SEEC – A Software Error Estimation Method for Multi-component Software Projects

Timo KUUSELA, Tuomas MÄKILÄ
Department of Information Technology
University of Turku, Finland

Abstract. In this article, some existing error estimation methods are presented and their suitability for modern software development projects is analyzed. Based on this analysis, a new software error estimation method SEEC (Software Error Estimation by Component-wise comparison), specifically designed for multi-site, multi-component software projects, is introduced. The required activities and the mathematical formulation of the new method are presented. Finally, the integration of the SEEC method to a software development process is illustrated with the SPEM process modeling language.

Keywords. Software development, software project planning, software error estimation

Introduction

More and more often software development projects can last years, cost loads of money and require continuous effort of hundreds of people. The projects can be divided in several sites in terms of software process activities and responsibilities or in several parts in terms of software functionality itself. Messing around with a bunch of code by a fistful of talented programmers has turned to large and complex projects that hold enormous resources and bury various different risks.

These large software projects require increasingly planning and management related activities: resource and schedule planning, budgeting, risk management, and software reliability and quality management to mention a few. To be reliable, all the related decision-making has to be based on established practices and techniques. One part of these techniques includes different estimations that are produced to assist the important planning activities and that can be based on, for instance, history data, expert opinions or future indicators.

One crucial, but perhaps a bit underrated, category of estimation is software error estimation. That is, estimating the number and the cumulative rate of software errors in the developed software during the project. The later an error is found in the project, the

more cost it brings to be corrected. It is clear that estimating software defect amounts has everything to do with the required resources in testing, error management and implementation. A reliable total error amount estimate helps the project to be prepared to the upcoming errors and to avoid unexpected expenses. In addition, comparing the prevailing error amount of the software to the original estimate also gives direct indications of the current state of the software's reliability during the project.

Error estimation can be seen as an important activity in a mature software process. Producing a reliable estimate requires some time and resources (depending on the applied estimation method) but it can be an effective tool to assist the planning and analysis of the project. This article presents the basic categories of software error estimation and introduces some known techniques and models. Having stated that none of them live up to the demands of a modern software development project, a new estimation method is introduced. A view on how the new method can be applied in a real project is also presented and modeled with the *SPEM* (*Software Process Engineering Meta-model*) language [1].

1. Different Classifications of Error Estimation

Some error estimation methods exist in the literature, varying from simple mathematical techniques to more complex, closely project time-line related models. Some of them are meant to be used in the early stages of the software development project producing one *static* total amount estimate that is used throughout the whole project. Some have a more *dynamic* nature. They are updated and fine-tuned during the project to give a more up-to-date picture of the current state of the project [2].

Both categories of error estimation have their purposes. An estimate made in the beginning of the project can be used by the project management to guide the resource planning and scheduling. It can also be used in software maturity estimations later on in the project when the decisions about software releasing points are made. A dynamic estimate being updated during project's timeline, on the other hand, gives a different kind of view on the state of the project by predicting the project based on the current situation [3].

An extensive error estimate – either static or dynamic – can be further divided into two more or less independent parts: total error amount estimate and error rate estimate (see Table 1). The former holds the estimate of the total error amount of the final software. These errors originate in different phases of the development project and are usually found in the different testing phases. The latter tries to describe at which stage of the project the errors are found. This behavior is usually described with a cumulative error curve [3].

Table 1: Error estimation categories.

Different categories of error estimation	Static total amount estimation	Dynamic total amount estimation
	Static error rate estimation	Dynamic error rate estimation

Many modern software development processes use iterative way of working. Therefore also testing – that is, discovering errors – is something that is being done in the development project a) during a long period of time and b) in many levels of abstraction (as also the traditional V-model of testing suggests). Hence, both error rate estimation and dynamic error estimation have an important role in the field of error estimation. This article focuses, however, mainly to the *static total amount estimation*. The main goal is to find the right way to estimate the total amount of software errors in a big, multi-site project developing large, feature-rich software.

2. Existing Error Estimation Methods

The word *technique* is used here to describe the somewhat straightforward ways to estimate the total error amount of software. A technique is considered to be a statistical or mathematical calculation that merely utilizes numerical data describing the software. A *model*, on the other hand, is considered to a more complex estimation method including more expert judgment and software project related figures like estimated time-line, software size or even more precise project parameters.

2.1. Error Estimation Techniques

Past error density is one of the oldest and simplest ways to estimate the total error amount [4]. It relies on comparing the size of the software of past projects to the one of the upcoming project. Having the error data of past projects available and relying to the fact that the ratio of *errors to software size* is the same in past and future software, error estimate for the future software can be made based on an analogy.

Error seeding is also a widely documented technique that is based in traditional statistics [4]. As the name suggests, some errors are produced on purpose by an extra group of programmers while another group is trying to find them by testing the software. Knowing 1) the number of seeded errors, 2) the number of discovered seeded errors and 3) the total number of discovered errors, estimate of the total error amount can be made. It relies on the assumption that a small sample of errors – that is, the seeded errors – reflects the state (in terms of total error amount) of the whole software. It should be noticed that the error seeding technique can only be utilized after the software has already undergone some testing activities. The formulation of the technique is as follows:

$$N_{errors,total} = (N_{errors,seeded,total} / N_{errors,seeded,found}) * N_{errors,found} \quad (1)$$

In the *error pooling* technique, the testers are divided into two independent groups, both reporting errors to their own *pools* [4]. The testing should be profound and cover the whole functionality of the software. At any given point of time the amount of unique errors in both pools, and most importantly, the amount of common errors found from both of the pools, can be calculated. According to the approach of the error seeding technique, the total amount of software errors can now be estimated using the following formulation (not mathematically derived, but estimated itself):

$$N_{errors,total} = (N_{errors,A} * N_{errors,B}) / N_{errors,A\&B} \quad (2)$$

2.2. Error Estimation Models

The field of software error estimation models highlights somewhat different aspects of estimation. A wider range of data describing the current development project is utilized. In addition, the methods become more complex, making good use of things like expert judgment in addition to mere mathematical calculations.

One large category of error estimation models is formed by the different models based on the *Weibull distribution*, a time-dependent mathematical formulation [2]. It includes parameters of shape, scale and time and describes the dynamics of the cumulative error rate in different situations. In the environment of software development, the two most widely applicable special cases of the distribution are the *Rayleigh model* and the *exponential model* [2], [5]. It should be noticed, though, that the total error amount is only a parameter in these formulations, not the output. They cannot be applied to the total amount estimation as such and are therefore left without further studies in this context.

Probably the most well-known model for software quality and error amount estimation is *COQUALMO* (*Constructive Quality Model*), published by Barry Boehm in 1997 and based on the widely acknowledged COCOMO (Constructive Cost Model) [6]. It has the somewhat same structure than COCOMO II but has its focus in software quality and error amounts instead of cost. At the time being published, it had already gone through some calibration with error data from certain real software projects.

COQUALMO is actually a combination of two sub-models: the Defect Introduction model and the Defect Removal model. The former estimates the total amount of software errors discovered from the software during the project, the latter estimates the amount of errors being removed from the software. In the context of this article, the former sub-model is the more interesting one.

The Defect Introduction model divides the incoming errors in three sources: requirements, design and coding. The input parameters of the mathematical formulation of the Defect Introduction model include a) software size, b) an error source specific calibration constant and c) an economy scale factor - all separately for the three error sources – as well as d) the so called defect introduction drivers (DID's) that are the main factors used to fine-tune the estimate and are partly inherited from the originating COCOMO II. This results in the following formulation:

$$N_{errors,total} = \sum_{j=1}^{3} A_j * (Size_j)^{B_j} * \prod_{i=1}^{21} DID_{ij} \quad (3)$$

where j relates to the three sources of errors, A_j is the calibration constant related to the j:th error source, $Size_j$ is the size of the j:th error source, B_j is the economy scale factor of the j:th error source and DID_{ij} is the i:th DID of the j:th error source.

The starting point of the model is size that has to be estimated separately for the three different error sources. It can be expressed in lines of code, in function points or

in some other applicable way depending on the nature of all the error sources being estimated. The error source specific calibration constant and the economy scale factor are parameters that can be utilized to adjust the estimation based on available project data outside the scope of the DID's.

The DID's form the central control point of the model. They are divided in four main categories: platform, product, personnel and project. In these categories the DID's are further divided to altogether 21 separate drivers (like required reusability, platform volatility, programmer capability and process maturity). As the formulation shows, these drivers acting as multiplicative constants are the main medium used to fine-tune the estimate. They hold the organization and project specific information that is utilized to get the final estimate. The initial valuation of the DID's – the most essential thing having an effect on the estimation result – is based on expert judgment. As projects go by, the drivers must be calibrated with the available data.

2.3. Notable Shortcomings of the Existing Methods

Let us revise the modern software development environment to which an error estimation method should fit. First of all, the software can be remarkably large and consist of various independent components. It may include a lot of new features and technologies compared to preceding releases and/or to software of other companies. Secondly, the software can be developed by multiple suppliers working in various separate sites. The different suppliers may have their own processes and ways of working. Thirdly, organizations developing software this large usually manage several successive projects with the software somewhat related to each other. It would not be realistic to assume that an organization would only have been put up for one project. This factor of continuity is also something that has an effect on the ways to handle error estimation. So, what are the biggest deficiencies of the existing methods? Is one of them the solution for the whole problem?

Relying purely on the analogy between past and upcoming projects – as in the past error density technique – can be quite dangerous. As new features and technologies are introduced in the software, it is impossible to say whether it can directly be compared to the preceding released software of the organization in question or not. In addition, using only the size of the software to illustrate its complexity can lead to incorrect estimates.

The biggest problem with the error seeding technique is the demand of extra resources. While the normal implementation and testing activities are ongoing, other groups are needed to produce and to look after the seeded errors. It is highly debatable whether a software development organization would invest to error estimation by hiring several new professionals or not. The answer is most likely: not. In addition to the first problem, error seeding also contains the risk of seeded, even critical, errors to remain undiscovered in the software. These can cause unwanted functionality or side-effects later on in the software.

The error pooling technique has some of the same downsides than the error seeding. Firstly, two independent testing teams are required. Both of them are required to have the expertise to test the whole functionality of the software. Secondly, a lot of extra work (in terms of creating several overlapping error reports) is done for the sake of estimation. Thirdly, one more expert is needed to do the comparison between the

two error pools. Hence, the technique seems too resource-greedy to be actually utilized in real projects.

Different life-time models, like the models based on the Weibull distribution, are very useful when describing the cumulative error amounts during the development project, described by the error curve. They cannot, however, be used in the total amount estimation because they only utilize such information, do not produce it. Combined with a reliable way to estimate the total amount estimate they can, however, form a solid practice to end-to-end software error estimation.

The Defect Introduction model of the COQUALMO is the most realistic and extensive effort to match the demands of proper error estimation method. It perceives the software development project as a complex system from which several independent areas of making can be identified and separately valuated. It has many clear weak points, though. First of all, the model focuses on calibration which means that it requires data from several upcoming consecutive projects to become reliable. It is not guaranteed that the model still fits the environment of the applying organization after the calibration time has passed and it is finally ready to be utilized. Secondly, the key input parameters like the DID's are highly supplier specific and are unwieldy adapted to describe the whole organization of a multi-supplier project. The multi-site factor also makes it hard to form a common practice to evaluate the size of the software which is the starting point of the whole estimation.

3. SEEC – A New Approach to Software Error Estimation

As the existing techniques and models do not seem to offer a complete solution, the principles of a whole new estimation method have to be settled. A new method may make good use of some of the upsides and inventions of the presented methods but it introduces a new way to divide and conquer the difficult field of error estimation. In the following, the most important aspects of software error estimation are discussed and the important decision-making related to the new estimation method is presented.

First of all, the *unit of estimation* should be established. The two alternatives are a) to deal with the whole software as a one large unit or b) to divide the software to well-defined components. The software development environment to which the new estimation method is targeted highly demands the latter approach to be used. The software is most likely developed in independent components holding specific features and technologies and it can be developed by multiple suppliers, internal or external to the managerial organization.

At the same time, one should keep in mind that the components and their suppliers are to be closely examined and valuated when the final estimate is fine-tuned. After all, many different things may have an effect on the error amount of a software component. So, dividing the software to too many components can lead to a dead end when the final adjustment of the estimate is made. A proper level of granularity should therefore be the goal of the component division of the software.

Secondly, the *reference level* of the estimation should be decided. Some existing methods use the estimated software size as the starting point, some can be used only after the testing is started and some actual error amount data is available. But if one

wants the new method to be a reliable tool for the project planners, the reference level should be available in the beginning of the project and rely on existing data, not to be an estimate itself.

The answer to this problem is analogy: not to use it in the whole total amount estimation but to use it in the discovering of the reference error levels. In practice this means that the error amounts of the defined components in the past projects of the organization in question are chosen as the starting points for the estimation. In whatever manner a component is discovered and defined, using the same kind of tactics the error amount of that component in the preceding software is determined. If some past error data from several projects is available, it is only reasonable to make good use of as many of them as possible.

Third essential thing is to come up with a well-defined way to *fine-tune* the reference error amounts so that the final estimate would actually reflect the expected state of the upcoming software. The adjustment should not only be a collection of educated guesses but instead a solid practice that could be used and further developed through a flow of successive projects. Achieving this requires identification of the key factors having an effect on the error amount of a component. These may include concepts like changes in feature sets, changes in applied technologies, or changes in supplier functions. This article does not try to identify all the possible influencing factors but instead highlights the fact that the identification itself should be done and the identified drivers should somehow be anchored to enable the evolution of the estimation method. The intention is not to freeze the whole set of influencing factors but to have a certain basic set as a starting point instead. This collection can and should be extended in the context of a new evaluated project.

In the mathematical formulation of the new method, see Eq. (7), these so called *change factors* take the form of multiplicative constants used to operate the reference values component-wise. It is therefore clear that they have to be valuated. In the ground level it means giving some kind of value for every type and amount of change in the component or its supplier comparing to the preceding versions. After the set of identified change factors is valuated, correct values are selected based on analysis on preceding and upcoming products. Although the valuation and the selection of correct values are essential parts of the estimation activity itself and they have to be done before the method can be applied, they are also considered to be outside the scope of this article. Only an estimation framework is introduced leaving highly domain, project and organization specific details to further studies.

Dividing the software to components and comparing these components in the preceding products with the ones in the evaluated product captures the essence of the new estimation method. The initial goal was to discover a way to estimate software error amounts. Taking these facts into account, the new method is given the name SEEC - Software Error Estimation by Component-wise comparison.

Figure 1 illustrates how the SEEC method can be integrated as a part of a large software project. The two primary activities utilizing the method are project planning and project tracking. In the former activity, SEEC estimate can be applied for resource planning. In the latter activity, it can be used to predict the current level of software maturity and reliability, and, thus the progress of the project. The estimation activity

relies on past error and project history data. Therefore the process of the applying organization must include corresponding data gathering activities.

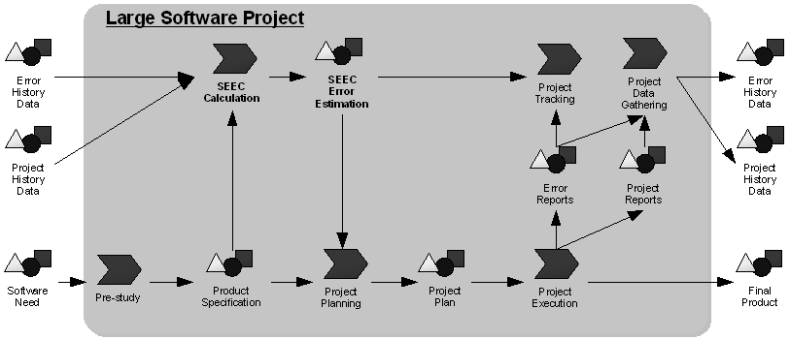

Figure 1. SEEC method as part of a large software project.

One more important aspect of estimation has to be pointed out. Software projects or processes are not machines that can be totally estimated and controlled in a watertight way. In addition, the pure numerical data does not always describe them in a comprehensive way. The recognition of these facts leads to the acknowledgement of the importance of the human factor, the expert judgment. It is needed in many points of the estimation, even with the new method. Both component division and the identification and valuation of the change factors require expert judgment. What makes the most essential difference between mere guessing and applying this method, though, is the logical and well-defined manner the method is constructed and instructed to be applied.

4. Required Activities and the Mathematical Formulation of SEEC

Now that the most essential aspects of software error estimation are discussed and related decisions concerning the new method made, the resulting approach has to be summarized as a re-usable formulation. The error estimation is done component-wise which indicates that the mathematical formula has to be a sigma expression. Error amounts of past products are used as the reference level so they form the core of the formulation. Change factors are used to fine-tune the estimate so they are present in the formula as corresponding multiplicative constants. The required working activities and the final formulation of the new method are presented in the following.

First, the component-division of the developed software is made. E.g. different documents like project plans (maybe draft versions at the time), feature lists or product specifications can be used to assist this activity. The main goal is to clearly define independent software components (probably delivered by different suppliers) in a proper level of generality.

Secondly, the software error amounts of the identified components in the preceding software products are determined. Assuming that the projects are well-

organized and documented, error data of past projects should be available. The outcomes of this activity are the reference error amounts:

$$REF_1 ... REF_n \tag{4}$$

where n is the number of identified components and REF_i refers to the reference error amount of the i:th component.

Thirdly, based on careful analysis of the organization, project and the upcoming product itself, the complete list of change factors is established. In case of the project not being the first one of the organization in question deploying this method, this is done by going through the basic set of change factors and by completing it if necessary. In any case, the outcomes of this activity are the identified change factors:

Before the change factors can be applied they have to be valuated. In practice this means that the values for different extents of change in the scope of that change factor have to be decided. Regarding to the change factors of the basic set, the valuations are only updated, if necessary. After this, correct values for every identified change factor related to every identified component are selected from the collection of valuations based on profound analysis on past and upcoming products. The outcomes of this activity are the change factor values for every identified component:

$$CFV_{11} ... CFV_{nm} \tag{5}$$

where n is the number of identified components, m is the number of identified change factors and CFV_{nm} refers to the value of m:th change factor related to n:th component.

Putting the reference error amounts and the change factor values together results in the final formulation of the new estimation method:

$$N_{errors,total} = \sum_{i=1}^{n} \left(\prod_{j=1}^{m} CFV_{ij} \right) * REF_i \tag{6}$$

where n is the number of identified components, m is the number of identified change factors, CFV_{ij} refers to the value of m:th change factor related to the n:th component and REF_i is refers to the reference error amount of the i:th component.

Figure 2 illustrates the required activities and related work products, roles and tools of the SEEC method. Valuating change factors is the most demanding activity when applying SEEC in real software projects. Thus, a new role responsible for the valuation is introduced.

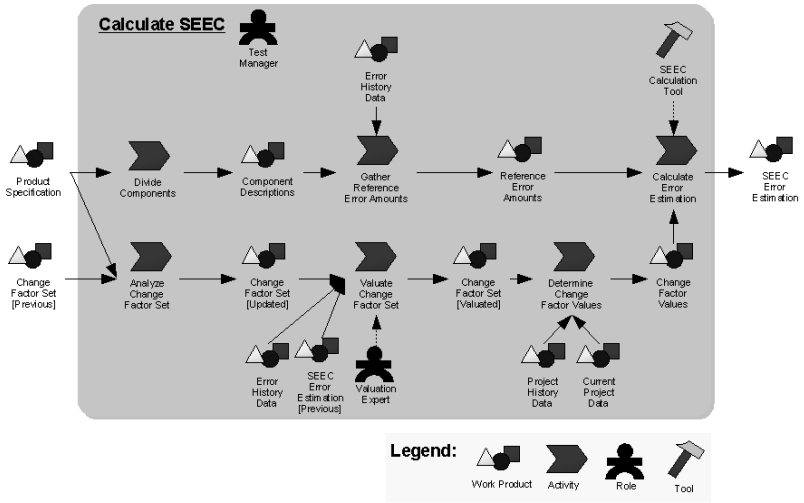

Figure 2. Internal activities, work products and roles of the SEEC method.

5. Conclusions

It was discovered that the existing error estimation methods are not fully sufficient for modern, multi-site software development projects including several different components. Therefore, a new way to estimate error in this kind of environment – the SEEC method – was presented. It serves as a framework offering the basic principles and activities to produce a reliable software error estimate. Before it can be applied in real software projects, it has to be customized to support the software process of the applying organization. Despite the fact that the overall structure of the method has been defined, individual activities have to be further examined in order to maximize the benefits of the method.

Also, illustration of software process metrics using the SPEM process modeling language was experimented in this article. It is evident that the graphical presentation helps to situate a solitary measurement method into the development process and to form the general view of the requirements and benefits of the method.

References

[1] Object Management Group. 2005. *Software Process Engineering Metamodel Spesification – version 1.1*. Object Management Group, January 2005. formal/05-01-06.

[2] Kan, S. 1995. *Metrics and Models in Software Quality Engineering*. Reading, Massachusetts: Addison Wesley.

[3] Kuusela, T. 2005. *Developing a Software Error Estimation Method for Series 60 Product Programs*. Master thesis, Turku University, March 2005.

[4] McConnell, S. 1997. *Software Project Survival Guide*. Redmond, Washington: Microsoft Press.

[5] Putnam, L. & Myers, W. 1992. *Measures of Excellence: Reliable Software on Time within Budget*. New Jersey: Prentice Hall.

[6] Boehm, B. & Chulani, S. 1999. *Modeling Software Defect Introduction and Removal: COQUALMO*. Technical report, USC - Center for Software Engineering, 1999.