Predicting Software Project Size using Project Generated Information

Márcio de O. Barros
Postgraduate Information Systems Program – UNIRIO
Av. Pasteur 458, Urca – Rio de Janeiro, RJ – Brazil
marcio.barros@uniriotec.br

Abstract. In this paper we present a simulation based approach to predict the expected probability distribution that describes the size of a software project in a given period in the future. Since the simulation strongly depends on historical information, we propose the collection of such data from version control systems, which are well-known and widely used in the industry. We discuss the information collection process and the simulation model that is built and executed based on such data. Finally, we present a case study in which we apply the proposed approach to estimate the size of a large software project on a three month time frame, comparing simulation results with other estimation procedures.

Keywords: software estimation, software economics, measurement and empirical Software Engineering.

1 Introduction

Size prediction is a hard problem for software projects. Many are the external and internal forces that can influence software growth, and the combined effect of such forces may prove difficult to model. Nevertheless, estimating future size is paramount as an early warning sign of missing milestones and the need to negotiate adjustment to project schedule and cost baseline. By predicting the size of a software project in the near future, a manager can identify whether the development team is productive enough to attain the functionality necessary to meet the next milestones within the desired time frame. So, the manager can plan team allocation on maintaining the software assets that are already constructed and the new functions to be built.

Two perspectives are usually explored when researchers propose a software project prediction method: (a) modeling the project and its influence factors in great detail, yielding complex simulation models that can be executed under distinct scenarios to provide insight on the future of the software project [1] [2] [3]; or (b) selecting a small group of relevant influence factors, deriving analytical formulations that explain the attributes under estimation based on data from past projects, and applying these equations to estimate the project at hand [4] [5].

While the first perspective demands knowledge about the software project to build the simulation models, the second perspective requires a selection criteria by which past projects are chosen to provide data to estimate the new one. Usually, this selection criterion is based on similarity, a subjective and ambiguous term whose interpretation may vary according to the different concerns people present about the project. Also, external drivers (such as consumer’s expected delivery date and budget restrictions) may influence the selection of “similar” projects, favoring those whose data yield estimations for the current project within a feasible zone. Finally, there may be issues regarding which data about a past project is available, how such projects were characterized, and how the information was stored. Diversity on data formats, technology used, programming language, development team, environment, and so on, may require adjustments on the available datasets in order to make them useful for the new project.

In this paper, we propose a simulation based approach to predict the expected probability distribution for the size of a software project in a given period, based on information collected from the project itself. Our fundamental assumption is that the most similar project to the one requiring estimation is the project itself. Thus, past information about the project is probably the best estimator for its future performance. Version control systems, like CVS [6] or SubVersion [7], commonly used in many industrial software projects, provide the required information. The approach is related to the second project prediction perspective, since relevant factors collected from past data are used to predict the future of the software project. A case study regarding the application of the approach is presented and its limitations are addressed.

The remaining of this paper is organized in five sections. Section 2 presents concepts and definitions that support the collection of information from version control systems to feed the simulation model. Section 3 presents the proposed simulation model and how results can be drawn from it. Section 4 presents a case study where the proposed approach is applied to a large software project. Section 5 presents related work. Section 6 addresses some limitations of the proposed approach and, finally, section 7 draw conclusions and directions for future work.

2 Collecting Information from Version Control

Configuration management tools have become well known and widely accepted by the software industry [8]. The availability of free source tools for version control, the growth of outsourcing and geographically distributed development, and the need for larger development teams drive the use of version control systems in medium and large software projects.
Version control systems store every version of relevant artifacts that compose a software project, thus maintaining a history of changes done throughout the software life-cycle. Version information, such as comments about the changes that were made, completion date and developer in charge, are attached to each version of an artifact. Version control systems are commonly used to allow navigation and retrieval of different versions of source code modules, but research in the field is evolving to allow requirements, design documents, project plans, and every other document to be managed by these systems.

Therefore, over the years the central repository of a version control system becomes a rich source of time-related information about a software project. After some development time, the repository is populated and the information it holds can be used to estimate relevant attributes for the project. So, instead of relying on “similar projects”, our approach uses information from the project itself to support estimation.

Before describing the simulation process that yields the probability distribution for project size in a given period, we need to formalize the information structure under the control of a version control system. A version control system manages several software project repositories. We define a software project repository (SPR) as the finite set of files (Fi) that compose a software project:

$$\text{SPR} = \{F_i \mid 1 \leq i \leq N\}$$

Each composing a software project repository is described by the directory in which it is located and an ordered set of reviews (Ri). Each review (Rij) is described by a unique identifier, the developer who committed the review to the central repository, the moment when the commit operation was executed, a comment describing the changes, and the resulting source code.

$$F_i = (\text{directory}, R_i)$$
$$R_{ij} = \{R_{ij} \mid 1 \leq j \leq M\} : R_{ij}.time < R_{ik}.time, \forall j < k$$
$$R_{ij} = (\text{id}, \text{developer}, \text{time}, \text{comment}, \text{code})$$

Based on such definitions, we define the size function, which represents the size of a given file review. The function maps a concept from the space defined by the reviews of a given file to the space of natural numbers, where size is defined.

$$\text{size}(R_{ij}) : R_i \Rightarrow N$$

We also define the current operator, which returns the last review committed in a given moment for a given file. It allows us to define the size function against time, measuring the size of the current review in that moment.

$$\text{current}(F_i, t) = R_{ij}$$
$$: R_{ij}.time \leq t, R_{ij} \in F_i$$
$$: \not \exists R_{ik} : R_{ik}.time < R_{ij}.time \leq t, \forall 1 \leq k \leq M$$
$$\text{size}(F_i, t) = \text{size}(\text{current}(F_i, t))$$

Finally, we define the diff and growth operators. The first calculates the nominal size change observed from time t1 to t2, t1 < t2, while the second calculates the percentile size change in the period.

$$\text{diff}(F_i, t_1, t_2) = \text{size}(F_i, t_2) - \text{size}(F_i, t_1)$$
$$\text{growth}(F_i, t_1, t_2) = \text{diff}(F_i, t_1, t_2) / \text{size}(F_i, t_1)$$

The selection of a metric to act as the size function is not trivial. Lehman suggests counting the number of source files as a size measure for large software projects [9], while Godfrey and Tu [10] use lines of source code to measure size. Moreover, [11] has shown that the number of source code lines is high correlated to the number of source files in large software projects and, therefore, these metrics grow at roughly the same rate. We decided to use lines of code because most size estimation procedures are directly related to this measure, such as [4, 5].

Nevertheless, collecting information from version control systems is a time consuming activity [12] that must be automated to scale to large projects with many thousands of files. We have developed a framework, namely JCVS, which is able to collect information from version control repositories and store such data to XML files or database tables. These are easier to query for project specific information than the poorly structured textual files that compose the version control repository. In the proposed approach, our major interest is related to the number of lines of code in the files’ reviews, but the framework is capable of retrieving the source code itself and all the attributes presented in the former formulations.

3 The Simulation Model

The historical growth data about the files composing a software project can be analyzed as a time series, that is, a set of values observed in successive time intervals. Converting time series to probability distributions (by computing the number of times each value is repeated in the series) is a practical procedure to highlight the series’ descriptive characteristics, such as average value and variance. Moreover, by analyzing the correlations among time series, we can describe how they behave together, that is, how a change in one series is reflected in the other.

We propose the use of Monte Carlo simulation to estimate the size of a software project in a given period based on correlated probability distributions derived from time series collected from the project’s version control system. Monte Carlo simulation is a sampling technique that estimates probability distributions for one or more results, given distributions for a set of parameters. The simulation process requires: (a) a finite set of parameters with known probability distributions; (b) a finite set of results, whose probability distributions will be estimated; and (c) a model that states how given parameter values yield a value for each result. Monte Carlo simulation consists in several cycles, each generating a value for each parameter (according to its probability distribution), calculating results according to the rules and relations prescribed in the model, and annotating these results. After thousands of such cycles, thousands of values are
annotated for each result and their probability distribution can be derived from these values. Monte Carlo simulation is popular because it allows parameters to be described by any type of probability distribution (such as the normal or beta distributions) and model development does not require specific knowledge on statistics or how to apply operations upon probability distributions.

In our proposed estimation method, the parameters are parts of the software project, being characterized by probability distributions estimated from growth time series extracted from version control systems; the single result is the observed growth of the project; and the model prescribes that the growth of the project is equal to the sum of the nominal growth of its parts divided by project size. So, the definition for both parameters and model depends on how the project is broken into parts. Two restrictions apply to this decomposition: (i) it must be possible to collect growth time series for each part from version control systems; and (ii) high correlated time series must be grouped and treated as a single part1.

The first restriction allows two decomposition strategies: to take each file composing the project as a part or to take selected groups of files as a part. However, the second restriction eliminates the first strategy, since for large projects there would be thousands of composing files. In such a situation, there is a large probability that there will be any two files, F1 and F2, so that changes to F1 will be coincident to changes to F2 over time. This results in high correlation among these time series, breaking the second restriction. Thus, the remaining option is to divide the project into file groups and treat each group as a part.

In the context of the proposed method, we define a component as a group of files that are logically related to each other in a project (for instance, they represent distinct design aspects for the same concept or process). The project must be divided into a set of complementary components (Ck). We suggest using of the project’s directory structure as a starting point for the division, refining it according to the distribution of project features among the source code distribution units (for instance, packages in an UML model). The final set of components must be so that each and every file in the repository pertains to one and only one component.

\[
\begin{align*}
\text{SPR} &= \{C_k | 1 \leq k \leq Z\} \\
C_k &= \{F_i | 1 \leq i \leq N\} \\
&\quad \because \forall F_i \in \text{SPR} \Rightarrow \exists C_k: F_i \in C_k, 1 \leq i \leq N, 1 \leq k \leq Z \\
&\quad \because \forall F_i \in \text{SPR}, F_i \in C_k \Rightarrow \text{not } \exists C_l: F_i \in C_l, 1 \leq i \leq N, 1 \leq k, 1 \leq Z; l \neq k
\end{align*}
\]

Given these definitions, we can define the size function and the \textit{diff} and \textit{growth} operators for components. The size function is defined below. The definition for the \textit{diff} and \textit{growth} operators is straightforward.

\[
\text{size}(C_i, t) = \sum_{i=1}^{N} \text{size}(F_i, t) \quad \therefore F_i \in C_k, 1 \leq i \leq N
\]

After dividing the project into components, we collect a growth time series for each component from the version control system. These growth time series are calculated by applying the following process: (a) build a time series for the size of each project file, collecting these data directly from the version control system, as presented in section 2; (b) sum up the time series for the files that compose each component, yielding a time series for the size of each component; (c) apply the growth operator in a weekly basis over each component size time series, resulting in a weekly growth time series for each component.

The weekly basis was selected since we expect that our approach will be used in medium or large size projects, where development time is measured in months or years. Using a monthly basis would leave few data points to build the component’s probability distribution, while a day seems too small a period to measure changes in components’ sizes. Further investigation is required to determine if the weekly basis can be generalized to any project, independent of its size or any other characteristic.

Next, the growth time series for each component must be analyzed to identify and eliminate outlier points. Such outliers are common in software development, and they are discernible as ripples in the growth time series as files are reused into the component (upward ripple), files are transferred to another component (downward ripple), developers delay the commitment of large changes to the central repository (upward ripple), or third-party code is put under version control (upward ripple). We observed that such ripples are more common early in the project life-cycle or during the transition phase of a service acquisition (such as third-party development). Since they do not represent the expected behavior for the project, these outliers must be eliminated before simulation.

Each weekly growth time series is then converted to a probability distribution that describes the component in the simulation process. Correlations among these time series are calculated and organized in a symmetric matrix where each row and column represents a component and each cell conveys the correlation among the components in its row and column. After sampling a growth value for each component in a simulation cycle, such values are applied to the correlation matrix in order to represent the joint behavior of growth in components. Thus, we have a set of correlated parameter values.

The simulation process also depends on the estimation period, that is, the number of intervals after a given date on which project size will be estimated. As data collection is established in a weekly basis, the estimation period (\(\Delta\)) is also expressed in weeks. Each simulation cycle samples correlated parameter values \(\Delta\) times, yielding \(\Delta\) random weekly growths for each component (\(G_i\)). The size of the component after the estimation period is given by:

1 When using Monte Carlo sampling with correlated time series, their correlations are organized in a symmetric matrix that must be positive defined. This property cannot be observed if two or more series present high correlation.
\[ size(C_i, T + \Delta) = size(C_i, T) \cdot \prod_{i=1}^{\Delta} (1 + G_i) \]

After simulating each component size \( \Delta \) weeks ahead, project size is estimated by summing up the sizes of its components. So, each simulation cycle yields an estimated project size and an estimated size for each component. By executing thousands of cycles, the proposed approach yields a probability distribution for project size and a probability distribution for each component’s size.

### 4 Case Study

This section describes the application of the proposed approach to a large scale system. The system is used by financial institutions and financial departments of non-financial companies to identify, analyze, and manage investments in market, over-the-counter, and hedge assets. It is under development since early 2005, with an average team of ten developers, using object-oriented and software component technologies. The software is developed using an incremental process and is currently composed of about 500 KLOC distributed along 1,931 source code files.

In this case study we were interested in addressing the ability of the proposed approach to estimate project size in the future, comparing the estimated size to the observed growth. We had version control data for the project since February, 2005 to September, 2007. We designed the case study so that such data was separated into two samples: data from February, 2005 to June, 2007 was used to parameterize the simulation model, while data from July, 2007 to September, 2007 was used as a test sample, being compared to results achieved from simulation.

Based on an assessment of its directory structure and a meeting with the system architect, the software project was decomposed into fifteen components. Six of these are under development since the project started (February, 2005), five others were started a year later (February, 2006), and the last four components were started about a year and a half after development was commenced (November, 2006). Though some components had version control data available since February, 2005, we decided to collect information only for the period when all components were under development (that is, from November, 2006 to September, 2007). This allowed us 34 weeks of sample data and 12 weeks of test data. Also, this period would better reflect the actual productivity, since the development team was smaller in the early stages of the project life-cycle. Table 1 presents the size (in KLOC) of each component after the 34 weeks of sample data (first row) and after the 12 weeks of test data (second row).

<table>
<thead>
<tr>
<th>Component</th>
<th>aux</th>
<th>sdbb</th>
<th>sasf</th>
<th>sinf</th>
<th>srep</th>
<th>sdec</th>
<th>svar</th>
<th>scfr</th>
</tr>
</thead>
<tbody>
<tr>
<td>aux</td>
<td>21.7</td>
<td>45.0</td>
<td>51.7</td>
<td>53.7</td>
<td>6.1</td>
<td>23.8</td>
<td>28.0</td>
<td>20.7</td>
</tr>
<tr>
<td>sdbb</td>
<td>24.9</td>
<td>48.1</td>
<td>51.8</td>
<td>54.5</td>
<td>14.0</td>
<td>28.2</td>
<td>33.7</td>
<td>21.3</td>
</tr>
</tbody>
</table>

Next, we created the growth time series for the components, as presented in Section 3. By visually inspecting these time series, we observed some outliers and proceeded with an investigation with the system architect to understand whether these outliers were valid values that should be accounted for in the simulation or there were reasons for these large growth variations that should not repeat in the future. Therefore, we conducted a qualitative outlier elimination based on information gathered through interviews with the project architect, instead of a model based, quantitative outlier elimination procedure. This investigation revealed important issues about project components:

- The six components built from February, 2005 (aux, sasf, sinf, sdec, svar, and scfr) were developed by reusing code from a previous version of the same system. So, their early development stages show high growth (thousands of KLOC added in a single week), which was not compatible with the effort dedicated to the components and could only be related to the inclusion of reused code. Therefore, data from the early development stages should be suppressed from the analysis. This was not an issue to the current case study (which used information from November, 2006 on), but was recorded as a lesson learned for field data collection;

- Two components (srep and irep) were developed by an external company, being integrated to the project’s version control system later in their development process. So, the time series for these components present large ripples as new code is integrated monthly (due to a staged delivery schedule), instead of a smoother daily integration. Moreover, the maintenance of these components was transferred to the company within the test data period. So, the large growth ripples ceased and a moderate growth behavior was observed. This could not be projected by the components’ time series, since they reflected a third-party development period. We decided to estimate project size without these components, thus eliminating their series from analysis;

- A component (aux) presented a “ladder style” time series: a week presenting high growth was followed by several weeks without development (that is, growth very close to zero). By enquiring the architect about this component, we discovered that it was composed by utilities classes, many of them reused from previous projects. The reuse of previous code justified the “ladder-effect” in the time series, as complete, quality code from other projects was injected into the version control repository. We decided to keep the component under analysis, eliminating a single very large ripple that made the component’s size double in a single week.

After removing the outliers from the time series, we generated their probability distributions and calculated
their correlation matrix. The multivariate distribution was submitted to the simulation process, yielding future size distributions for each component. The distributions after 10,000 simulations are available at the URL http://www.uniriotec.br/~marcio.barros/seke2008/index.html.

Figure 1 presents the probability distribution for project size 12 weeks after the sample data period, related statistical information, and a comparison to the observed size. The final project size (vertical line at 454.5 KLOC, excluding the srep and irep components) is within the boundaries of the probability distribution and close to the region determined by its first standard deviation from the average (469.7 to 510.1 KLOC).

<table>
<thead>
<tr>
<th>Information</th>
<th>KLOC</th>
<th>Information</th>
<th>KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed Size</td>
<td>454.5</td>
<td>Estimated Size</td>
<td>489.9</td>
</tr>
<tr>
<td>Std Deviation</td>
<td>20.2</td>
<td>Median</td>
<td>487.3</td>
</tr>
<tr>
<td>1st Quartile</td>
<td>465.5</td>
<td>3rd Quartile</td>
<td>511.8</td>
</tr>
</tbody>
</table>

Figure 1 - Probability distribution for future project size and related statistics

We compared this result with a projection of system size after 12 weeks using the growth of the whole system as a single time series. Over the 34 sample data points, the average growth for the whole system was about -0.91%, which yields an expected size of 376.8 KLOC after the 12 weeks of test data. Thus, the error between the simulation based estimation and the real system size after the test period was about 7%, while a projection of system size over the same period based on a single system growth time series would generate an error rate of about 17%.

We also compared simulation results to estimation based on exponential weighting the growth time series for the whole system, a usual estimation procedure in time series analysis. Exponential weighting uses a polynomial based on a factor (λ) by which observations are weighted. The polynomial is built so that recent observations have more weight than older observations. The estimated system growth (EG) is calculated as shown below.

\[ EG = \left( G_1 + \lambda G_{k-1} + \cdots + \lambda^N G_{k-N} \right) / \left( 1 + \lambda + \cdots + \lambda^N \right) \]

\[ : G_{k,K} \text{ observed system growth on time t-K} \]

By varying the lambda factor from 100% to 70%, we observed that estimation error for system size grows from 17% to 20%. Thus, correlations among component growth time series play a fundamental role to estimation, which is not captured by analyzing the single system growth path.

Analyzing the distribution for component sizes, we observed that svar presented higher growth than expected from past information, idbb presented lower growth, and idec presented less growth variations than observed in the time series. Questioning the architect about such results, we were informed that new features were recently added to the svar component (leading to higher growth than expected), and that sample data captured a phase in which new features were implemented in idbb, complementing features that were earlier implemented in idbb component (thus, test data represented an stabilization period for such component, presenting lower growth). Finally, the architect could not explain the large variability in the idec component in recent, test data.

Thus, analysis results provide indications that the proposed method may be useful to address project size in the future. Moreover, analysis indicates that it can elicit relevant questions about project component’s evolution, useful to register project history and to predict future behavior for the software project.

5 Related Work

The most typical approaches for using version control data to provide insight upon a software project are based on visualization. Many approaches, such as the Seesoft tool [13], the Aspect Browser [14], and Augur [15], allow the visualization of dependencies among changes to a software project. Such tools usually summarize project information in line lengths and/or color codes, allowing thousands of data elements to be presented simultaneously on a screen. However, these tools are not yet able to draw conclusions from the data, leaving this task to the analyst.

Monte Carlo simulation has been used to draw the evolutionary path of specific uncertainties in software projects. Grey [16] and Hullet [17] present simulation methods to estimate project schedule and cost baseline. In [2], we see the use of Monte Carlo sampling to model uncertain results of a system dynamics model, according to probability distributions associated to its parameters.

To the best knowledge of the author, there is no attempt to use size information from version control systems to predict the future size of software projects. However, in a recent paper Kitchenham et al [18] addressed the benefits of using in-company estimation methods (that is, estimation models based on information from previous projects developed by the same company) instead of using cross-company methods. It was observed that, in some companies, in-company methods performed better than cross-company models, thus enforcing that in-project data may be useful for estimation.

6 Limitations of the Proposed Approach

Due to the nature of the information used to describe the parameters of the simulation process, the proposed
approach has some limitations. First, it is tightly related to the coding activity, since version control systems are usually used during such activity. However, we see growing interest on using such systems before coding, to store and evolve specifications, design diagrams, and other artifacts created during project development. As research in this direction unfolds, we probably will be able to use other artifacts to support estimation.

Second, the approach cannot estimate project size early in the life-cycle, since there must be “past information” upon the project. We suggest that analogy estimation, based on similar projects from the past, might be used to provide a first estimation for the project. This estimation can be refined as project information is made available in version control systems. The combination of analogy and project-based estimation strategies requires investigation and is a future perspective for our research.

We also believe that the usefulness of the proposed approach may vary according to the usage of different project life-cycle models. Project-based estimation is probably more useful when development is carried on using an incremental or spiral life-cycle model, in contrast to a waterfall life-cycle model, since the coding activity (and generation of version control information) starts earlier in the former models.

Other important aspect of the proposed method is that estimation is solely based on the production of source code lines. Though it may seem that many factors that influence the growth of a software project are overlooked, the method assumes that recent history will repeat itself in the near future. Thus, the same factors that affected recent production of source code lines will play their role again, affecting production in the estimation time horizon. So, we rely on a single relevant productivity indicator, assuming that it captures the influence of other factors.

7. Conclusions and Future Perspectives

This paper presented a simulation process to estimate the size of a software project in a given time horizon, according to information collected from the project’s version control system. The project is divided into components, component size changes in the past are calculated from version control data, probability distributions are drawn to describe component growth dynamics, and Monte Carlo simulation is used to estimate the joint effect of these distributions upon project size.

Further investigation is being conducted to assess the usefulness of the approach under distinct development models. We intend to evaluate it under agile development, process-oriented organizations, and distinct life-cycle models. We also intend to provide better procedures for the component division and outlier elimination stages of the proposed approach. Moreover, since Monte Carlo simulation is computer intensive, we can study version control information on the perspective of time series models, such as auto-regressive (AR), moving average (MA) and integrated models (ARIMA).

**Acknowledgments.** The author would like to thank and acknowledge the support from the Brazilian Research Council (CNPq) and the Foundation for Scientific Development of the Rio de Janeiro State (FAPERJ).

**References**