

____ 1 - 2 ____ 3 - 4 ____ 5 - 6 ____ 7 -

17. Have you used any inspection or review method in software quality process?

____ Yes ____ No

18. If Yes, what type of inspection or review method have you have been using so far?

____ Software Review/Inspection ____ Software Audit

____ Formal Technical Review ____ Code Inspection

____ Walkthrough

Other (please specify) :

_____

_____

_____

19. Is the briefing material concept sufficient in replacing the kick-off meeting in asynchronous meeting?

(Sufficient) 1 2 3 4 (Not Sufficient)

20. Can you explain why?

_____

_____

_____

21. Is the specification document via the HTML a better option than accessing the paper version?

(Better) 1 2 3 4 (Worst)

22. Why?

_____

_____

_____

23. Is the help facilities concept sufficient for the inspection process?

(Sufficient) 1 2 3 4 (Not Sufficient)

24. Can you explain why?

_____

_____

_____

25. Do you think that the information collected in log entry is sufficient?

(Sufficient) 1 2 3 4 (Not Sufficient)

26. Why?

_____

_____

_____

27. Do you think that FlexSIG is suitable for asynchronous mode of process?

(Suitable) 1 2 3 4 (Not Suitable)

28. Do you have any suggestion on what type of communication facilities should be added in order to improve the asynchronous mode of process?

_____

_____

_____

_____

29. Is the briefing material a good complement to the kick-off meeting in a synchronous meeting?

(Agree) 1 2 3 4 (Not Agree)

30. Does remote viewer help you in accessing the file viewed by other group members quickly?

(Helpful) 1 2 3 4 (Not Helpful)

31. How easy do you find communicating with other group members synchronously using chat?

(Easy) 1 2 3 4 (Difficult)

32. How easy do you find communicating with other group members synchronously using pop message?

(Easy) 1 2 3 4 (Difficult)

33. Do you think that FlexSIG is suitable for synchronous mode of process?

(Suitable) 1 2 3 4 (Not Suitable)

34. Do you have any suggestion on what type of communication facilities should be added in order to improve the synchronous mode of process?

_____

_____

_____

35. Do you think that FlexSIG is flexible enough for asynchronous mode of process and synchronous mode of process?

(Suitable) 1 2 3 4 (Not Suitable)

36. Is FlexSIG a better choice than manual formal review meeting in conducting an inspection process?

(Better) 1 2 3 4 (Worst)

37. Is FlexSIG a better choice than a code inspection system that support one mode of process only (support asynchronous OR synchronous ONLY)?

(Better) 1 2 3 4 (Worst)

38. Do you think that by implementing FlexSIG on the Web will provide better flexibility in terms of multi-platform support?

(Better) 1 2 3 4 (Worst)

39. Any other comment on FlexSIG or inspection process in general?

Appendix H

STATISTICAL ANALYSIS OF PHASE ONE'S QUESTIONNAIRE

H.1 PHASE ONE DATA COLLECTION

Data collection in phase one concerns information about subject practice and perception towards software inspection process. Information gathered in this phase, along with the literature review, serves as a basis for the model developed.

H.1.1 Outline of Phase One

In phase one's data collection, a questionnaire method was put to use. A group of subjects was given a questionnaire to complete with questions ranging from personal background to software inspection practice. The total number of subjects participating in this phase was twenty one people. The participants in phase one are a group of students from the Centre of Advanced Software Engineering, Kuala Lumpur, Malaysia. This centre is a joint venture between Faculty of Computer Science and Information System from Malaysia University of Technology, and Campus Thomson from France. The students involved in phase one are from the Master of Software Engineering program. The majority

of the masters students have experience in software development in a professional working environment.

The data collected was analysed using basic statistical tools. The findings serve the development of the FlexSIG model by providing additional data to the construction of the model.

H.1.2 Questionnaire

There were thirty two questions in the questionnaire given to the subject. Ten questions relate to the personal background of the participant. Examples from this type of question is the participant's name, occupation, and academic background. Six questions related to the subject's experience in software development and software quality. Examples from this category of question were how many years of experience in computer related fields, how many years of experience in software quality, and how many software projects they have been involved in. The final sixteen questions dealt with the participant's practice and expectations in the software inspection process. Examples from this class of question are what type of inspection process they have used, do they have any difficulty in using any inspection process, and how they communicate with other team members in a software development project. The complete questionnaire can be found in Appendix D. A response is considered valid if it follow the instruction given in the questionnaire. A blank response is considered as an invalid response. In the questionnaire, some of the question use a scale of 1 to 4. The value of 1 is a positive response to the question, and meant most likely or less difficult, and the value 4 is a negative response, which meant less likely or most difficult.

H.2 ANALYSIS OF PHASE ONE'S QUESTIONNAIRE

Twenty one subject participated in this phase. The majority of the subject is male with only 14.3 percent female. 76.2 percent of the participants were below thirty years of age. The occupation range from tutor, lecturer, programmer, system analyst, software engineer, research officer, to electrical and weapons engineer. Their employers range from a university, a semiconductor company, an airline, a telecommunication company, a computer firm, to the navy. Most of them have a degree in computer science. 90 percent of the valid respondent have a degree in computer science or related fields and 10 percent have a degree in engineering.

52.4 percent have more than three years experience working in information technology. Almost all of them, 95.2 percent, have experience in programming. 52.4 percent have experience in system analysis. But only 19.0 percent have experience in software project management and 9.5 percent have experience in quality management. In the questionnaire, the participants can choose more than one category of experience. The complete listing is in *Figure H - 1*. 30 percent of the subjects have more than three years working experience in software development. Almost half of the respondents, 47.6 percent, have had involvement in more than three software projects in their software development experience. The response shows that a large number of the participants have experience in programming and software development.

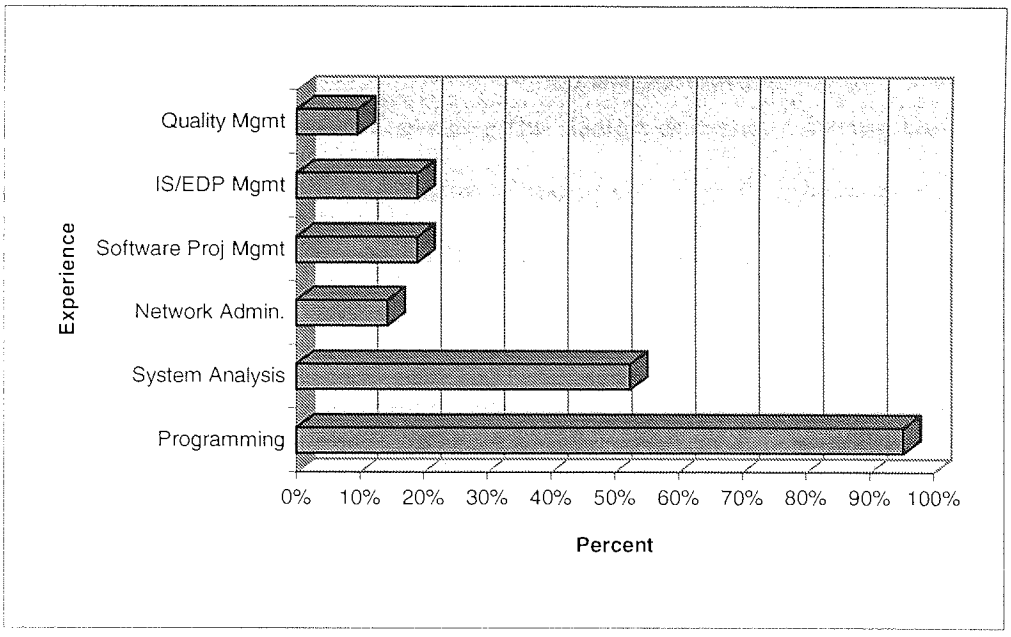

Figure H - 1 : *Respondent IT Experience*

H.2.1 Software Inspection Practice

Only 38.1 percent said that they received any training or experience in software quality. 90 percent of the valid respondents have 2 years or less experience in software quality. Surprisingly, 71.4 percent of the sample says that they have used some form of inspection or review techniques in software development. This indicates that half of the respondents here use the review or inspection technique without any training. The most common technique used by the respondent was the walkthrough with 57.1 percent saying they have used it. 42.9 percent of the sample said they had used software inspection. Only 4.8 percent said that they had used software audit.

50 percent of the valid respondents stated that they have had difficulties in using the inspection and review process. Among the reasons stated were difficulties in categorising the errors and defects, no training given on how to conduct the process, and not being familiar with the inspection process itself. Other reasons given were that the inspection process is time consuming, the

process is a tedious job, there are quite a number of documents to handle in the process, and also difficulty in accessing the design document during the inspection process. On the specific question of using the formal code inspection, 21.4 percent stated that they had had difficulty. The reasons given were not enough experience and exposure to the process and the tedious nature of the process.

More than half of the respondents said that it was difficult to get hold of the requirement and design document for reference when they ran into trouble in the coding and testing stage. On the scale of one to four, where one is easy and four is difficult, 19.0 percent choose one and two, 47.6 percent choose 3, and 33.3 percent choose 4 (see *Figure H - 2*). From *Table H - 1,* it can safely be stated that the response shows it is difficult to get hold of the design and requirement document during the inspection or review process.

Mean	Standard Error	Standard Deviation
3.10	0.18	0.83

Table H - 1 : *Mean for Level of Difficulty in Accessing Specification*

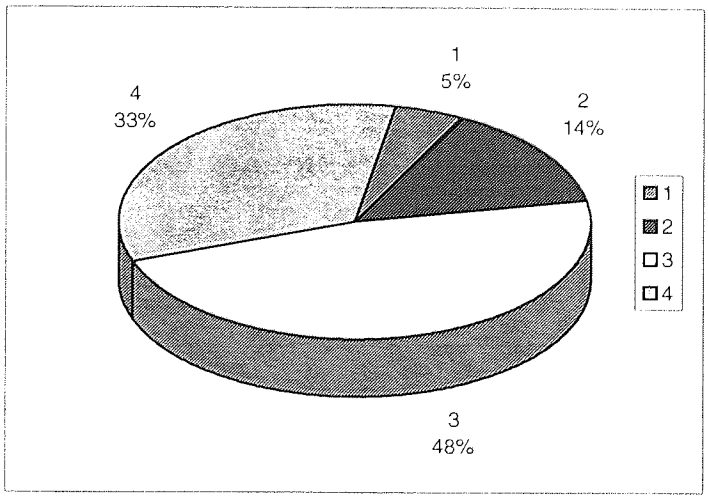

Figure H - 2 : *Level of Difficulty in Accessing Specification*

On the question of how easy it was to conduct a formal meeting with project members, 71.4 percent choose 1 and 2 (see *Figure H - 3*). The statistic (see *Table H - 2*) shows that the difficulty level of conducting a meeting, although the easier side, is near the middle of the scale.

Mean	Standard Error	Standard Deviation
2.29	0.17	0.78

Table H - 2 : *Mean for Level of Difficulty to Conduct a Meeting*

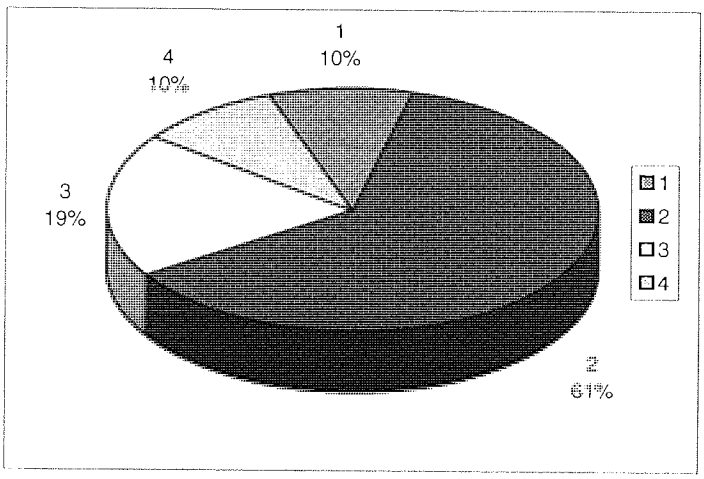

Figure H - 3 : *Level of Difficulty to Conduct a Meeting*

H.2.2 Software Development Practice

The most common method of communication during a software development project is a face-to-face meeting. 90.5 percent of the respondents use meetings as a mode of communication (see *Figure H - 4*). The next favourite method is telephone with 76.2 percent, followed by e-mail and memos with 66.7 percent and 61.9 percent respectively. Electronic chat is the least used method with only 23.8 percent of the respondents using it. The respondents also mentioned

other methods of communication such as informal conversation and informal meeting.

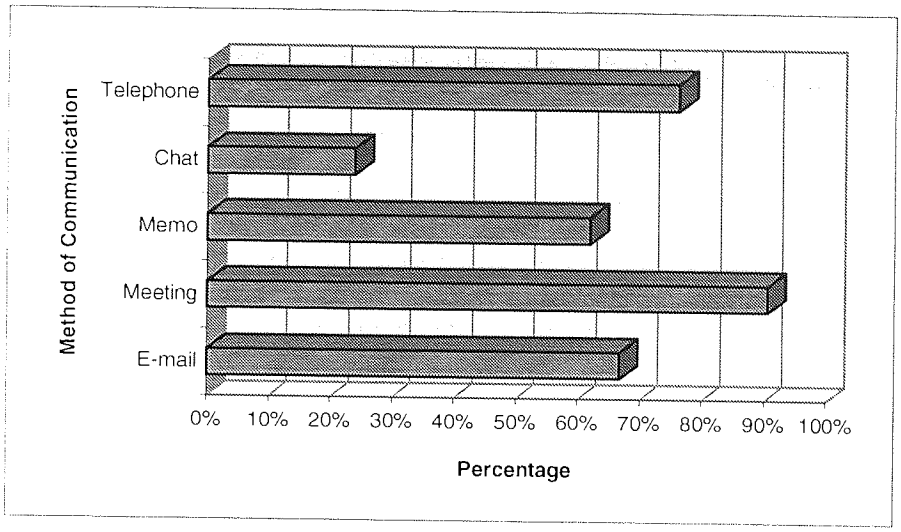

Figure H - 4 : *Method of Communication during Software Development*

On the method of document distribution among project members, 90.5 percent of the respondents said that they use paper as a medium (see *Figure H - 5*). 66.7 percent use e-mail. 15.3 percent use fax to distribute project documents. None of the respondent use HTML as a medium. FTP was also stated as one of the method used by the respondents. The statistics reveal that paper is still the overwhelming medium of project document distribution among team members.

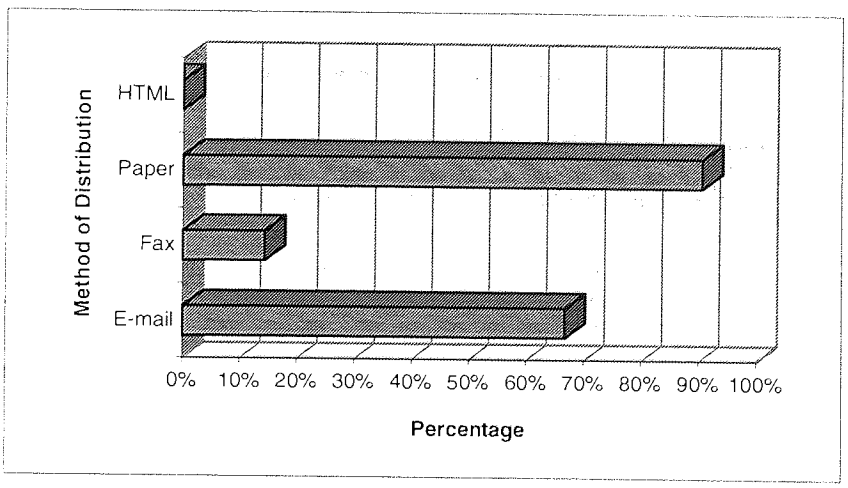

Figure H - 5 : *Method of Document Distribution*

H.2.3 Expectation of the Inspection Process

One answer given when asked on what kind of improvement they would like in the inspection process is automation of the process. They would also like to see the inspection process conforming to some form of standard. 94.4 percent of the valid response agrees that a CASE tool for the inspection process is helpful. Among the features that they would like to see is direct access to specification documents such as process flow and relationships. The respondents said that the tool should support specification, design, and code inspection. Some even suggested support for testing, such as automatic test generator. One respondent stressed the importance of a graphical user interface and good report generation features.

72.7 percent of the valid respondent stated that a distributed CASE tool for inspection process is a better choice than a stand-alone tool. Among the reasons given by them is the advantages of client-server architecture. They also stated that a distributed tool will force the issue of standardised methods.

On the question of the likelihood of using a software inspection process in the future, from the scale of one to four, 40.0 percent choose 1, 35.0 percent choose 2, 20.0 percent choose 3, and 5.0 percent choose 4 (see *Figure H - 6*). The indication from *Table H - 3* is very clear that the respondents are positive towards using a software inspection process in the future.

Mean	Standard Error	Standard Deviation
1.90	0.20	0.91

Table H - 3 : *Mean for The Likelihood of Using Inspection in the Future*

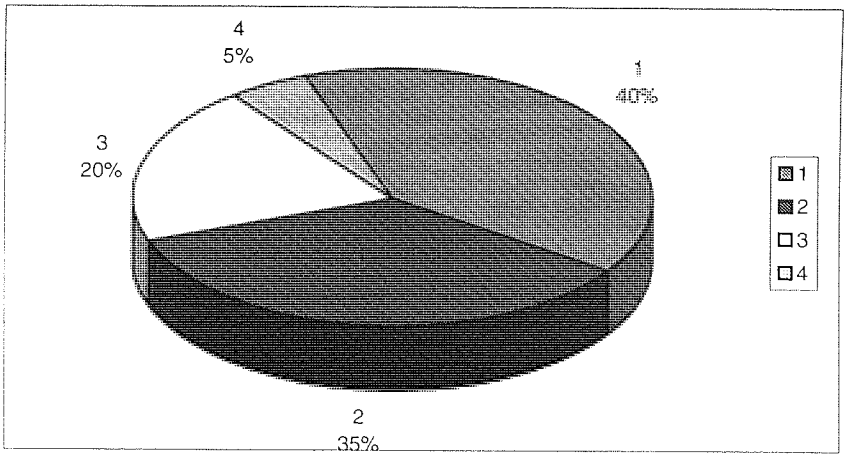

Figure H - 6 : *The Likelihood of Using Inspection in the Future*

H.2.4 Relationship Between the Variables

Further analysis was done to analyse the relationship between certain variables. Crosstabulation and boxplots was employed in order to achieve this. Crosstabulation was used first to identify any relationship between two variables. Boxplot graph was put to use later on to visualised the relationship.

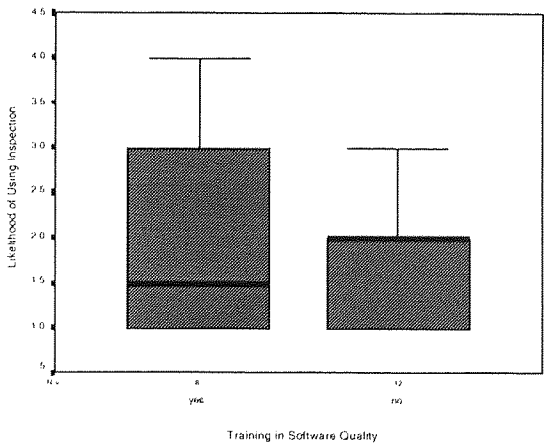

Figure H - 7 : *Training in Software Quality versus the Likelihood of Using Inspection in the Future*

Examining the relationship between the training in software quality and the likelihood of using software inspection in the future, *Figure H - 7* reveals that 50 percent of the sample with training are more likely to use inspec-

tion in the future than the 50 percent of the sample with no training. But the 25th percentile for 'yes' is higher than 'no', at 3.0 and 2.0 respectively. The 10th percentile for 'yes', at 4.0, is also higher than 'no', at 3.0. Although 50 percent of 'yes' is more likely to use inspection than 'no', the reverse is true for the 75 percent and 90 percent of the sample. Therefore, this implies that the exposure to training in software quality didn't seem to contribute more than no exposure to software quality training in the likelihood of using inspection in the future.

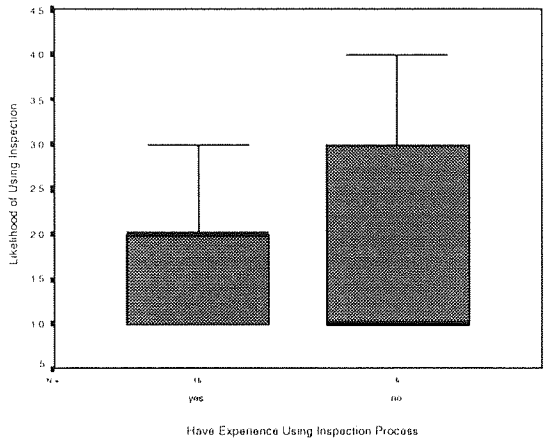

Figure H - 8 : *Experience Using Inspection Process versus the Likelihood of Using Inspection in the Future*

In the analysis of experience using an inspection process versus the likelihood of using inspection, the 50th percentile for 'yes' is 2.0 and for 'no' is 1.0 (see *Figure H - 8*). This means that 50 percent of the sample who have no experience in using inspection are more likely to use inspection in the future than those with experience. But for the 25th percentile, value for 'yes' is 2.0 and value for 'no' is 3.0. This means that for 75 percent of the sample who say 'yes' is more likely to use inspection than 75 percent of the sample who say 'no'. Except for the 50th percentile, the 25th and the 10th percentile for 'yes' is lower than 'no', also none of the sample from 'yes' choose value 4 as the likelihood of using inspection in the future. Thus, these indicate that experience

contributes positively towards the likelihood of using an inspection process in the future.

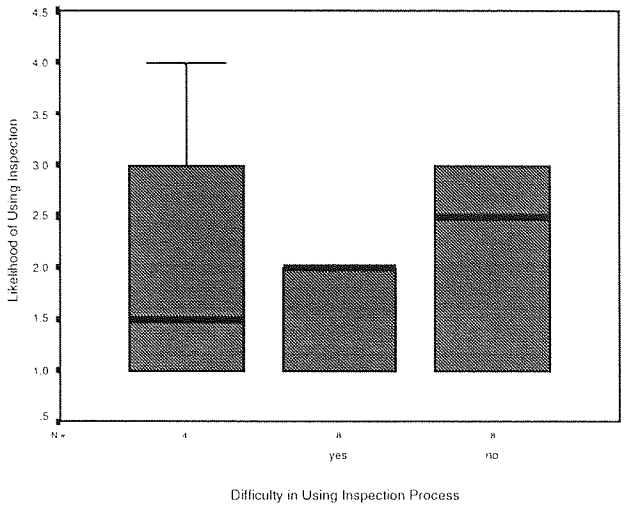

Figure H - 9 : *Difficulty in Using Inspection Process versus the Likelihood of Using Inspection in the Future*

The 50th percentile for the sample that have difficulties in using any inspection or review process is 2.0 and for those who have no difficulties is 2.5 (see *Figure H - 9*). The 25th and the 10th percentile for 'yes' is also lower than 'no'. The statistics indicate that experience of having difficulties in using any inspection or review process does not discourage the likelihood of using inspection in the future.

On the relationship of the difficulty in using a formal code inspection process versus the prospect of using an inspection process in the future, at 50th percentile, the sample with difficulties is less likely to use inspection in the future than the sample with no difficulty (see *Figure H - 10*). The 25th and the 10th percentile for 'yes' and 'no' are the same, at value 2.5 and 3 respectively. This suggests that having experienced difficulty in using formal code inspection does contribute negatively towards the usage of an inspection process in the future.

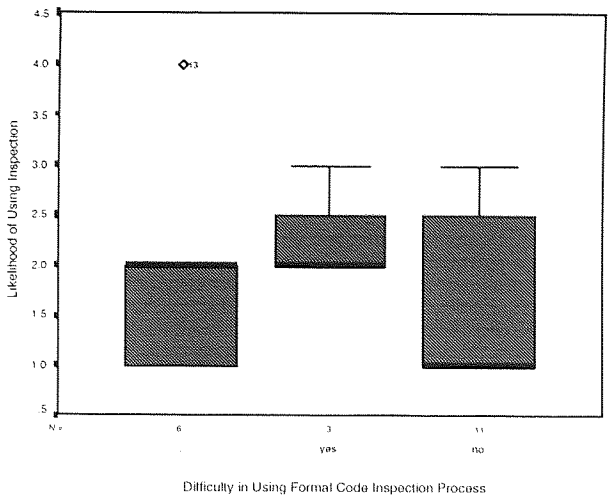

Figure H - 10 : *Difficulty in Using Formal Code Inspection Process versus the Likelihood of Using Inspection in the Future*

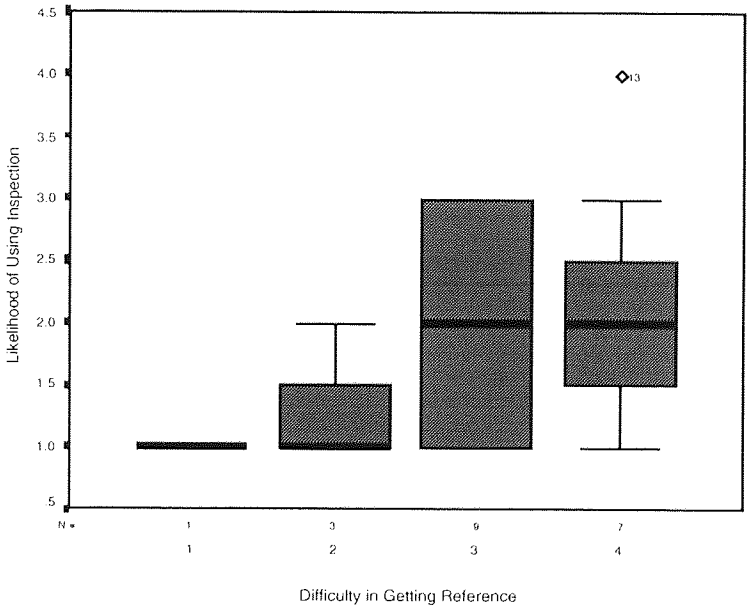

Figure H - 11 : *Difficulty in Getting Reference versus the Likelihood of Using Inspection in the Future*

The boxplot graph of difficulty in getting reference document versus the likelihood of using inspection in the future reveals a very close relationship between the two variables (see *Figure H - 11*). At the 50th percentile, the sample that choose value 1 and 2 as the level of difficulty in getting reference document, is more likely to use inspection in the future than the sample that choose

value 3 and 4. Also, at the 25th percentile, the sample that choose value 1 as the level of difficulty in getting reference is more likely to use inspection in the future than the sample that choose value 2, which in turn is more likely to use inspection than the sample that choose value 3 and 4. The same holds true for the 10th percentile. Thus, it can be concluded that the sample with experience of less difficulty in getting reference document during coding is more likely to use inspection processes in the future than the sample that experienced more difficulty.

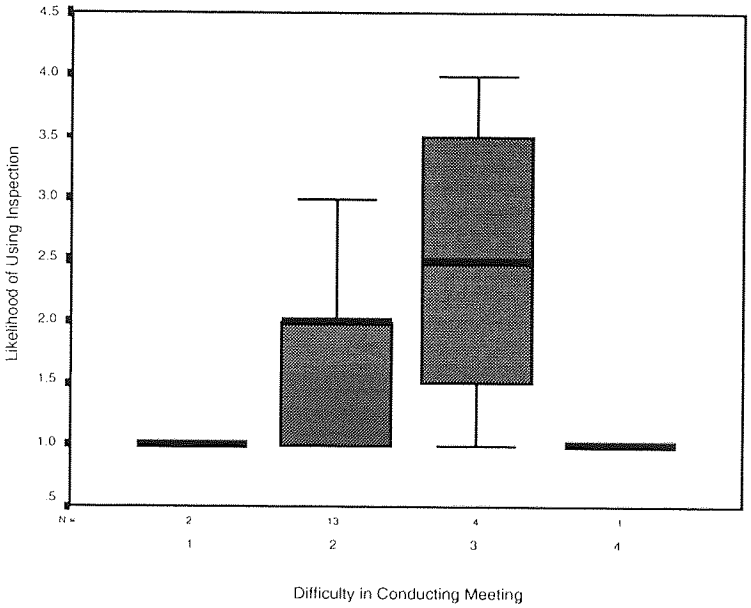

Figure H - 12 : *Difficulty in Conducting Meeting versus the Likelihood of Using Inspection in the Future*

Similar dependency can be observed for the relationship between difficulty in conducting a formal meeting with project members and the likelihood of using inspection in the future (see *Figure H - 12*). At the 50th percentile, the sample that choose 1 as the level of difficulty in conducting a meeting is more likely to use inspection in the future than the sample that choose value 2. The sample that choose 2, in turn is more likely to use inspection than the sample that choose 3. The same pattern holds true at the 25th and 10th percentile,

where the sample that choose 1 is more likely to use inspection in the future than the sample that choose 2, which is more likely to use inspection than the sample that choose 3. There is only one sample that choose 4 as the level of difficulty in conducting a meting. The analysis showed that the more difficulty the sample have in conducting meeting, the less likely that they will use inspection process in the future.

Appendix I

STATISTICAL SUMMARY OF PHASE ONE

Frequencies

Statistics

	N	
	Valid	Missing
Gender	21	0
Age	21	0
Yrs in IT	21	0
Progr Xp	21	0
SA Xp	21	0
Net admin xp	21	0
Sw proj mgr xp	21	0
IS/EDP mgmt xp	21	0
Q mgmt xp	21	0
yrs in sw dev	20	1
how many sw proj	21	0
xp/train in SW Q	21	0
yrs in sw q	10	11
use any si/review method	21	0
use sw review	21	0
use sw audit	21	0
use ftr	21	0
use sw inspect	21	0
use code insect	21	0
use walktru	21	0
Problem using si/rev	16	5
how likely using review in future	20	1
Problem using formal code inspect	14	7
case tool helpfull in inspect?	18	3
distributed case better?	11	10
comm e-mail	21	0

Statistics

	N	
	Valid	Missing
comm meeting	21	0
comm memo	21	0
comm chat	21	0
comm phone	21	0
distribute doc e-mail	21	0
distribute doc via fax	21	0
dist doc via paper	21	0
dist doc via html	21	0
easy to get ref/req doc	21	0
easy to conduct meeting	21	0
Degree	20	1

Gender

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Male	18	85.7	85.7	85.7
	Female	3	14.3	14.3	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

Age

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	20 - 24	3	14.3	14.3	14.3
	25 - 29	13	61.9	61.9	76.2
	30 - 34	5	23.8	23.8	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

Yrs in IT

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 - 2	10	47.6	47.6	47.6
	3 - 4	4	19.0	19.0	66.7
	5 - 6	5	23.8	23.8	90.5
	7 -	2	9.5	9.5	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

Progr Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	20	95.2	95.2	95.2
	no	1	4.8	4.8	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

SA Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	11	52.4	52.4	52.4
	no	10	47.6	47.6	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

Net admin xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	3	14.3	14.3	14.3
	no	18	85.7	85.7	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

Sw proj mgr xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yea	4	19.0	19.0	19.0
	no	17	81.0	81.0	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

IS/EDP mgmt xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	4	19.0	19.0	19.0
	no	17	81.0	81.0	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

Q mgmt xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	2	9.5	9.5	9.5
	no	19	90.5	90.5	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

yrs in sw dev

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 - 2	14	66.7	70.0	70.0
	3 - 4	4	19.0	20.0	90.0
	5 - 6	1	4.8	5.0	95.0
	7 -	1	4.8	5.0	100.0
	Total	20	95.2	100.0	
Missing	.	1	4.8		
	Total	1	4.8		
Total		21	100.0		

how many sw proj

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 - 2	11	52.4	52.4	52.4
	3 - 4	9	42.9	42.9	95.2
	7 -	1	4.8	4.8	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

xp/train in SW Q

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	8	38.1	38.1	38.1
	no	13	61.9	61.9	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

yrs in sw q

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 - 2	9	42.9	90.0	90.0
	3 - 4	1	4.8	10.0	100.0
	Total	10	47.6	100.0	
Missing	.	11	52.4		
	Total	11	52.4		
Total		21	100.0		

use any si/review method

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	15	71.4	71.4	71.4
	no	6	28.6	28.6	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

use sw review

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	10	47.6	47.6	47.6
	no	11	52.4	52.4	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

use sw audit

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	1	4.8	4.8	4.8
	no	20	95.2	95.2	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

use ftr

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	2	9.5	9.5	9.5
	no	19	90.5	90.5	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

use sw inspect

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	9	42.9	42.9	42.9
	no	12	57.1	57.1	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

use code insect

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	11	52.4	52.4	52.4
	no	10	47.6	47.6	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

use walktru

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	12	57.1	57.1	57.1
	no	9	42.9	42.9	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

Problem using si/rev

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	8	38.1	50.0	50.0
	no	8	38.1	50.0	100.0
	Total	16	76.2	100.0	
Missing	.	5	23.8		
	Total	5	23.8		
Total		21	100.0		

how likely using review in future

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	8	38.1	40.0	40.0
	2	7	33.3	35.0	75.0
	3	4	19.0	20.0	95.0
	4	1	4.8	5.0	100.0
	Total	20	95.2	100.0	
Missing	System Missing	1	4.8		
	Total	1	4.8		
Total		21	100.0		

Problem using formal code inspect

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	3	14.3	21.4	21.4
	no	11	52.4	78.6	100.0
	Total	14	66.7	100.0	
Missing	.	7	33.3		
	Total	7	33.3		
Total		21	100.0		

case tool helpfull in inspect?

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	17	81.0	94.4	94.4
	no	1	4.8	5.6	100.0
	Total	18	85.7	100.0	
Missing	.	3	14.3		
	Total	3	14.3		
Total		21	100.0		

distributed case better?

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	8	38.1	72.7	72.7
	no	3	14.3	27.3	100.0
	Total	11	52.4	100.0	
Missing	.	10	47.6		
	Total	10	47.6		
Total		21	100.0		

comm e-mail

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	14	66.7	66.7	66.7
	no	7	33.3	33.3	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

comm meeting

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	19	90.5	90.5	90.5
	no	2	9.5	9.5	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

comm memo

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	13	61.9	61.9	61.9
	no	8	38.1	38.1	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

comm chat

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	5	23.8	23.8	23.8
	no	16	76.2	76.2	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

comm phone

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	16	76.2	76.2	76.2
	no	5	23.8	23.8	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

distribute doc e-mail

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	14	66.7	66.7	66.7
	no	7	33.3	33.3	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

distribute doc via fax

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	3	14.3	14.3	14.3
	no	18	85.7	85.7	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

dist doc via paper

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	19	90.5	90.5	90.5
	no	2	9.5	9.5	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

dist doc via html

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	no	21	100.0	100.0	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

easy to get ref/req doc

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	1	4.8	4.8	4.8
	2	3	14.3	14.3	19.0
	3	10	47.6	47.6	66.7
	4	7	33.3	33.3	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

easy to conduct meeting

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	2	9.5	9.5	9.5
	2	13	61.9	61.9	71.4
	3	4	19.0	19.0	90.5
	4	2	9.5	9.5	100.0
	Total	21	100.0	100.0	
Total		21	100.0		

Degree

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	BSc CS	18	85.7	90.0	90.0
	BSc O	2	9.5	10.0	100.0
	Total	20	95.2	100.0	
Missing	.	1	4.8		
	Total	1	4.8		
Total		21	100.0		

Descriptives

Descriptive Statistics

	N	Minimum	Maximum	Mean		Std.
	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic
how likely using review in future	20	1	4	1.90	.20	.91
easy to get ref/req doc	21	1	4	3.10	.18	.83
easy to conduct meeting	21	1	4	2.29	.17	.78
Valid N. (listwise).	20					

Crosstabs

Case Processing Summary

	Cases					
	Valid		Missing		Total	
	N	Percent	N	Percent	N	Percent
how likely using review in future * easy to conduct meeting	20	95.2%	1	4.8%	21	100.0%
how likely using review in future * xp/train in SW Q	20	95.2%	1	4.8%	21	100.0%
how likely using review in future * use any si/review method	20	95.2%	1	4.8%	21	100.0%
how likely using review in future * Problem using si/rev	16	76.2%	5	23.8%	21	100.0%
how likely using review in future * Problem using formal code inspect	14	66.7%	7	33.3%	21	100.0%
how likely using review in future * easy to get ref/req doc	20	95.2%	1	4.8%	21	100.0%

how likely using review in future * easy to conduct meeting Crosstabulation

Count

		easy to conduct meeting				Total
		1	2	3	4	
how likely using review in future	1	2	4	1	1	8
	2		6	1		7
	3		3	1		4
	4			1		1
Total		2	13	4	1	20

how likely using review in future * xp/train in SW Q Crosstabulation

Count

		xp/train in SW Q		
		yes	no	Total
how likely using review in future	1	4	4	8
	2	1	6	7
	3	2	2	4
	4	1		1
Total		8	12	20

how likely using review in future * use any si/review method Crosstabulation

Count

		use any si/review method		
		yes	no	Total
how likely using review in future	1	5	3	8
	2	7		7
	3	3	1	4
	4		1	1
Total		15	5	20

how likely using review in future * Problem using si/rev Crosstabulation

Count

		Problem using si/rev		
		yes	no	Total
how likely using review in future	1	3	3	6
	2	5	1	6
	3		4	4
Total		8	8	16

how likely using review in future * Problem using formal code inspect Crosstabulation

Count

		Problem using formal code inspect		
		yes	no	Total
how likely using review in future	1		6	6
	2	2	2	4
	3	1	3	4
Total		3	11	14

how likely using review in future * easy to get ref/req doc Crosstabulation

Count

		easy to get ref/req doc				
		1	2	3	4	Total
how likely using review in future	1	1	2	3	2	8
	2		1	3	3	7
	3			3	1	4
	4				1	1
Total		1	3	9	7	20

Appendix J

STATISTICAL SUMMARY OF PHASE THREE

Frequencies

Statistics

	N	
	Valid	Missing
Gender	10	0
Age	10	0
Qualification	10	0
Yrs in IT	10	0
Prog Xp	10	0
SA Xp	10	0
Net Admin Xp	10	0
S/W Proj Mgr Xp	10	0
IS Mgmt Xp	10	0
S/W Q Mgmt Xp	10	0
Yrs in SW Dev	10	0
SW Proj Involved in	10	0
SW Q Xp	10	0
Yrs in SW Q	10	0
Used Inspect	10	0
SW Inspect Xp	10	0
SW Audit Xp	10	0
FTR Xp	10	0
Code Inspect Xp	10	0
Walkthru Xp	10	0
Briefing Sufficient	10	0
Spec better option	10	0
Help Suffice	9	1
Log suffice	10	0
Suitable for asynch	10	0
Briefing complement?	10	0
Remote viewer	10	0
Chat	10	0
Pop	10	0
Suitable for synch	10	0
Flex for synch and asynch	10	0
Better than manual	10	0
Better than one mode	10	0
Flex on web	10	0

Gender

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Male	8	80.0	80.0	80.0
	Female	2	20.0	20.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Age

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	20 - 24	4	40.0	40.0	40.0
	25 - 29	4	40.0	40.0	80.0
	35 -	2	20.0	20.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Qualification

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	BSc CS	3	30.0	30.0	30.0
	MSc CS	5	50.0	50.0	80.0
	MSc Eng	1	10.0	10.0	90.0
	Other	1	10.0	10.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Yrs in IT

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 - 2	1	10.0	10.0	10.0
	5 - 6	3	30.0	30.0	40.0
	7 -	6	60.0	60.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Prog Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	10	100.0	100.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

SA Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	3	30.0	30.0	30.0
	No	7	70.0	70.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Net Admin Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	1	10.0	10.0	10.0
	No	9	90.0	90.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

S/W Proj Mgr Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	2	20.0	20.0	20.0
	No	8	80.0	80.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

IS Mgmt Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	1	10.0	10.0	10.0
	No	9	90.0	90.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

S/W Q Mgmt Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	No	10	100.0	100.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Yrs in SW Dev

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 - 2	1	10.0	10.0	10.0
	3 - 4	7	70.0	70.0	80.0
	5 - 6	2	20.0	20.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

SW Proj Involved in

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 - 2	3	30.0	30.0	30.0
	3 - 4	6	60.0	60.0	90.0
	7 -	1	10.0	10.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

SW Q Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	1	10.0	10.0	10.0
	No	9	90.0	90.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Yrs in SW Q

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	3 - 4	1	10.0	10.0	100.0
		9	90.0	90.0	90.0
	Total	10	100.0	100.0	
Total		10	100.0		

Used Inspect

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	3	30.0	30.0	30.0
	No	7	70.0	70.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

SW Inspect Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	No	3	30.0	30.0	100.0
		7	70.0	70.0	70.0
	Total	10	100.0	100.0	
Total		10	100.0		

SW Audit Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	1	10.0	10.0	80.0
	No	2	20.0	20.0	100.0
		7	70.0	70.0	70.0
	Total	10	100.0	100.0	
Total		10	100.0		

FTR Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	1	10.0	10.0	80.0
	No	2	20.0	20.0	100.0
		7	70.0	70.0	70.0
	Total	10	100.0	100.0	
Total		10	100.0		

Code Inspect Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	2	20.0	20.0	90.0
	No	1	10.0	10.0	100.0
		7	70.0	70.0	70.0
	Total	10	100.0	100.0	
Total		10	100.0		

Walkthru Xp

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	3	30.0	30.0	100.0
		7	70.0	70.0	70.0
	Total	10	100.0	100.0	
Total		10	100.0		

Briefing Sufficient

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	3	30.0	30.0	30.0
	2	6	60.0	60.0	90.0
	3	1	10.0	10.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Spec better option

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	5	50.0	50.0	50.0
	2	4	40.0	40.0	90.0
	3	1	10.0	10.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Help Suffice

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	5	50.0	55.6	55.6
	2	3	30.0	33.3	88.9
	3	1	10.0	11.1	100.0
	Total	9	90.0	100.0	
Missing	System Missing	1	10.0		
	Total	1	10.0		
Total		10	100.0		

Log suffice

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	4	40.0	40.0	40.0
	2	6	60.0	60.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Suitable for asynch

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	5	50.0	50.0	50.0
	2	4	40.0	40.0	90.0
	3	1	10.0	10.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Briefing complement?

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	7	70.0	70.0	70.0
	2	3	30.0	30.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Remote viewer

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	7	70.0	70.0	70.0
	2	3	30.0	30.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Chat

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	7	70.0	70.0	70.0
	2	3	30.0	30.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Pop

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	7	70.0	70.0	70.0
	2	3	30.0	30.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Suitable for synch

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	7	70.0	70.0	70.0
	2	3	30.0	30.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Flex for synch and asynch

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	5	50.0	50.0	50.0
	2	3	30.0	30.0	80.0
	3	2	20.0	20.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Better than manual

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	7	70.0	70.0	70.0
	2	3	30.0	30.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Better than one mode

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	8	80.0	80.0	80.0
	2	2	20.0	20.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Flex on web

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	9	90.0	90.0	90.0
	2	1	10.0	10.0	100.0
	Total	10	100.0	100.0	
Total		10	100.0		

Descriptives

Descriptive Statistics

	N	Minimum	Maximum	Mean		Std.
	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic
Briefing Sufficient	10	1	3	1.80	.20	.63
Spec better option	10	1	3	1.60	.22	.70
Help Suffice	9	1	3	1.56	.24	.73
Log suffice	10	1	2	1.60	.16	.52
Suitable for asynch	10	1	3	1.60	.22	.70
Briefing complement?	10	1	2	1.30	.15	.48
Remote viewer	10	1	2	1.30	.15	.48
Chat	10	1	2	1.30	.15	.48
Pop	10	1	2	1.30	.15	.48
Suitable for synch	10	1	2	1.30	.15	.48
Flex for synch and asynch	10	1	3	1.70	.26	.82
Better than manual	10	1	2	1.30	.15	.48
Better than one mode	10	1	2	1.20	.13	.42
Flex on web	10	1	2	1.10	1.00E-01	.32
Valid N (listwise)	9					

Crosstabs

Case Processing Summary

	Cases					
	Valid		Missing		Total	
	N	Percent	N	Percent	N	Percent
Better than manual * Suitable for synch	10	100.0%	0	.0%	10	100.0%
Better than manual * Suitable for asynch	10	100.0%	0	.0%	10	100.0%
Better than manual * Flex for synch and asynch	10	100.0%	0	.0%	10	100.0%

Better than manual * Suitable for synch Crosstabulation

Count

		Suitable for synch		Total
		1	2	
Better than manual	1	6	1	7
	2	1	2	3
Total		7	3	10

Better than manual * Suitable for asynch Crosstabulation

Count

		Suitable for asynch			Total
		1	2	3	
Better than manual	1	4	3		7
	2	1	1	1	3
Total		5	4	1	10

Better than manual * Flex for synch and asynch Crosstabulation

Count

		Flex for synch and asynch			Total
		1	2	3	
Better than manual	1	4	2	1	7
	2	1	1	1	3
Total		5	3	2	10

Crosstabs

Case Processing Summary

	Cases					
	Valid		Missing		Total	
	N	Percent	N	Percent	N	Percent
Flex for synch and asynch * Better than one mode	10	100.0%	0	.0%	10	100.0%
Flex for synch and asynch * Flex on web	10	100.0%	0	.0%	10	100.0%

Flex for synch and asynch * Better than one mode Crosstabulation

Count

		Better than one mode		Total
		1	2	
Flex for synch and asynch	1	4	1	5
	2	2	1	3
	3	2		2
Total		8	2	10

Flex for synch and asynch * Flex on web Crosstabulation

Count

		Flex on web		Total
		1	2	
Flex for synch and asynch	1	5		5
	2	3		3
	3	1	1	2
Total		9	1	10