Towards a Model to Support *in silico* Studies of Software Evolution

Marco Antônio Pereira Araújo
COPPE/UFRJ
Caixa Postal: 68511
Rio de Janeiro, Brazil
+55 21 2562-8712
maraujo@cos.ufrj.br

Vitor Faria Monteiro
COPPE/UFRJ
Caixa Postal: 68511
Rio de Janeiro, Brazil
+55 21 2562-8712
vitorfaria@cos.ufrj.br

Guilherme Horta Travassos
COPPE/UFRJ
Caixa Postal: 68511
Rio de Janeiro, Brazil
+55 21 2562-8712
ght@cos.ufrj.br

ABSTRACT

Software evolution is recognized as one of the most challenging areas in the field of Software Engineering. The observation of evolution is time-dependent, reducing opportunities for actual observations in short periods of time. Usually, maintenance cycles are proportional to the software life cycle. Therefore, the amount of research has not been enough to deal with all the issues related to the evolution of software. However, simulation through confident models represents an interesting strategy to support software decay observation in short period of time. Towards that, this paper describes a model aimed at supporting the software decay simulation through systems dynamics. The Laws of Software Evolution and ISO 9126 were used as initial knowledge to support the discovery of software characteristic (size, periodicity, complexity, effort, reliability, and maintainability) relationships. Next, evidence to strengthen the existence of such relationships was acquired through quasi-systematic literature reviews. In sequence, the model was applied to support the simulation of industrial software decay. The results suggested its feasibility and correctness, making it an interesting candidate to support future software decay studies.

Categories and Subject Descriptors

K.6.3 [**Management of Computing and Information Systems**]: Software Management – *software development*, *software maintenance, software process.*

General Terms

Management, Measurement, Experimentation.

Keywords

Software Evolution, Software Maintenance, Object-oriented Software, Simulation Model, *in silico* Study, Experimental Software Engineering.

1. INTRODUCTION

Experience and practice in software development allows the understanding that change is an intrinsic and almost inevitable characteristic in the software development process. Software evolution has been recognized as one of the most problematic and challenging areas in the field of Software Engineering. It has been observed that over 80% of the development life cycle costs have been spent after system delivery [1].

According to Lehman and Ramil [2], the greater the understanding of systems evolution the greater the possibility will be of improving methods and processes towards its planning, management, and implementation.

Despite its importance and all the effort put in this topic, results are not enough to support its full comprehension. In general**,** researchers and practitioners have neither sufficient information nor the understanding as to why and how systems gradually change over time [3]. Therefore, software engineers usually face unknown risks when dealing with the planning and managing of the software evolution processes.

In a Software Engineering context, the term evolution can be interpreted in two ways. One view considers that important topics regarding the evolution of software are the ones that describe the means through which evolution occurs, i.e., the mechanism, instruments, and tools used in this process. In this case, the focus is the understanding of how software is modified. Another view seeks to understand the nature of the evolution phenomenon, its causes, properties and characteristics, consequences, impacts, management, and control. In this case, the focus is to investigate what software evolution is and why it happens. In this context, the Laws of Software Evolution (LSEs) mean to support the understanding of the evolutionary process of software [3].

The technical literature has offered several papers in this area. Most of them discuss the evolution of software usually on a conceptual level, or by observing the evolution of versions of a particular system, considering the source code of legacy systems or free software [4]. In contrast, Smith and Ramil [5] presented an initial model of System Dynamics for Software Evolution through qualitative analysis, without providing a tool to support model execution. Stopford and Counsell [6] presented the only study we were able to find in the technical literature which consists of a framework for simulating the evolution of software based on a fictitious simulation process.

Accepting that all LSEs [7] are valid (Table 1) and dependent on each other, they can be used to organize a conceptual environment to support the definition and execution of *in silico* studies of software decay throughout the software development process activities. It's not the purpose of this work to prove or reject any of the Laws of Software Evolution but use them to prepare a conceptual environment to support the definition of experimental studies concerned with the different object-oriented software development process phases. We are using, as a basis for our research, the Requirements Specification, High and Low Level Design and Coding phases, usually present in most of the OO software development processes.

Considering this, Araújo [4] introduced a model to support the observation of software evolution for *E-type systems* [9] through simulation, using systems dynamics. An *E-type system* represents software for solving a problem or addressing an application in the real word. More specifically, the model intends to allow the observation of software quality decay along the execution of maintenance cycles and thus aims at providing a better understanding of how the software can be affected by various changes throughout its life cycle. The LSEs and ISO 9126 [11] represent the starting points to build such model. A set of software characteristics and possible relationships among them has been established based on interpreting the LSEs formulations (Table 1).

To strengthen the confidence in the model's validity and investigate the possible influences amongst software characteristics, a series of secondary studies has been done [8]. It allowed identifying evidence in technical literature on the following software characteristics relationships: Size, Periodicity, Complexity, Effort, Reliability and Maintainability. Besides, to increase our confidence in the model, it was used to simulate consecutive maintenance cycles (*in silico* study) concerned with a large-scale Web-based application (*E-type systems*) using System Dynamics.

Table 1 – The Laws of Software Evolution [2]

Laws of Software Evolution
Law I - Continuing Change: An E-type system must be continually adapted or it will become progressively less satisfactory in use
Law II - Increasing Complexity: As an E-type system evolves its complexity increases unless work is done to maintain or reduce it
Law III – Self Regulation: Global E-type system evolution processes are self-regulating
Law IV - Organizational Conservation: Average activity rate in an E-type process tends to remain constant over system lifetime or segments of that lifetime
Law V - Conservation of Familiarity: In general, the average incremental growth (growth rate trend) of E-type systems tends to decline
Law VI - Continuing Growth: The functional capability of E-type systems must be continually increased to maintain user satisfaction over the lifetime of the system
Law VII - Declining Quality: Unless rigorously adapted to take into account changes in the operating environment, the quality of an E-type system will seem to decline as it evolves
Law VIII - Feedback System: E-type evolution processes are multi-level, multi-loop, and multi-agent feedback systems

This paper presents the definition and initial assessment of such model to support *in silico* studies of software evolution (quality decay). Thus, such study aims to understand the mechanisms through which one software product is submitted when in a maintenance process in order to better accommodate the changes that have to be made. The observation of this evolution process would allow assisting in accommodating changes, providing the Software Engineer with a mechanism to support decision-making to get higher quality and maintainability in future versions of a software product, extending its useful life, and producing greater investment efficiency.

Apart from this introduction, this paper is organized as follows: Section 2 presents the basis to organize the software evolution observation model. Section 3 offers an explanation on how model confidence has been strengthened through a secondary study. In Section 4 the model assessment in an industrial software application is described. Finally, Section 5 presents conclusions, including the limitations and future work on the model.

2. SOFTWARE DECAY MODEL

Model building was done through the selection of software characteristics [10] representing both system quality and LSEs' features, aiming at providing a better understanding of what would affect software decay. The selection of software characteristics took into account the ISO 9126 [11] standard, as follow [4]:

- Size, the magnitude of artefacts produced in each life cycle phase of the proposed software (e.g., the amount of lines of code in the source code or the number of requirements in the requirements specification document);
- Periodicity, the time interval between each release version of a produced artefact (e.g., software or documentation versions);
- Complexity, the elements that can measure the structural complexity of an artefact (e.g., cyclomatic complexity of methods, or number of classes in the class diagram);
- Effort, the amount of work performed to produce a version of some artefact (e.g., measured in terms of man-hour or equivalent unit);
- Reliability, the number of defects identified per artefact in each software version;
- Maintainability, the time spent in identifying, removing and fixing defects.

After identifying the software's characteristics, each one of the LSE was interpreted according to its possible decay influence in the selected software characteristics. The results were represented through truth tables describing the expected behaviour of each software characteristic regarding a particular item of the LSE. It led to logically formulate a set of assumptions suggesting the behaviours that could be observed under the perspective of the existent linkage among the LSE and software characteristics. So, logical formulations describe software characteristic trends (increasing - ↑, decreasing - ↓ or no changing - ↔) [10] when some LSE can be observed in the software project, or not. So, these assumptions become the observational decay hypotheses regarding the LSE as described in Table 2. These hypotheses are used to verify the behaviour of the LSE on a given system, from the measurements collected for each software characteristic.

Table 2. Observing software decay through LSEs

Observational Hypotheses
(¬↑Periodicity **AND** ¬↓Effort) ⇒ **Continuing Change**
(↑Size **OR** ↑Complexity **OR** ↑Effort **OR** ↓Modularity **OR** ↓Maintainability) ⇒ **Increasing Complexity**
(¬↑Size **AND** ¬↓Reliability **AND** ¬↓Efficiency) ⇒ **Self Regulation**
(↔Effort **AND** ↔Efficiency) ⇒ **Conservation of Organizational Stability**
(↔Size **AND** ↔Complexity **AND** ↔Effort) ⇒ **Conservation of Familiarity**
(↑Size **AND** ¬↑Periodicity) ⇒ **Continuing Growth**
(↑Complexity **OR** ↑Effort **OR** ↓Modularity **OR** ↓Reliability **OR** ↓Maintainability) ⇒ **Declining Quality**
(Collection of relative measures to Size, Periodicity, Complexity, Effort, Modularity, Reliability, Efficiency, Maintainability) ⇒ **Feedback System**

3. STRENGHTENING MODEL REPRESENTATION

3.1. Study Protocol Overview

Generally, technical literature reviews are done through with informal and subjective methods for collecting and interpreting studies. Therefore, researchers may tend to selectively quote studies reinforcing pre-estimates, that is, following a chain of reasoning, giving bias to research results. On the other hand, the systematic literature review (SLR) focuses on executing a comprehensive and thorough search to find studies related to a formally defined research question. Furthermore, it uses well defined criteria to select studies, their quality assessment and also to present a summary of the results according to predetermined procedures [12].

According to Travassos et al. [13], the concept of *quasi* SLR represents a systematic literature review where the PICO's Comparison dimension [12] is represented by an empty set, thus reducing the possibility of executing meta-analysis or any kind of more elaborated research synthesis. Therefore, such reviews represent an intermediate level between mapping studies [14] and systematic literature reviews. The definition of a research protocol containing guidelines for the execution and delivery results of the *quasi* SLR, such as research question definition, inclusion, and exclusion criteria for study selection, criteria for assessing study quality and results, amongst others, give a high degree of formality to this type of secondary study. This formality brings great benefits to the community, as related to the research area under study, such as the reuse of the results achieved as a reference for other surveys, and especially the possibility of re-executing the research protocol for a possible evaluation of the results obtained or to update it.

Therefore, a *quasi* SLR research protocol was organized, to seek evidence on the linkage amongst software characteristics, thus reinforcing the existence of the relationships amongst them, including their mutual influence [8]. The goal is to reduce a threat to the validity of the study based on the number of relationships considered, together with the question that these relationships are processed on different levels of abstraction and at different moments in the lifecycle of a software product, as it is necessary to evaluate the results obtained in the observation of evolving systems and the reality presented by these systems over successive cycles of maintenance. A relationship between two software characteristics shall be considered to exist if it could be identified in any object-oriented software development stage [15]: Requirements Specification, High Level Design, Low Level Design, and Coding. The testing stage is not being directly considered as it generates information (failure detection) that will support the finding of defects to be fixed in some of the previous stages, which will generate possible evolution opportunities. Measurements allow the observation of software characteristics. Each measurement has been separately taken, according to the software development stage. For example, size may be measured as number of requirements at the Requirements Specification stage or as number of lines of source code in the Coding stage. More details on how measurements can be associated with each characteristic software at different stages of the process are described in [4].

So, model building [4] consisted of answering 15 different research (capturing relationships) questions, aiming at investigating the existence of linkages amongst each of the selected software characteristics.

Aiming to investigate all the research questions, considering each software development stage, a meta-protocol for the *quasi* SLR was produced [4]. Each pair $\{C_a,C_b\}$ of software characteristics was used in the composing of three questions (1 primary and 2 secondary):

$\mathbf{Q_i}$: Is there any influence between the software characteristics $\{C_a,C_b\}$ in the OO software development process?

> $\mathbf{Q_{i.1}}$: What is the direction of influence between the software characteristics $\{C_a,C_b\}$?
>
> > $\mathbf{Q_{i.1.1}}$: What is the intensity/rate of influence between the software characteristics $\{C_a,C_b\}$?

All the three questions for each pair of software characteristics (questions $\mathbf{Q_1..Q_{15}}$) were separately analyzed for the software development stage considered, as follows:

- $\mathbf{Q_1}$: *Size* and *Complexity*;
- $\mathbf{Q_2}$: *Size* and *Reliability*;
- $\mathbf{Q_3}$: *Complexity* and *Effort*;
- $\mathbf{Q_4}$: *Effort* and *Reliability*;
- $\mathbf{Q_5}$: *Complexity* and *Maintainability*;
- $\mathbf{Q_6}$: *Effort* and *Maintainability*;
- $\mathbf{Q_7}$: *Effort* and *Periodicity*;
- $\mathbf{Q_8}$: *Periodicity* and *Maintainability*;
- $\mathbf{Q_9}$: *Size* and *Effort*;
- $\mathbf{Q_{10}}$: *Size* and *Maintainability*;
- $\mathbf{Q_{11}}$: *Periodicity* and *Size*;
- $\mathbf{Q_{12}}$: *Complexity* and *Reliability*;
- $\mathbf{Q_{13}}$: *Periodicity* and *Complexity*;
- $\mathbf{Q_{14}}$: *Maintainability* and *Reliability*, and,
- $\mathbf{Q_{15}}$: *Periodicity* and *Reliability*.

The basic sources of information were represented by digital libraries including conferences and journals. Additionally, the search for conference proceedings whose themes are concerned with software maintenance or evolution was also considered, such as the *International Conference on Software Engineering (ICSE)*, *International Conference on Computer Science and Software Engineering (CSSE)*, *International Conference on Software Maintenance,* and *European Conference on Software Maintenance and Reengineering*.

EI Compendex and *Scopus* were used as the main search engines to look for technical papers published until October 2010. The reason for choosing them is the coverage they provide in the field. For instance, the *Scopus* search engine indexes the vast majority of papers published in the field of Software Engineering, such as, *IEEE*, *ACM, Cite Seer*, *Elsevier*, and *Springer*. The *quasi* SLR considered only sources available in the Web, in English, including theoretical papers, proof of concept, and especially experimental studies. The studies should also consider the relationship between software characteristics and metrics for *E-type systems*. Optionally, the articles should present influences between software characteristics, influence and direction of this intensity or rate of influence among the software characteristics.

Each research question consists of four searches related to the four development process stages. Thus, 60 different search strings were executed regarding questions $\mathbf{Q_{i,i=[1..15]}}$. The structure of the search strings e follows a pattern, considering the development stage and the corresponding software characteristics. Therefore, the search strings have many similar keywords which can affect the results with respect to the number of duplicate papers.

The execution of all search strings in both search engines returned a total of 6,753 papers, most of them duplicated. It led us to group them by research question. Even among the different questions, there were duplicate papers, as one same software characteristic was present in different research questions and some papers are concerned with several of the analyzed software characteristics. It represents a kind of expected behaviour that we were not able to eliminate. Nevertheless, the use of the JabRef tool (http://jabref.sourceforge.net/) to organize and classify the papers made this task easier.

To build the search strings the following general keywords were used: Software Characteristic, metric, relation, relationship, correlation, dependency, influence, effect, direction, primary study, experimental study, empirical study, intensity, and rate. For instance, (1) represents the search string for $\mathbf{Q_1}$ on coding stage:

(Relationship OR Relation OR Correlation OR Influence OR Dependence OR Effect OR Linkage) AND (size OR LOC OR {lines of code} OR {source lines of code} OR {methods per classes}) AND (complexity OR {Depth of Inheritance per Class} OR {Coupling between Objects} OR {Response for a Class} OR {Lack of Cohesion in Methods} OR {Children per Class} OR {Cyclomatic Complexity per Method}) AND (Codification OR Programming OR Building OR Construction OR Implementation OR Coding) AND ({Software Characteristic} OR {Software Metric} OR {Software Development Project} OR {software project} OR {software measure} OR {software measurement}). **(1)**

An information extraction guide was defined to keep track of the necessary data to answer the research questions and also to support paper assessment. The following data was extracted from the papers: title, author, source, publication date, type of study (experimental, case study, survey, technical report), category (conference or journal), context, applied technologies, stages of development involved, measures or software characteristics used, and description of influence (direction and rate or intensity) amongst the software characteristics analyzed.

A set of inclusion and exclusion criteria was defined for selecting the studies. The criteria consider papers written in English, available in the Internet and presenting studies on the relationship between software characteristics or correspondent metrics. Moreover, they should consider object-oriented software systems defined as *E-Type Systems* [9], and preferably present studies on the direction and rate or intensity of the relationship amongst software characteristics.

The process to select the studies was done by two experienced researchers and consisted of two analyses. The first one was guided by the reading of the title, abstract, and conclusions. The second and more detailed analysis consisted of a full reading. On both analyses, the inclusion and exclusion criteria of studies were applied.

With the removal of the duplicate papers, a total 884 papers remained that cleared the first analysis process [4]. After this, there remained 98 papers which cleared a second and more detailed analysis. Based on the results it was possible to choose 27 papers able for inclusion as possible evidence to support our model. Therefore, these papers passed through the information extraction process. The results and the complete definition of this process can be found in [16].

After information extraction, an assessment of the papers was performed. The assessment criteria applied to the papers selected were defined to identify whether methods or procedures were applied in order to provide higher quality study results. Eight items were defined as criteria for assessing the papers and they are related to the data analysis used in the study (identification and treatment of outliers), the application of methods of sensitivity or residual analysis, the use of appropriate statistical methods, the presentation of information on the projects used, methods of comparison applied, the accuracy of results and presentation of the methods used to achieve the results. Amongst the criteria used, those related to data treatment and applied statistical methods were considered as more important [16]. So, it resulted in 22 papers that could be accepted (5 papers were excluded). Table 3 shows the amount of evidence found for each pair of characteristics.

Table 3. Relation between research questions and evidence

Research Question	Evidence from paper
Q_1	[17] [18] [19] [20] [21]
Q_2	[18] [19] [22] [23] [24] [25] [26] [27] [28]
Q_3	[1] [21] [29] [30] [31]
Q_4	[32]
Q_5	[29] [30] [33] [34] [35] [36] [37] [38] [39] [40]
Q_6	[36] [41]
Q_7	[42]
Q_8	Not found
Q_9	[18] [30] [31] [43] [44] [45] [46] [47] [48] [49] [50] [50]
Q_{10}	[33] [34] [36]
Q_{11}	[24] [50]
Q_{12}	[18] [23] [30] [37] [52] [53] [54] [55]
Q_{13}	[56]
Q_{14}	[57]
Q_{15}	[57]

Each paper was evaluated according to the questions shown in Table 4, assigning a score per article, which was used to prioritize evidence to be considered in the model. Some questions produced few evidences, especially those related to software characteristics related to the process, such as reliability, where a few studies were found in the technical literature. Further details on the protocol for the *quasi* SLR used in this work and the study implementation are available in [16].

Table 4. Criteria for Paper Assessment

Questions
1. Is the data analysis appropriate? (Yes/No)
1.1. Was the data investigated to identify outliers and to assess distributional properties before analysis?
1.2. Was the result of the investigation used appropriately to transform the data and select appropriated data points?
2. Did the studies carry out a sensitivity or residual analysis? (Yes/No)
2.1. Were the resulting estimation models subject to sensitivity or residual analysis?
2.2. Was the result of the sensitivity or residual analysis used to remove abnormal data points if necessary?
3. Were accuracy statistics based on the raw data scale?
4. How good was the study comparison method?
5. Is it clear what projects were used to construct each model?
6. Is it clear how accuracy was measured?
7. Is it clear what cross-validation method was used?
8. Were all model construction methods fully defined (tools and methods used)?

3.2.Visualizing and Supporting Model Use

Based on evidence, a Cause and Effect Diagram [58] describing the model was built. It represents the basis of an infrastructure to allow the observation of software evolution [4]. This model is shown in Fig. 1. To support it, 25 evidence items from technical literature have been catalogued and evaluated {$\mathbf{Q_1..Q_7}$, $\mathbf{Q_9}$, $\mathbf{Q_{10}..Q_{15}}$}. No evidence has been found to support {$\mathbf{Q_8}$}, therefore there is no marked relationship between *Periodicity* and *Maintainability* in Fig. 1. All of the used evidence supports the existence of a correlation amongst the software characteristics. These correlations were seen through experimental studies using historical data on software projects, as reported in their original papers (references in Table 3). A similar approach has been used by [59] to create a model to support a project manager to trade-off the resources used against the outputs in a software project.

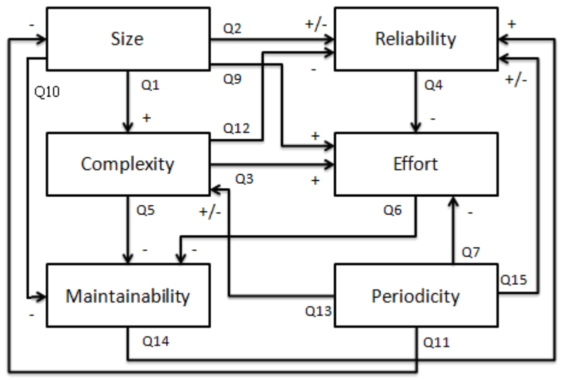

Figure 1. Cause and Effect Diagram from [4]

The use of a Cause and Effect Diagram does not imply causality in the presented relationships. According to its notation [58], a connection between two variables (software characteristics) indicates the existence of a relationship (possible influence) between them. The connection arrow indicates the relationship direction between software characteristics. The signals associated with each relationship indicate the suggested software characteristic trend (increasing +, decreasing -, increasing/decreasing depending on a threshold +/-) in that relationship. The signals were also obtained from the evidence found by reviewing each pair of software characteristics. For example, complexity positively influences effort, that is, the higher the complexity (e.g., measured by cyclomatic complexity) the greater the effort will be to develop or maintain the system. This way, the model allows observations of software evolution done over different software development stages, with the use of appropriate metrics for each stage. The observation will be done separately for each stage, that is, for a given stage only the same chosen metrics related to it must be consistently collected to allow consistency and contextualization.

Based on these principles, an environment was built comprising a set of activities for its operation. Initially, the software engineer collects data (representing the software characteristic measurements) from a real system in which one would like to observe the evolution process. It is important that the measurements are consistent and similar for a given system, keeping the same unit for a particular software characteristic. This information can be used to generate the equations used in the simulation with linear regression. It is appropriate to use linear regression, despite the possibility of a greater error margin on the basis of such observations being typically performed using semi-quantitative analysis for the analysis of data, where the trend is the focus, more than the precision of actual values. The generated set of equations can be computed by a tool providing support for dynamic models, such as, for example, System Dynamics, simulating the behaviour of software systems in evolution. Finally, through this tool, we can see the result of the decay process and the simulation software, also indicating the state of each one of the LSE, depending on the trends for behaviour of the software characteristics.

From this perspective, it is possible to perform *in virtuo*[1] and *in silico*[2] experimental studies. *In virtuo* studies are accomplished with the collection of metrics and analysis of the presented software characteristics from the observation of real systems. *In silico* studies are executed from the simulation of metrics and software characteristics behaviour, regarding the different software development process stages. Therefore, an environment has been constructed, as shown in Figure 2.

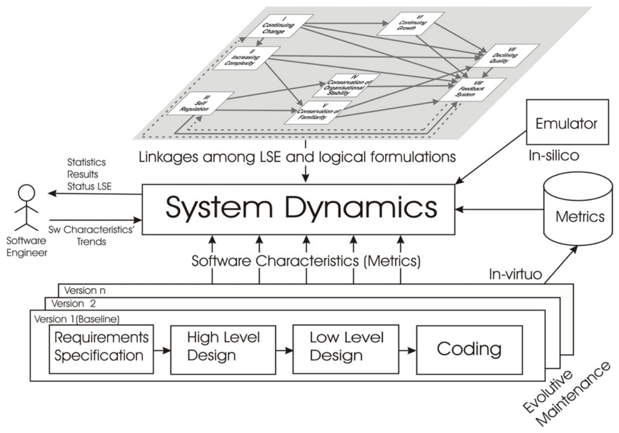

Figure 2. Architecture of the environment

1 *In virtuo* experiments: Experiments that involve interaction amongst participants and a computerized model of reality.

2 *In silico* experiments: Studies are marked for both subjects and real world, being described as computer models.

4. MODEL ASSESSMENT

This Section shows the results of an *in silico* study [4] using the model to track software decay. Historical data regarding a Web-based information system was used to extract the measurements for software characteristics used by the model. It represents a large-scale Web-based application controlling the business workflows of research projects, both in financial and administrative aspects. Geographically distributed teams took part in the development using Java and JSF platforms. The development team is stable, with about 12 developers. The development cycle was iterative and incremental, organized according to CMMI level 3 suggestions. Software quality is an ongoing concern, promoting the use of verification, validation, and testing techniques throughout the software development.

To support observation, 13 different system releases were considered. The historical data on these releases was available in version control system repositories, bug tracking services, and effort registration spreadsheets, whose measurements are relevant to the observation of system evolution.

This study analyzed the coding stage. The following measures were considered: *Size* in KLOC; *Periodicity* is the interval in days for delivering a new system release; *Complexity* is represented by the cyclomatic complexity of class methods; *Effort* is the total time spent (in hours) to develop a new release; *Reliability* is the number of defects identified per release; and *Maintainability* is the total time spent to identify, remove, and fix defects. Table 5 shows actual data on the 5 first system releases.

Table 5. Data Collected [4]

Version	1	2	3	4	5
Size	62,183	62,578	62,524	62,537	62,551
Periodicity	0	1	6	2	7
Complexity	11.338	11.407	11.391	11.391	11.392
Effort	6	0.25	30.75	13.75	1
Reliability	1	1	7	2	1
Maintainability	0	0.25	15.75	3.75	1

To increase the simulation process accuracy some statistical analyses were done on the historical data, through graph visualization, analysis of outliers and calculation of the measurements of central tendency, linear regression, and least squares. The standard deviation was cumulatively calculated for each software characteristic in each system release supporting the definition of the smallest number of necessary releases to calibrate the model. An outlier analysis was also done. To observe the trend for each software characteristic, scatter plots were constructed, as shown in Fig. 3, which portrays an increasing trend for *Size*.

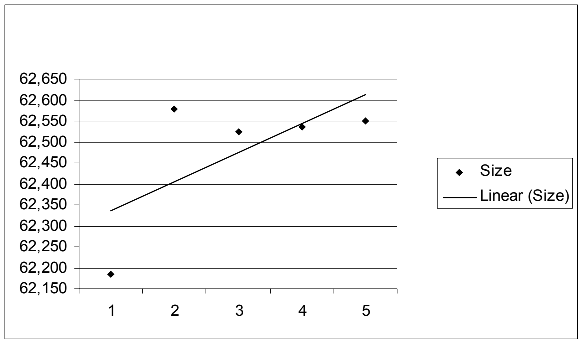

Figure 3. Scatter plot for software characteristic Size

After this, equations (linear regression + least squares method) were generated from historical data for each relationship involving the software characteristics presented in the model. For instance, **(2)** represents the equations for this system. The constants are the regression line slope calculated through linear regression, using the least squares method. The first one was calculated by considering *Size* and *Complexity* and the second used *Periodicity* and *Complexity* measurements:

*Size = 0.10 * Periodicity;*
*Complexity = 0.33 * Size + 0.05 * Periodicity;*
*Maintainability = 13.12 * Complexity + 0.17 * Effort + 3.87 * Size;*
*Reliability = 0.49 * Size - 0.04 * Periodicity + 0.28* Complexity + 0.13 * Maintainability;*
*Effort = 18.65 * Size + 2.82 * Periodicity + 57.82 * Complexity - 2.01 * Reliability;* (2)

With all equations and based on the Cause and Effect Diagram shown in Fig. 1, a System Dynamics model was built [4], as presented in Fig. 4.

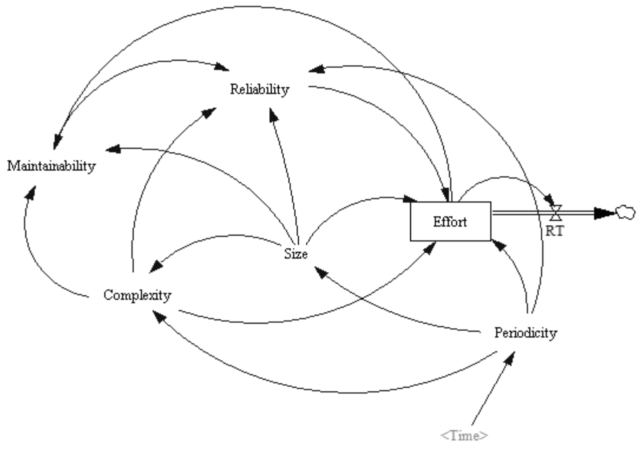

Figure 4. System Dynamics Model [16]

The simulation was mainly performed with the use of the Illium Software Evolution Simulation Tool [4], which uses the Illium Simulation Tool engine [60]. This engine was originally developed to support the simulation of risk analysis in software projects. The Illium engine had been used and evaluated in other studies [61] [62], involving the simulation of dynamic system models. Besides, considering the same System Dynamics model, Illium and an academic version of Vensim PLE [63] present equal results [16], demonstrating that Illium is also a feasible infrastructure to support System Dynamics simulations.

To assess the future behaviour of the evolving system and simulation results, new data was collected over eight new releases (Table 6), representing later versions not previously used to generate the equations.

Table 6. Collected Data from New Versions [4]

Version	1	2	3	4	5	6	7	8
Size	63,990	64,310	64,310	64,370	65,430	65,760	65,960	65,960
Periodicity	14	8	20	7	30	5	16	48
Complexity	11.57	11.63	11.63	11.64	11.85	11.93	11.96	11.96
Effort	4.75	22.50	2.00	29.50	53.50	10.50	3.50	25.75
Reliability	1	4	1	8	8	5	1	5
Maintainability	4.75	2.50	2.00	13.50	6.50	3.50	3.50	3.00

Again, the data was analyzed and scatter plots were constructed to observe the trend of each software characteristic compared with the simulated results. Table 7 shows these trends as observed behaviour, where Expected refers to the observed behaviour of the measures by linear regression, Simulated refers to the behaviour generated by the equations using system dynamics, and Observed is the actual behaviour of the measures collected in the new system releases. Forecast trends for *Size*, *Periodicity*, *Complexity*, *Reliability,* and *Maintainability* were maintained for the new software releases. It is important to emphasize that the purpose of observing the software evolution is to visualize curve trends, from a semi quantitative view, where the curve slope is generated by linear regression, proper for analysis [4]. The grey line in Table 7 highlights the difference amongst expected, simulated, and observed behaviours. Please note that expected results represent the software characteristic trend observed in isolation, as it means only its own historical dataset was used to suggest, based on data behaviour, something that could happen with that software characteristic in future releases.

Table 7. Simulation Results with Original Model [16]

Behaviour	Expected	Simulated	Observed
Size	↑	↑	↑
Periodicity	↑	↑	↑
Complexity	↑	↑	↑
Effort	↓	↑	↑
Reliability	↓	↓	↓
Maintainability	↑	↑	↑

In the first software releases *Effort* showed a downward trend. However, after the simulation process, a reversal in behaviour was seen, with an upward trend, compatible with the actual software characteristic behaviour, which can be seen in the new collected versions. All other behaviours were observed to be consistent with this information system behaviour [4].

These results may assist in decision-making as related to managing the development and maintenance of software processes. It is important to see that the results presented are restricted to the data set used in the experiment and it is not possible to generalize the results to other systems. However, the model is generic enough to be tried in other software projects.

In other words, the simulated behaviour for all characteristics corresponded to actual system behaviour. The results indicate that, for this system, the model of software evolution can be used to support the software development management, something that effectively happened in this industrial project.

5. CONCLUSIONS

This paper presented a model to support *in silico* studies of software evolution. The execution of a *quasi* SLR including 15 primary research questions contributed to identify 34 evidence items in the technical literature concerned with the existence of software characteristics relationships. The increased coverage given by the *quasi* SLR allowed strengthening the conceptual validity of the software evolution observation model, thereby reducing the threats to model construct validity.

The use of the model to support the simulation of a real software project was shown to be feasible and correct. Therefore, the managers of such project started to make use of the model to support their decision-making activities.

The results of the *quasi* SLR lead us to consider the relevance of each one of the relationships presented in the model, as there was no additional evidence to reaffirm all the relationships. Therefore, we would like to suggest, as future work, the examination of the existence of a minimum set of relationships to simplify the model, whilst keeping or increasing the quality of the results obtained from simulations based on it.

The model was built based on the analysis of 2-way interactions amongst the characteristics. Analyses of higher order interactions also represent an interesting future work to improve model confidence.

The execution of additional experiments with other data sets was not possible at this time due our difficulty in finding different data sets with available information (measures) for all the software characteristics presented in the model. This reduces the chance of generalizing the results, which can affect the external validity of the study.

The next steps of our research are concerned with using the model in performing additional *in virtuo* and *in silico* studies to evaluate the model's practical impact on new software projects.

6. ACKNOWLEDGMENTS

The authors would like to thank CNPq and CAPES for their supporting this research. We are also indebted to Profs. Barbara Kitchenham for her support to the initial discussions regarding this research, Márcio Barros for his System Dynamics support and Msc. Breno França for suggestions regarding this paper.

7. REFERENCES

[1] Bocco, M., Moody, D. and Piattini, M. 2005. Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. *Research Articles*. *J. Softw. Maint. Evol.* 17, 3 (May 2005), 225-246. DOI=10.1002/smr.v17:3 http://dx.doi.org/10.1002/smr.v17:3.

[2] Lehman, M. M. and Ramil, F. F. 2002. Software Evolution and Software Evolution Processes. *Ann. Softw. Eng.* 14, 1-4 (December 2002), 275-309. DOI=http://dx.doi.org/10.1023/A:102055752590.

[3] Madhavji, N. H., Ramil, J. F., and Perry, D. 2006. Software Evolution and Feedback: Theory and Practice. *John Wiley & Sons*.

[4] Araújo, M. A. 2009. *Um Modelo para Observação da Evolução de Software*. Doctoral Thesis. Federal University of Rio de Janeiro.

[5] Smith, N., Ramil, J.F., 2002. Qualitative Simulation of Software of Evolution Process. In: *WESS´02 Eighth Workshop on Empirical Studies of Software Maintenance*.

[6] Stopford, B., Counsell, S., 2008. A Framework for the Simulation of Structural Software Evolution. *ACM Transactions on Modeling and Computer Simulation*, Vol. 18, No. 4, Article 17.

[7] Lehman, M. M. 1980. Programs, Life Cycle and the Laws of Software Evolution. *Proc. IEEE Special Issue on Software Engineering*, vol. 68, no. 9, pp. 1060 -1076.

[8] Araújo, M. A. P., Travassos, G. H. 2008. A System Dynamics Model based on Cause and Effect Diagram to Observe Object-Oriented Software Decay.*Technical Report ES-720/08*, COPPE/UFRJ.

[9] Lehman, M. M. and Ramil, F. F. 2003. Software evolution: background, theory, practice. *Information Processing Letters*, v. 88 n. 1-2, p. 33-44, 2003. DOI= http://dx.doi.org/10.1016/S0020-0190(03)00382-X.

[10] Araújo, M. A. P., Travassos, G. H.. Kitchenham, Barbara. 2006. Evolutive Maintenance: Observing Object-Oriented Software Decay. *Technical Report*, COPPE/UFRJ.

[11] ISO 9126-1, 1997, International Standard. Information Technology – Software Quality Characteristics and Metrics – Part 1: Quality Characteristics and Sub-Characteristics.

[12] Pai M., McCulloch M., Gorman J., Pai N., Enanoria W., Kennedy G. 2004. Systematic reviews and meta-analysis: an illustrated step-by-step guide. *Natl Med J India*. 2004;17:86–95.

[13] Travassos, G. H., Santos, P. S. M., Mian, P., Dias Neto, A. C., Biolchini, J. 2008. An Environment to Support Large Scale Experimentation in Software Engineering. *In: Proc. of the IEEE International Conference on Engineering of Complex Computer Systems*, ICECCS 2008, p. 193-202.

[14] Kitchenham, B.A., Charters, S. 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. *Technical Report EBSE-2007-01*.

[15] Travassos, G.H., Shull, F., Carver, J. 2001. Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language. *Advances in Computers*, San Diego, v.54, n.1, p.35 - 97.

[16] Monteiro, V. F. 2011. Infraestrutura Computacional para Observação de Evolução de Software. Master Thesis. *Federal University of Rio de Janeiro*.

[17] Aggarwal, K., Singh, Y., Kaur, A. and Malhotra, R. 2006. Empirical Study of Object-Oriented Metrics, in *Journal of Object Technology*, vol. 5. no. 8, November-December 2006, pp. 149-173.

[18] Heijstek, W., Chaudro, M.R.V. 2009. Empirical Investigations of Model Size, Complexity and Effort in a Large Scale, Distributed Model Driven Development Process. Software Engineering and Advanced Applications, 2009. SEAA '09. *35th Euromicro Conference on 2009* , Page(s): 113 – 120. DOI=http://dx.doi.org/10.1109/SEAA.2009.70.

[19] Basili, V.R., Perricone, B.T., 1984. Software Errors and Complexity: An Empirical Investigation. *Communications of the ACM*, 27, 42 – 52.

[20] Ebert, C., 1996. Evaluation and application of complexity-based criticality models. In: *International Software Metrics Symposium, Proceedings*, 174 – 184.

[21] Misic, V.B., Tesic, D.N., 1998. Estimation of effort and complexity: An object-oriented case study. *Journal of Systems and Software*, 41, 133 – 143.

[22] El Emam, K. , Benlarbi, S., Goel, N., Melo, W., Lounis, H., Rai, S. N. 2002. The Optimal Class Size for Object-Oriented Software. *IEEE Transactions on Software Engineering*, v.28 n.5, p.494-509, May 2002. DOI= http://dx.doi.org/10.1109/TSE.2002.1000452.

[23] English, M. Exton, C., Rigon, I., Cleary, B. 2009. Fault detection and prediction in an open-source software project. *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, May 18-19, 2009, Vancouver, British Columbia, Canada. DOI= http://doi.acm.org/10.1145/1540438.1540462.

[24] Fenton, N.E., Neil, M., Marsh, W., Hearty, P., Radlinski, L., Krause, P. 2008. On the effectiveness of early life cycle defect prediction with Bayesian Nets. In *Proceedings of Empirical Software Engineering*. 2008, 499-537. DOI= http://dx.doi.org/10.1007/s10664-008-9072-x.

[25] Leszak, M. 2005. Software Defect Analysis of a Multi-release Telecommunications System *In Product Focused Software Process Improvement* (2005), pp. 98-114.

[26] Zhang, H. 2009. An investigation of the relationships between lines of code and defects. ICSM, pp.274-283, 2009 *IEEE International Conference on Software Maintenance*, 2009.

[27] Selby, R.W., 1990. Empirically based analysis of failures in software systems. *IEEE Transactions on Reliability*, 39, 444 – 454.

[28] Malaiya, Y.K., Denton, J., 2000. Module size distribution and defect density. In: *Proceedings of the International Symposium on Software Reliability Engineering, ISSRE*, 62 – 71.

[29] Darcy, D., Kemerer, C., Slaughter, S., and Tomayko, J. 2005. The Structural Complexity of Software: An Experimental Test. *IEEE Trans. Softw. Eng.* 31, 11 (November 2005), 982-995. DOI=http://dx.doi.org/10.1109/TSE.2005.130.

[30] Kozlov D., Koskinen J., Sakkinen M., Markkula J. 2008. Assessing maintainability change over multiple software releases. In *Journal of Software Maintenance and Evolution: Research and Practice* 2008; 20:31–58. DOI=http://dx.doi.org/10.1002/smr.v20:1.

[31] Sentas, P., Angelis, L., Stamelos, I. 2008. A Statistical Framework for Analyzing the Duration of Software Projects. *Empirical Software Engineering (Springer)*. 13:147–184. DOI=http://dx.doi.org/10.1007/s10664-007-9051-7.

[32] Bianchi, A. Caivano, D., Lanubile, F., Visaggio, G., 2001. Evaluating Software Degradation through Entropy. *IEEE*.

[33] Canfora, G., García, F., Piattini, M., Ruiz, F., Visaggio, C. A. 2005. A family of experiments to validate metrics for software process models. *Journal of Systems and Software,* v.77 n.2, p.113-129, August 2005. DOI=http://dx.doi.org/10.1016/j.jss.2004.11.007.

[34] Cruz-Lemus, J., Genero, M., and Piattini, M. 2007. Using Controlled Experiments for Validating UML Statechart Diagrams Measures. In *Software Process and Product Measurement*. Lecture Notes In Computer Science, Vol. 4895. Springer-Verlag, Berlin, Heidelberg 129-138. DOI=http://dx.doi.org/10.1007/978-3-540-85553-8_11.

[35] Poels, G., Dedene, G. 2001. Evaluating the Effect of Inheritance on the Modifiability of Object-Oriented Business Domain Models. In *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering* (CSMR '01). IEEE Computer Society, Washington, DC, USA.

[36] Riaz, M.; Mendes, E.; Tempero, E. A. 2009. Systematic Review of Software Maintainability Prediction and Metrics. *Proceeding ESEM '09 Proceedings of the* 2009 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, Washington,

DC, USA, 367-377. DOI= http://dx.doi.org/10.1109/ESEM.2009.5314233.

[37] McCabe, T. J., 1976. A Complexity Measure. *IEEE Transactions on Software Engineering*.

[38] Lindell, J., Hagglund, M., 2004. Maintainability Metrics for Object Oriented Systems. *Software Quality*.

[39] Koten, C., Gray, 2005. An Application of Bayesian Network for Predicting Object Oriented Software Maintainability. *The Information Science Discussion Paper Series*. University of Otago.

[40] Aggarwall, K.K., Singh, Y., Kaur, A., Malhotra, R., 2006. Application of Artificial Neural Network for Predicting Maintainability using Object Oriented Metrics. *Transactions on Engineering, Computing and Technology Volume*.

[41] Nikora, A. P., Munson, J.C., 2003. Developing Fault Predictors for Evolving Software Systems. In: *Proc. Ninth International Software Metrics Symposium – METRICS´03*.

[42] Mockus, A. Weiss, D., Zhang, P., 2003, Understanding and Predicting Effort in Software Projects. In: *ICSE´03*.

[43] Dolado JJ. 2001. On the problem of the software cost function*. Inform Software Technology* 2001;43:61-72. DOI= http://dx.doi.org/10.1016/j.advengsoft.2004.10.001.

[44] Ferrucci, F., Gravino, C., and Di Martino, S.. 2008. A Case Study Using Web Objects and COSMIC for Effort Estimation of Web Applications. In *Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications* (SEAA '08). IEEE Computer Society, Washington, DC, USA, 441-448. DOI=http://dx.doi.org/10.1109/SEAA.2008.

[45] Tsunoda , M., Monden , A., Yadohisa, H., Kikuchi, N., Matsumoto, K. 2009. Software development productivity of Japanese enterprise applications. In *Information Technology and Management*, v.10 n.4, p.193-205, *December* 2009. DOI= http://dx.doi.org/10.1007/s10799-009-0050-9.

[46] Zhang, J. 2008. The Establishment and Application of Effort Regression Equation. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02* (CSSE '08), Vol. 2. IEEE Computer Society, Washington, DC, USA, 11-14. DOI=http://dx.doi.org/10.1109/CSSE.2008.726.

[47] Waltson, C. E., Felix, C.P., 1977, A Method of Programming Measurement and Estimation. *IBM System Journal*.

[48] Boehm, B., 1981. Software Engineering Economics. *Prentice Hall*.

[49] Bailey, J. W., Basili, V., 1981. A Meta-Model for Software Development Resource Expenditures. *IEEE*.

[50] Boehm, B., 1984. Software Engineering Economics. *IEEE Transactions on Software Engineering*.

[51] Premraj, R., Shepperd, M., Kitchenham, B., Forselius, P., 2005. An Empirical Analysis of Software Productivity over Time. In: *11th IEEE International Software Metrics Symposium (METRICS 2005)*.

[52] Aggarwal, K., Singh, Y., Kaur, A. and Malhotra, R. 2007. Investigating effect of design metrics on fault proneness in object-oriented systems. *Journal of Object Technology*. v6 i10. 127-141.

[53] Ferneley, E. H. 1999. Design metrics as an aid to software maintenance: an empirical study. *Journal of Software Maintenance* 11, 1 (January 1999), 55-72. DOI= http://dx.doi.org/10.1002/(SICI)1096-908X(199901/02)11:1%3C55::AID-SMR184%3E3.3.CO;2-F.

[54] Schneidewind, N., Hinchey, M. 2009. A Complexity Reliability Model. In *Proceedings of the 20th IEEE international conference on software reliability engineering* (ISSRE'09) on. page(s): 1 – 10.

[55] Basili, V., Briand, L., Melo, W., 1995. A Validation of Object Oriented Design Metrics as Quality Indicators, *Technical Report*, Univ. of Maryland, Dep. of Computer Science, College Park, MD, 20742 USA.

[56] Munson, J.C., 1996. Software faults, software failures and software reliability modeling. *Information and Software Tecnology*.

[57] Schneidewind, N., 1999. Measuring and Evaluating Maintenance Process using Reliability, Risk and Test Metrics. *IEEE Transactions on Software Engineering*.

[58] Forrester, J. W., Industrial Dynamics. *The M.I.T. Press*, Cambridge, Mass., 1961.

[59] Fenton, N., Marsh, W., Neil, M., Cates, P., Forey, S., Tailor, M., 2004. Making Resource Decisions for Software Projects, icse, pp.397-406, *26th International Conference on Software Engineering (ICSE'04)*.

[60] Barros, M.O., Werner, C. M. L., Travassos, G.H. 2000. *Illium: Uma ferramenta de Simulação de Modelos Dinâmicos de Projetos de Software*, In XIV Simpósio Brasileiro de Engenharia de Software, Caderno de Ferramentas, p. 355-358, João Pessoa, PB, Brasil.

[61] Barros, M. O., Werner, C. M. L., Travassos, G. H. . A System Dynamics Metamodel for Software Process Modeling. *Software Process Improvement and Practice*, 2003.

[62] Barros, M. O., Werner, C. M. L., Travassos, G. H. 2004. Supporting Risk Analysis on Software Projects. *Journal of Systems and Software*.

[63] Vensim PLE 2010. Ventana Systems, Inc. Avaiable at: <http://www.vensim.com>.