The dance of classes - A stochastic model for software structure evolution

Rafael Prates Ferreira Trindade*, Talita Santana Orfanó*, Kecia Aline Marques Ferreira* and Elizabeth Fialho Wanner*
*Federal Center for Technological Education of Minas Gerais - Department of Computing
Av. Amazonas, 7675 - Nova Gameleira - Belo Horizonte - MG - Brazil
Email: rafapft@gmail.com, taliorfano@yahoo.com.br, kecia@decom.cefetmg.br, efwanner@decom.cefetmg.br

***Abstract*—In this study, we investigate software structure evolution and growth. We represent software structure by means of a generic macro-topology called *Little House*, which models the dependencies among classes of object-oriented software systems. We, then, define a stochastic model to predict the way software architectures evolve. The model estimates how the classes of object-oriented programs get connected one to another along the evolution of the systems. To define the model, we analyzed data from 81 versions of six Java based projects. We analyzed each pair of sequential versions, for each project, in order to depict a pattern of software structure evolution based in *Little House*. To evaluate the model, we performed two experiments: one with the data used to derive the model, and another with data of 35 releases, in total, of four open-source Java project. In both experiments, we found a very low rate of error for the application of the proposed model. The evaluation of the model suggests it is able to predict how a software structure will evolve.**

***Keywords*-software structure; software evolution; code history comprehension; stochastic model; complex networks.**

I. Introduction

As a software system evolves, it continuously changes and grows, its complexity rises, whereas its quality declines [1]. The maintenance tasks, then, may become increasingly risky. In this context, the characterization of the software evolution process plays a central hole in Software Engineering. Many studies have been carried out to characterize software evolution [2]. However, there are still many questions that need to be investigated in order to consolidate the corpus of knowledge on software evolution, for example: *How do the internal structures of the software systems evolve? When does the software architecture start to rot?*. Answering such questions may help software engineers to better comprehend the software evolution process, as well as to define methods and tools to control or to avoid the effects of rotting design.

A recent survey carried out by Codoban et al. [3] with 217 developers has investigated the importance of knowing the software history. The results of that survey indicate the following main conclusion: software history is hard to understand and this difficulty lead to lost of context of the original code; organizing software history is difficult; comparing multiple parts of a code history is a hard task; resources for history visualization is remarkably desirable by developers because such resources can allow tracking the movement of a file through time, for instance.

This work is concerned with the problem of software evolution at the level of software architecture. We investigate the following research question: *RQ - Is there a pattern of software evolution at the level of architecture?* To answer this question, we investigate what happens between each pair of sequential versions along the evolution of a software system. For this purpose, we represent the software architecture by a topology called *Little House* [4]. This topology is a generic model that represent how the modules of a software system are connected one to another. In particular, we aim to understand how the *Little House* formation of software systems evolves and to investigate whether such evolution can be described by a pattern. We investigate if there is a generic mathematical model to represent the way the software architecture evolves in the sense of movement of files. The answer of this research question has leaded to a stochastic model, based in Markov Chains, that represent the evolution of software systems. Our study is focused in object-oriented software systems. A data set of 81 releases of six Java based projects was used to define the model and to evaluate it. We also evaluated the application of the model in 35 releases of four Java based projects.

The main contributions of this work are the following:

- the definition of a stochastic model for software evolution, based in Little House (Section IV);
- the results of the evaluation of the model (Section V);
- the analysis of the asymptotic distribution of the proposed model (Section V-1);
- the results of the test of the model in four open-source Java projects, with small, medium, and large size (Section V-A).

II. Background

This section provides background in the following concepts that are applied in this work: a model called *Little House* and Markov Chains.

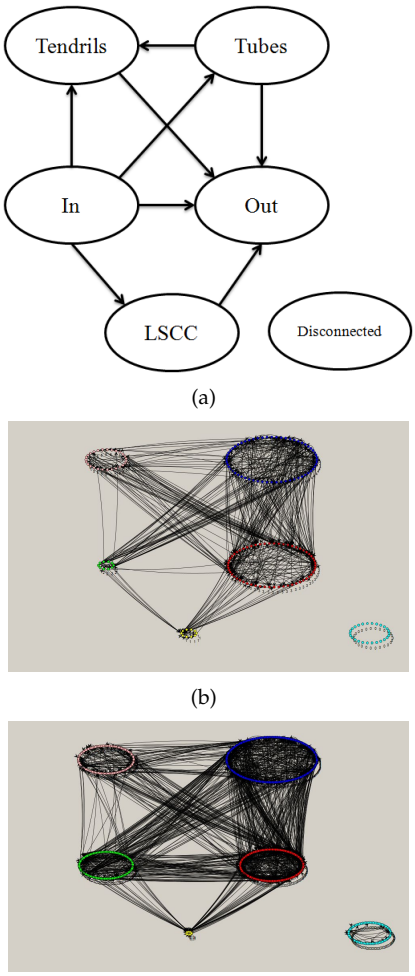


Fig. 1. (a) The *Little House* model [4]. The *Little House* topology of JHotDraw (b) version 5.2 and (c) version 6.0.

A. Little House

Ferreira et al. [4] identified that the so-called Bow-tie model [5], a model for the Web graph, is applicable to the dependence graphs of classes. In a dependence graph of classes, the nodes represent the classes, and the edges represent the dependencies between the classes. A class *X* depends upon a class *Y* when *X* uses a method or an attribute of *Y*, or when *X* inherits of *Y*. The authors, then, rephrased the Bow-tie model, resulting in a macro-graph named *Little House*, shown in Figure 1.

Little House is a macro-graph constituted by six nodes, namely: *Disconnected*, *IN*, *OUT*, *LSCC*, *TUBES*, and *TENDRILS*. These names were inherited from the Bow-tie model, the precursor of *Little House*. Each of these nodes is constituted by a set of classes connected one to another. These nodes are called "components of *Little House*" and are defined as follows [4]: ***Disconnected***, it contains the classes that are not connected to classes of other components; ***LSCC*** (Largest Strongly Connected Component), it is the "critical component" of *Little House*, since a class of *LSCC* reaches any other class inside *LSCC*, direct or indirectly; ***IN***, its classes may use classes of any component, except *Disconnected*, but its classes are not used by classes of other components; ***OUT***, its classes may be used by classes of other components, except *Disconnected*, but they do not use classes of other components; ***TUBES***, it works like a bridge between *IN* and *OUT*, in the following way: a class of *IN* uses a class of *TUBES*, which uses a class of *OUT*; ***TENDRILS***, it works like a longer bridge between *IN* and *OUT*, since the link among classes may occur in the following way: a class of *IN* uses a class of *TUBES*, which uses a class of *TENDRILS*, which reaches a class of *OUT*.

Previous studies have shown that *OUT* and *LSCC* are the most critical components of *Little House*, regarding the internal quality of their classes, as well as the change impact propagation of their classes [6], [7]. The present work applies *Little House* to understand software evolution.

B. Finite Markov Chain

Formal definition and basic properties of Finite Markov Chain are described in this section. Further information about Finite Markov Chain and stochastic process in general can be found in [8]. A Markov Chain is a memoryless, homogeneous, stochastic process with a finite number of states. In a process, a system changes after each time step t while in a stochastic process, such changes are random. The states are labeled by the elements of the set $\{1, 2, \cdots, n\}$, with X_t denotes the process state at time t. The state transition $i \leftarrow j$ at time t is indicated by $X_t = i$ and $X_{t+1} = j$. The process is memoryless if the probability of $i \leftarrow j$ transition does not depend on the history of the process. The process is homogeneous if it does not depend on the time t.

The transition matrix is a stochastic $n \times n$ matrix, $T = (p_{ij})$ in which $p_{ij} = P(X_{t+1} = j | X_t = i)$ represents the probability of transitioning from state i to state j. T being a stochastic matrix means that each entry is non-negative real numbers and the sum of each row is equal to 1.

The initial distribution is given by a $1 \times n$ vector $X(0) = (x_{01}, x_{02}, \cdots, x_{0n})$, $\sum_{i=1}^{n} x_{0i} = 1$ in which $x_{0i} = P(X_0 = i)$ is the probability that i is the initial state. The distribution of the states after k steps can be determined by taking the initial distribution and multiplying it by the transition matrix T raised by the k^{th} power, that is

$$X(t) = X(0) \times T^t, \quad \forall t \tag{1}$$

A stationary distribution for a Markov Chain is a vector $1 \times n$ q such that $q = q \times T$. Observe that if q is a stationary distribution then $q(t) = q(t + n)$ for all $n \in \mathbb{N}$.

Markov Chain is related to the dynamic system which evolves throughout time. In this way, it is interesting to know how the system will behave as time $t \leftarrow \infty$. Under

some easy-to-check conditions, a Markov Chain possesses a limiting distribution. If $T^{\infty} = \lim_{t \leftarrow \infty} T^t$ exists then each row of T^{∞} is called a limiting or an asymptotic distribution of the Markov Chain. If one observes a Markov Chain at some random time way out in future, then $T^{\infty}(j)$ is the probability of the state j.

III. Methods

We considered the following number of versions of six open-source Java projects: JHotDraw - 6, JMoney - 11, jSLP - 14, jSCH - 38, BlueCove - 7, and JUnit - 5. These software systems were chosen to be analyzed in this work due to three criteria: they are among the top most used software systems in empirical studies about software evolution [9], some of them was used in empirical studies about *Little House* [4], [7], and the number of versions they have is not too large. The number of versions was an important criterion because some evaluation demanded manual inspection.

The data were gathered using a tool called *Connecta*[1]. In its recent version, *Connecta* allows comparing the *Little House* data of two versions of a software systems. The results of the comparison report the state of each class of the system: added, removed or continued. Given the set of continued classes, i.e, the classes that are in both versions, *Connecta* generates reports about the "movement" of those classes among the components of *Little House*. In this work, we call the set of changes between two sequential versions as "evolution step". *Connecta*, then, provides the data of evolution steps to be used in this study.

The data of an evolution step are represented by a square 6×6 matrix M'. In this matrix, each line and each column corresponds to a component of *Little House*. The matrix contains the number of classes that migrated between two components in a evolution step size. So, $M'_{In,Tubes}$, for instance, represents the number of classes that migrated from *Tubes* to *In*.

First, the M' matrix was gathered for all pairs of sequential versions of the software systems. In the sequence, a corresponding probability matrix P' of migration of classes was computed. The matrix contains the probabilities of migration of classes between the pair of components. This matrix was computed from M': each element of P' is given by the corresponding element of M' divided by the sum of elements in the line in M'. So, $P'_{In,Tubes}$, for instance, represents the probability of a class migrates from *Tubes* to *In*.

A global matrix M was calculated by summing up the matrices M', as well as the partial probability matrices was summarized in a global probability matrix P, that is the central result of the study. From this global probability matrix, a stochastic model for the evolution of the *Little House* topology was defined based on the concepts shown in Section II-B.

[1]http://goo.gl/vVnJSE

The proposed model was evaluated as follows. (1) First, we applied the model to the releases of the six Java projects used to define the model. To assess the behavior of the model, we measured the error, that indicates the difference between the predicted distribution of classes among the components of Little House and the actual one. (2) After, we analyzed the asymptotic distribution of the Markov Chain of the model, to identify the limiting distribution of classes among the components of Little House. We compared the asymptotic distribution with the mean distribution found in the first evaluation. (3) Finally, we evaluated the model in other four Java open-source software systems, in a total of 35 releases. In this evaluation, we also measured the error.

IV. A Stochastic Model for Software Evolution

A. The Transition Matrix

TABLE I
Matrix M - the total number of classes that migrated among the *Little House* components

From \ To	*Disc.*	*LSCC*	*In*	*Out*	*Tubes*	*Tend.*	**Total**
Disc.	2084	2	5	2	0	8	2101
LSCC	3	1285	22	27	0	0	1336
In	0	30	1275	4	0	2	1311
Out	7	24	4	1287	30	18	1370
Tubes	0	0	6	43	613	11	673
Tend.	8	0	5	21	20	880	934
Total	2102	1340	1317	1384	663	919	

TABLE II
Probabilities of transitions between the *Little House* components in a evolution step

From \ To	*Disc.*	*LSCC*	*In*	*Out*	*Tubes*	*Tend.*
Disc.	0,9919	0,022	0	0,0051	0	0,0086
LSCC	0,0010	0,9611	0,0229	0,0175	0	0
In	0,0024	0,0165	0,9725	0,0029	0,0089	0,0054
Out	0,0010	0,0202	0,0031	0,9394	0,0639	0,0225
Tubes	0	0	0	0,0219	0,9108	0,0214
Tend.	0,0038	0	0,0015	0,0131	0,0163	0,9422

Table I shows the matrix M, which contains the global result of the evolution steps gathered in the study. Table II shows the probability matrix P. In the matrices, the main diagonal contains the higher values of a line. This result means that as a software system evolves, most of its classes do not migrate of *Little House* component.

The flow of classes from *TUBES* to *OUT* is the higher, that means as a software system evolves, more classes tend to behave as a *provider of services* in the system. There is no migration of classes from *TUBES* and *TENDRILS* to *LSCC*. This finding because is consistent with the *Little House* topology, in which there is no edge between *TUBES* and *LSCC* nor between *TENDRILS* and *LSCC*.

Figure 2 shows the graphs that represent the migration of classes among the Little House components. In this graph, the nodes represent the Little House components,

and the directed edges represent the migration of classes between two components. The edges are labeled with the probability that a class will migrate from a component to another when a new release of the system is launched.

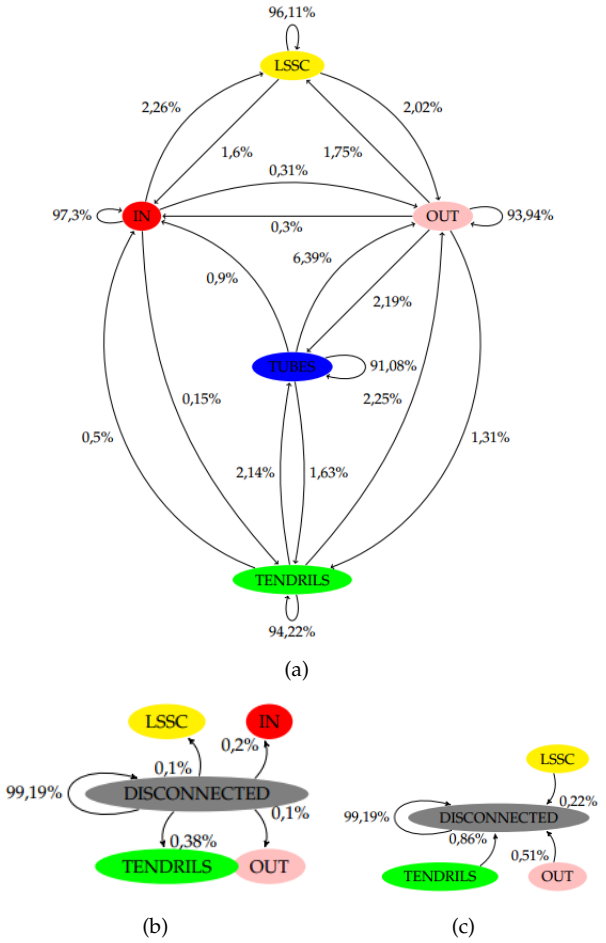


Fig. 2. (a) The graphic view of the migration of classes among the components of Little House. (b) Migration of classes from Disconnected to other components. (c) Migration of classes to Disconnected.

B. The Model Definition

The software evolution was modeled as a stochastic process, specifically a Finite Markov Chain. Applying definitions presented in Section II-B to the problem investigated in this work, the intervals of time correspond to each version of a software system, the states correspond to the distribution of classes among the components of *Little House*, and the transition matrix corresponds to the transpose of the matrix shown in Table II.

Considering that the distribution of classes in a version $t+1$ only depends on the distribution of classes in a version t, it is possible to say that the process is memoryless. Furthermore, assuming that the transition matrix is time-invariant, it is possible to say that it represents a finite Markov Chain.

The probability matrix P, shown in Figure 3, is the central part of this model, because it is used to predict the way the software structure will evolve. The proposed model to predict how a software system will evolve regarding the *Little House* topology comprises three steps that are described as follows.

1) Compute the total of classes per *Little House* component in the current version of the software system, registering the result in column array x, as follows.

$$x(k) = \begin{bmatrix} N_{Disconnected} \\ N_{LSCC} \\ N_{In} \\ N_{Out} \\ N_{Tubes} \\ N_{Tendrils} \end{bmatrix}$$

2) Compute the probability array for the software. In this array, each position will indicate the likelihood of a class to belong to a given component. For instance, the first position of x, $x[1]$, corresponds to the number of classes that belong to Disconnected. In the probability array, the position i will indicate the chance of a class to belong to Disconnected. Therefore, the values of the probability array are given by dividing the value of each position of x for the sum of the values in x.

3) Multiply the probability matrix P, shown in Figure 3, by the resulting probability array.

The resulting vector represents the probability of finding classes on each of class presented in *Little House* topology and can be seen as the evolution step of the project.

V. Evaluation of the Model

The proposed model was evaluated in all versions of the sample of this study. The evaluation was carried out to verify how well the model fits the evolution of the software systems. For this purpose, the error of estimation of the model was calculated in each case. It is worthwhile to say that the higher number of software samples, the smaller the error is. It is due to the accuracy of the transition matrix. However, since it is a preliminary study, only six software systems were used. Even that, the goal is to evaluate the model application, identifying the pros and the cons of the proposed stochastic model.

The error was calculated by multiplying the probability array x, which contains the proportion of classes per *Little House* component, by the transition matrix P. Then, the expected probability array x_e is found.

As an example, given the first version of jSCH, the expected probability array x_e, representing the expected next version distribution of states, is calculated as shown in Figure 4. Starting with the initial distribution of classes of jSCH, given by x_1:

$$P = \begin{bmatrix} 0,9919 & 0,022 & 0 & 0,0051 & 0 & 0,0086 \\ 0,0010 & 0,9611 & 0,0229 & 0,0175 & 0 & 0 \\ 0,0024 & 0,0165 & 0,9725 & 0,0029 & 0,0089 & 0,0054 \\ 0,0010 & 0,0202 & 0,0031 & 0,9394 & 0,0639 & 0,0225 \\ 0 & 0 & 0 & 0,0219 & 0,9108 & 0,0214 \\ 0,0038 & 0 & 0,0015 & 0,0131 & 0,0163 & 0,9422 \end{bmatrix}$$

Fig. 3. The matrix P: it represents the probabilities of transitions of a class between each par of the *Little House* components during an evolution step.

$$x_e = \begin{bmatrix} 0,9919 & 0,022 & 0 & 0,0051 & 0 & 0,0086 \\ 0,0010 & 0,9611 & 0,0229 & 0,0175 & 0 & 0 \\ 0,0024 & 0,0165 & 0,9725 & 0,0029 & 0,0089 & 0,0054 \\ 0,0010 & 0,0202 & 0,0031 & 0,9394 & 0,0639 & 0,0225 \\ 0 & 0 & 0 & 0,0219 & 0,9108 & 0,0214 \\ 0,0038 & 0 & 0,0015 & 0,0131 & 0,0163 & 0,9422 \end{bmatrix} \begin{bmatrix} 0,35 \\ 0,2 \\ 0,25 \\ 0,125 \\ 0,025 \\ 0,05 \end{bmatrix} \quad (2)$$

Fig. 4. Example of the application of the model to the first version of the project jSCH.

$$x_1 = \begin{bmatrix} 0,3500 \\ 0,2000 \\ 0,2500 \\ 0,1250 \\ 0,0250 \\ 0,0500 \end{bmatrix} \quad (3)$$

the expected distribution of classes in *Little House*, in the next evolution step, is given by x_e:

$$x_e = \begin{bmatrix} 0,3487 \\ 0,2005 \\ 0,2481 \\ 0,1253 \\ 0,0266 \\ 0,0509 \end{bmatrix} \quad (4)$$

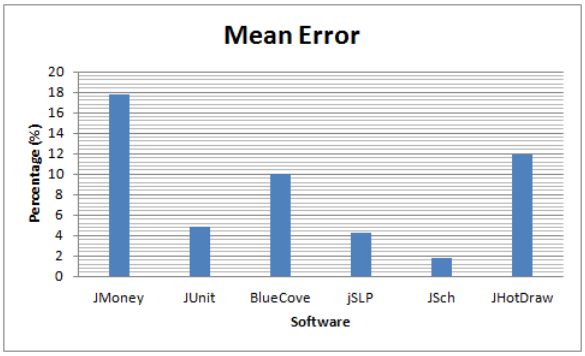


Fig. 5. Mean errors of estimation of the model

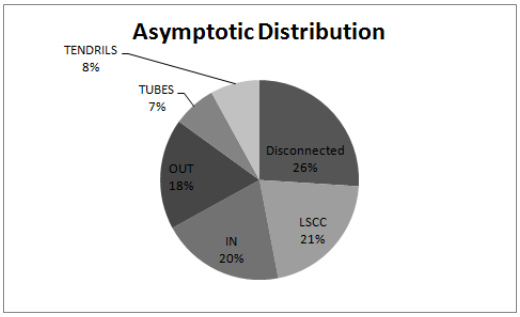
Using this results, the software engineers would expect, for example, 24,8% of the classes in *LSCC* in the next version. The error value, for each evolution step, is given by the Euclidean distance between the expected distribution and the actual distribution vectors. In the example, the error is given as shown in Equation 5.

$$error = \left\| \begin{bmatrix} 0,3487 \\ 0,2005 \\ 0,2481 \\ 0,1253 \\ 0,0266 \\ 0,0509 \end{bmatrix} - \begin{bmatrix} 0,3544 \\ 0,2025 \\ 0,2405 \\ 0,1266 \\ 0,0253 \\ 0,0506 \end{bmatrix} \right\| = 0.0099 \quad (5)$$

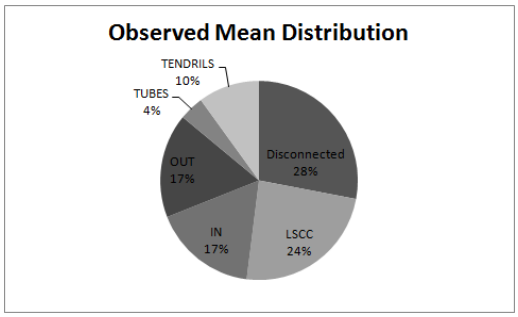
After calculation the error between two versions of each software system, its mean estimation error was calculated. Figure 5 shows the mean estimation error of the model in the systems. The evolution of jSCH has the best fitting by the model, with a mean estimation error of 1,8%. The higher error of estimation is in the evolution of JMoney, 17,14%. However, we observed few outliers during the evolution of JMoney. For instance, in the third version, many classes moved out of component. From the sixth to the seventh version, it seems that a huge reformulation occurred in JMoney, because many classes were deleted, and new classes were included in the project. The results suggest that the proposed stochastic model can predict how the structure of a software system will evolve, considering the *Little House* topology, with a small error.

1) Asymptotic Distribution: Using the transition matrix P, it is possible to show that this Markov Chain has an asymptotic distribution, T^{∞}, given by Equation 6. This result seems to indicate that it is the limiting distribution of a Java system as the number of versions is bigger enough. Figure 6(a) shows a chart graph representing the probability of finding classes in each class of the *Little House* topology when the number of versions tends to ∞.

$$T^{\infty} = \begin{bmatrix} 0,2597 \\ 0,2080 \\ 0,2036 \\ 0,1822 \\ 0,0645 \\ 0,0821 \end{bmatrix} \quad (6)$$



(a) Asymptotic Distribution



(b) Observed mean distribution

Fig. 6. Asymptotic distribution and the mean distribution of the considered stochastic process.

It is worth while to notice that the observed mean distribution of the classes of the considered systems, shown in Figure 6 (b) is not far from the asymptotic distribution presenting an error of 5.8% approximately.

A. Evaluation of the Model in other Projects

We evaluated the behavior of the model in four open-source Java projects that do not belong to the sample from which the model was derived. In this evaluation, we investigated how well the proposed model is able to predict the architectural evolution of software systems.

We considered, in total, 35 versions of the projects Guice, DHCP, and Hibernate, versions 3.0 and 3.1. We considered these two distinct versions of Hibernate because there are many differences among them. DHCP is a small size project. We evaluated all the ten versions of DHCP that were available on its repository at the time of this experiment. Guice is small in the beginning of its life, but has continuously increased along it evolution. We evaluated six versions of Guice, the ones whose bytecodes are available in the repository. Hibernate is a large size project, and it is popular and stable. As Hibernate has a large amount of releases, we did not considered all of them. We evaluated 11 releases of version 3.0, and 8 releases of version 3.1.

From the results, is is possible to observe that the error of the application of the model tends to decrease as the software systems evolves. In DHCP, the error of the model in the first pair of sequential versions is 0.4821, and of the last one is 0.0338. DHCP started with an error of 0.0240 and ended with no error, that is, the model perfectly predicted the evolution of the last two versions of DHCP. The initial error in Hibernate 3.0 is 0.0240, and the last one is 0.0131. The exception was Hibernate 3.1, which started with a very low error, near to zero, increased to 2.8319, and, then, continuously decreased, reaching the last error of 0.6142.

The mean errors observed in this study are very low. The resulting mean error when applying the proposed model to the Guice, DHCP, Hibernate 3.0, and Hibernate 3.1 are, respectively, 0.2167, 0.3108, 0.1164, and 0.0226. This result suggests that the proposed model is valid to predict the evolution of software systems architecture considering the Little House pattern.

VI. Discussion

A. Applications

The main application of the model is that it explains how the software structure will evolve. The history of the software structure is observed in this model by means of the movement of classes inside the software system along the time.

In this work, the history of the software structure is organized in a linear way: each point of the history is a version of the system, represented by its *Little House* composition. The versions are compared one with another by considering the movement of classes among the components of *Little House*. The model *Little House* itself provides a simple and generic mental model of software architecture. Studying the movement of classes by means of *Little House* might be of help to decrease the difficult to assess the software structure evolution. Each version of the software system is represented by a "photo", that is the graphical representation of the *Little House* pattern formation of that version. Therefore, visualizing software structure evolution may be also improved by the results of this work.

B. Limitations

The model was evaluated in the sample used to derive the model, i.e., the transition matrix was applied to estimate the changes in the same Java projects used to train the stochastic model. Doing this, one may argue that such training process could be a threat to the validity of the study. However, we consider that every data from the sample are important to improve the model since few projects were used to test the model. The more data, the better the model is. To overcome this threat, we also

evaluated the proposed model in other four software systems, with small, medium, and large size. In both cases, the model performed well, with a very low mean error.

Doing the proposed analysis, the stochastic model can be used to estimate how much the models explain the existing data instead of being used to predict the next version. In the evaluation, we compared the model with each pair of sequential versions, for each software system. The results of our analysis have shown that, in general, when a large error occurred, it was possible to verify some features that were presented when different versions were compared. From our results, it is possible to verify that the errors were very low implying that the model can efficiently explain the available data.

We considered open-source projects in this study. So, we are not able to assure that our conclusions can be generalized to proprietary software.

VII. Related Work

Software evolution characterization is a main issue in Software Engineering, and many studies about this subject, using various different approaches, have been carried out in the last years [9]. Israeli and Feitelson [10] investigated whether the Lehman's laws are applicable to the Linux kernel. They found that the Lehman's laws were noticed in the Linux kernel, except the growth of complexity and the lost of quality. Caneill and Zacchiroli [11] studied the evolution of Debian by means of software metrics, such as size of the system, size of the packages, and size of the files. They found that the size of Debian has increased when maintenance was performed on it and that the bigger packages have became even bigger. Herraiz et al. [12] investigated the evolution of the value of software metrics along the evolution of software. They found that all the investigated metrics are modeled by a Pareto distribution. Kulkarni [13] studied the evolution of the dependence graphs of object-oriented projects. The main conclusion of their work is that simple relations among classes evolve towards more complex relations.

The present work in concerned with the identification of a pattern for software structure evolution and growth. For this purpose, we considered the Little House model, which is previously proposed in the literature [4], to define a stochastic model for software evolution. The proposed model brings insights about the pattern formation of software structure. We carried out a deeper analysis about how the inner structures of software systems evolve. We envision that this model can provide a predicting resource for software engineers.

VIII. Conclusion

Developers have pointed the following main challenges in the inspection of software history: *history is hard to understand* and *organization of history is difficult* [3]. The results of the present work contribute to overcome those challenges. A macro-topology of software networks, named *Little House*, was the basis of this study. We investigated the pattern of migration of classes between the components of *Little House* along the evolution of the software system. From the results of our study, we can conclude that the answer of RQ1 is *yes, software architecture evolution can be described by a pattern*. In this work, we defined a model for the evolution of the structure of software systems. The model is based in stochastic Finite Markov Chain theory.

Although the results found in this work are positive, there are need of further investigation. A detailed qualitative analysis should be performed. Moreover, it would be of value to assess whether there are significant differences in the evolutionary patterns of software systems regarding type, size, and application domain.

References

[1] M. M. Lehman, "Laws of software evolution revisited," in *Proceedings of the 5th European Workshop on Software Process Technology*, ser. EWSPT '96. London, UK, UK: Springer-Verlag, 1996, pp. 108–124.

[2] M. Di Penta, "Empirical studies on software evolution: Should we (try to) claim causation?" in *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, ser. IWPSE-EVOL '10. New York, NY, USA: ACM, 2010, pp. 2–2.

[3] D. D. Mihai Codoban, Sruti Srinivasa Ragavan and B. Bailey, "Software history under the lens: A study on why and how developers examine it," in *Proceedings of the International Conference on Software Maintenance*, ser. ICSM '15. IEEE, 2015, pp. 1–10.

[4] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, and B. M. Gomes, "Software evolution characterization - a complex network approach," in *Proceeding of the Brazilian Symposium on Software Quality - SBQS*, Curitiba, Brazil, 2011, pp. 1–15.

[5] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," in *WWW9 Conference*, 2000, pp. 309–320.

[6] K. A. M. Ferreira, M. A. S. Bigonha, R. C. N. Moreira, and R. S. Bigonha, "The evolving structures of software systems," in *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics, WETSoM 2012, Zurich, Switzerland, June 3, 2012*, 2012, pp. 28–34.

[7] M. M. Ferreira, K. A. M. Ferreira, and H. T. Marques-Neto, "Mapping the potential change impact in object-oriented software," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: ACM, 2015, pp. 1654–1656.

[8] O. Haggstrom, *Finite Markov Chain and Algorithmic Applications*. Cambridge: Cambridge University Press, 2002.

[9] M. M. Syeed, I. Hammouda, and T. Systä, "Evolution of open source software projects: A systematic literature review," *Journal of Software*, vol. 8, no. 11, pp. 2815–2829, Nov. 2013.

[10] A. Israeli and D. G. Feitelson, "The linux kernel as a case study in software evolution," *J. Syst. Softw.*, vol. 83, no. 3, pp. 485–501, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.09.042

[11] M. Caneill and S. Zacchiroli, "Debsources: Live and historical views on macro-level software evolution," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 28:1–28:10.

[12] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, "Towards a theoretical model for software growth," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 21–.

[13] N. Kulkarni, S. P. Bommaraju, and M. Dasa, "Structural evolution of software: A social network perspective," in *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*, ser. WETSoM 2014, New York, NY, USA, 2014, pp. 19–22.