Selecting a defect prediction model for maintenance resource planning and software insurance

Paul Luo Li
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh PA 15213
1-412-268-3043
Paul.Li@cs.cmu.edu

Mary Shaw
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh PA 15213
1-412-268-2589
Mary.Shaw@cs.cmu.edu

Jim Herbsleb
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh PA 15213
1-412-268-8933
jherbsleb@acm.org

ABSTRACT

Better post-release defect prediction models could lead to better maintenance resource allocation and potentially a software insurance system. We examine a special class of software systems and analyze the ability of currently-available defect prediction models to estimate user-reported defects for this class of software, widely-used and multi-release commercial software systems. We survey currently available models and analyze their applicability to an example system. We identify the ways in which current models fall short of addressing the needs for maintenance effort planning and software insurance.

General Terms

Management, measurement, economics, reliability, software maintenance.

Keywords

User-reported defect estimation, empirical maintenance models, software defect models.

1. INTRODUCTION

Hedging the risks associated with owning software is an important economic issue for both producers and consumers of software products. Risk aversion on the part of software consumers has created a market for software maintenance contracts and has opened the opportunity for software insurance. Software producers must manage the uncertainties of providing and marketing support that balance the risks and benefits they present to their customers. A good defect estimation model is an important first step toward pricing maintenance contracts and insurance and toward predicting support costs such as maintenance staffing. We compare models for estimating user-reported defects in the particular setting of widely-used, multi-release commercial software systems. For a defect prediction model to be useful in this setting the model has to account for aspects of the software system and its operating environment that could cause large variances in predictions.

Widely-used and multi-release commercial software systems include, for example, operating systems, servers, web-browsers, and office suites. These software systems are not intended to be mission critical, and it is generally recognized that they contain defects. However, they are used in situations where they are essential to the business interests of the user. The risk aversion characteristics of these users create a market for software maintenance support and software insurance.

Maintenance contracts provide assurance that a reported defect will be resolved within a contracted time and/or in a certain manner, while software insurance would compensate the user for the losses associated with a defect [8]. Although maintenance planners are interested in the costs associated with repairing a defect and insurers are interested in the damages that a defect can cause, both are interested in the number of defects that are likely to occur.

Currently, planning for maintenance activities is mostly ad-hoc, and business insurance does not separately consider software failures. Defect estimation models would improve the quality of information available for these decisions. Widely-used multi release commercial systems are usually developed and tracked with processes that gather historical information that can be used by a defect estimation model. A good defect prediction model should use this information to accurately model defect occurrences in the field and to account for aspects of the system that cause variation in defect rates between releases.

This paper differs from papers that evaluate the effectiveness of reliability models, e.g. [9] in that we focus on user-reported defects, which are defect occurrences in the field, for widely-used multi-release commercial software systems. Papers evaluating reliability models focused on defects found during testing. In addition, they concentrated on custom developed and one-off systems.

While we can only speculate on the value of a software insurance system, the motivation for better maintenance resource planning is clear. A market for software maintenance already exists. Allocating the right amount of resource can provide a competitive advantage for software producing organizations such as making better pricing decisions and lowering staffing costs, while maintaining the level of service.

Section 2 presents an example system and Section 3 describes the complexities associated with estimating defects for such a system. Section 4 surveys the currently-available defect estimation models. Section 5 analyzes how well the models address concerns

EDSER '03,

raised in Section 3. Section 6 make presents suggestions for further research and conclusions

2. EXAMPLE SYSTEM

We will use a concrete example to help us reason about the defect estimation problem. Consider a software system, SystemX.

SystemX is an operating system with the usual functionalities associated with such a system. Specification of these functionalities does not come from one customer, but rather has evolved through time by adding capability, in response to market demands. Customers of SystemX vary, from application development organizations that develop software for SystemX, to small businesses that depend on SystemX to run applications coordinating suppliers, and private users that use SystemX for leisure.

SystemX is developed by an organization whose business goal is to maximize profit. This organization has a clearly defined, repeatable development process. The organization's profit maximizing strategy is to keep customers happy and loyal by implementing advanced features and by providing customer service, all at minimum costs. In addition to developing SystemX this organization also provides customer service support. This customer service division answers and tracks user/customer questions that arrive through various channels. Incoming inquiries include a combination of questions that reflect defects in the software and questions that are requests for information or symptoms of user confusion. The support organization records a defect when it determines that a customer question is a code related problem. The recorded defects then become the responsibility of the maintenance organization, which could be the same as the development organization, to resolve.

We will use SystemX to compare and evaluate defect estimation models.

3. THE OPERATIONAL SETTING

SystemX is a widely-used multi-release commercial system. This means that successive releases of SystemX implement changes as dictated by the market. Each release will have many instances, installed with different configurations. A new release of SystemX repairs some defects and, as history indicates, introduces new defects. The decisions to adopt a new release and to report defects are made by the customer. This section examines the implications of these conditions. Each condition introduces variation in defect occurrence characteristics, which includes both the number of defects and the occurrence rates. Effects of these variations must be understood in order to predict defect rates accurately.

3.1 User-reported defects

This section addresses user-reported defects, the occurrence of interest. We follow the definition set forth in [6] and [7]: *a user-reported defect is a mistake at the coding level, which manifests in a deviation from the expected behavior that is reported by a user*. We take defects to mean user-reported defects in this paper.

After SystemX is released, the meaningful measure of perceived quality in the commercial setting is defects as reported by the user [6]. The number of users using the system directly causes variations in user-reported defect occurrence characteristics.

We count the number of defects associated with actual code defects. Many users might report the same defect or different symptoms of the same defect. For our purposes, a defect is recorded the first time it is reported, and subsequent reports of the same defect are ignored or attached to the prior defect report. In addition, not all calls to a call center reflect defects; some, for example, reflect misunderstanding by the user. Defects are recorded only when the support organization can associate them with code defects.

3.2 Widely-used systems

This section addresses characteristics of a system that is used in different configurations and for different purposes. Widely-used systems have many instances in operation running with different hardware and software configurations, as is often the case, for example when a system is sold to many different customers.

Differences in configuration for SystemX include other systems, such as databases, middleware, and applications, hardware components, such as processors and storage devices, and/or in purpose of use, such as for business critical infrastructure or for leisure.

Our description of "widely-used" is not to be confused with "widely-distributed", which refers to systems that are implemented with many distributed communicating components. It is also not to be confused with a system that has simply many users. Although System X could have many users, we are concerned here with systems that have multiple instances.

Widely-used characteristic causes variation between defect occurrences characteristics during testing and in the field. The number of different possible configurations of System X makes it infeasible to consider and test all possible configurations during development. Moreover, the large number of different types of uses makes it impractical to discover all usage scenarios.

3.3 Multi-release systems

This section discusses the characteristics of an evolving software system that changes to meet developing trends by incorporating the latest innovations. We characterize multi-release systems as having successive releases that incorporate incremental changes and improvements.

Depending on the features introduced in a release and a customer’s usage characteristics, some customers might choose to adopt the latest release of SystemX right away, other might delay adoption, and others might choose to stay with older releases. The development process is also undergoing constant improvement.

Different user adoption strategies cause variation in defect occurrence rates. A development organization might always adopt the latest release of SystemX to assure continued compatibility with its own products. A small business might delay adoption. Leisurely users might never adopt the latest release of SystemX unless they wish to utilize an important feature. Defects associated with particular hardware and software configurations might or might not be discovered for a given release.

Differences in the complexity and extent of the changes introduced by a release can cause variations in defect occurrence characteristics. Releases that introduce major modifications or new technologies, which due to their critical nature and/or complexity might have significant impacts on the development process, the resulting software system, and user adoption and usage characteristics.

Multi-release systems could also experience variations in defect occurrence characteristics resulting from changes in the development process. Development organizations are constantly trying to improve their processes. The improvements are incremental, but these process changes can affect the number of residual defects in software system.

3.4 Commercial systems

This section presents characteristics of a software system developed by an organization in which the release schedule is driven by market forces.

SystemX's quality is one among many forces guiding development and release decisions. Other considerations include demands of the market, timetables set forth by management, and resource constraints of the organization. Business objectives might make it necessary to make trade offs between time to market and quality. Commercial systems will have variations in defect occurrence characteristics, since the software must sometimes be release with substantial number of defects still in the system.

3.5 Summary

The problem is multi-dimensional. Table 1 presents a summary of the concerns that the defect model must address. This serves as a reference when thinking about possible solutions

Table 1. Summary of concerns.

Characteristics	Sources of variation in user-reported defects
User-reported defects	-Number of users
Widely-used system	-Software configurations -Hardware configurations -Usage characteristics
Multi-release system	-Adoption characteristics -Changes introduced -Development process changes
Commercial system	-Number of residual defects at time of release

4. EXISTING MODELS

For simplicity, we focus only on the defect occurrence rates and treat all defects equally. We recognize that different defects will have different associated costs, such as their damage to the customer associated or cost of their repair to the development organization. We hope to address these concerns in later works. This section examines currently available prediction models focusing on aspects of the models that we will use for our analysis in section 5.

4.1 Parameterized mathematical models

Parameterized mathematical models have a fixed form with parameters that are tuned using data from the current release. Since this class of models was originally developed for customized one-off systems, they assumed that the intended usage environment could be simulated accurately, thus they were able to extend defect occurrence rates found during testing into the field [10]. These models fit their parameters using defect counts and defect occurrence times starting from time zero up to time t. From the tuned model, the defect occurrence rates and in some cases, the numbers of remaining defects after time t are estimated.

β and α are model parameters, which have different meaning and values for each model. The total number of defects, N, is assumed to be given in some models, like the Weibull [6] and variants of the exponential [11]. Since N cannot always be accurately estimated, other models include the total number of defects as a model parameter to be estimated. The model parameters are tuned using data from the current release up to some time t, through mathematical methods like maximum likelihood or least squares. Due to the different model forms, each model will produce a different prediction for the number of defect and the rate of defect occurrences after time t. A summary of commonly applied models is in Table 2.

Table 2. Summary of parameterized mathematical models.

Model Group	Model Form	Input	Output
Finite Exponential	$\mu(t)= \alpha (1–\exp(-\beta t))$	Defect occurrence counts and times, up to time t	Defect rates and defect counts after time t
Infinite Geometric	$\mu(t)=(1/\beta)\ln((\alpha\beta\exp(\beta))t + 1)$	Defect occurrence counts and times, up to time t	Defects rates and defect counts after time t
Infinite Logarithmic	$\mu(t)=\ln(\alpha\beta t+1)/\beta$	Defect occurrence counts and times, up to time t	Defect rates and defect counts after time t
Finite Weibull	$\mu(t)=N(1–\exp(-\beta t^{a}))$	Defect occurrence counts and times, up to time t	Defects rates and defect counts after time t
Finite Gamma	$\mu(t)=\alpha(1–(1+\beta t)\exp(-\beta t))$	Defect occurrence counts and times, up to time t	Defect rates and defect counts after time t

Table 3 gives a summary of implied assumption and research and applications. Due to the different model forms and parameterizations, each model in Table 2 has a different implied assumption regarding the hazard rate, the rate at which the defects remaining in the system are uncovered at time t. The hazard rate will be important when we analyze and compare the ability of parameterized models to predict defect for SystemX in Section 5.1. Detailed explanations of the concepts presented in this section can be found in [8].

Table 3. Summary of assumptions

Model Group	Hazard Rate	Defect Total	Research
Finite Exponential	Constant	Finite	Jelinski and Moranda, 1972 Musa, 1979
Infinite Geometric	Decreasing	Infinite	Moranda, 1979
Infinite Logarithmic	Decreasing	Infinite	Musa-Okumoto, 1984
Finite Weibull	Increasing	Finite	Kenny, 1993
Finite Gamma	Increasing	Finite	Yamada, Ohba, and Osaki 1983

4.2 Bayesian methods

In contrast to parameterized mathematical models, which require data from the current release up to time t to estimate defects after time t, Bayesian models can take prior information, such as information from previous releases, to estimate defects before the release of the current release. An important feature of the Bayesian framework is its ability to use information as it become available to improve and adjust estimates. The Bayesian model is the same as its non-Bayesian cousin. The difference is that the parameters in the Bayesian models are treated as variables as well.

Bayesian model need prior distributions for each parameter in the model [10]. There are various ways to construct the prior such as creating a distribution of possible parameter values using fitted model parameters of previous releases or eliciting possible values from experts, and in the worst case where there no prior information is available, it is possible to mathematically construct a non-informative prior that offers no insight into the system but which makes it possible to generate a prediction. Once data becomes available from the field, the model parameters can be adjusted from the prior to better fit the data. At any time t it is possible to have modified predictions as result of re-estimating model parameters using field defect information.

Table 4 gives a summary of the inputs, outputs, and research on Bayesian models. We analyze the ability of Bayesian models to model defects for SystemX in Section 5.2.

Table 4. Summary of Bayesian models.

Model Group	Research	Input	Output
Bayesian Exponential	Littlewood 1987	Historical defect occurrence information and available defect occurrence information	Defects occurrence rates and counts
Bayesian Gamma	Littlewood-Verrall 1980	Historical defect occurrence information and available defect occurrence information	Defect occurrence rates and counts
Bayesian Geometric	Liu 1987	Historical defect occurrence information and available defect occurrence information	Defect occurrence rates and counts

4.3 Product/process models

Process and product models estimate the number of defects in the current release using estimates of the size of the product or deviation from the previous release and known organizational information.

Research in this area traces back to Belady and Lehman's work on system growth and system structure degradation in successive system releases [7]. Their research serves as the basis for several methods, which measure the amount of change in a system and predict the number of defects that result from the changes. Some notable results are summarized in Table 5. We analyze the ability of product/process models to predict defects for SystemX in Section 5.3.

Table 5. Product/process models.

Model Group	Research	Input	Output
Using lines of code with organizational adjustments	Rome Laboratory, 1992 COQUALMO, 1999	Lines of code estimate and process effectiveness estimates	Total number of residual defects
Using software change information	Mockus et al. 2003, Graves et al. 2000	Change management information	Number of residual defects (effort)

5. ANALYSIS OF EXISTING SOLUTIONS

Given the various models available we now examine how well they address the concerns involved with modeling user-reported defect for SystemX.

5.1 Ability of the model to fit the data

The first step is to choose a model that accurately describes defect occurrence data. This will also involve picking a model from Table 3 that fits the conditions of SystemX.

Section 4.1's survey of parameterized mathematical models in practice shows that three categories of models are widely applied: finite exponential, infinite, and finite Weibull and Gamma.

Finite exponential models like those that used by Jelinski and Moranda [11]and Musa [14], (First row in Table 3.) assumes that the number of defects in the system is finite and that the hazard rate, the rate at which defects are uncovered at time t, is constant. These assumptions do not fit SystemX since the repair process can introduce defects, and the rate at which the varying numbers of customers are uncovering defects is non-constant

Infinite models like those used by Musa-Okumoto [15] and Moranda [12], (The third and forth rows in Table 3.) assumes that the number of defects in the system is infinite due to a decreasing hazard rate. Although this class of models correctly assumes that defects are being re-introduced into the system. It also assumes that the rate at which defects are being uncovered is always decreasing, but since the number of users of SystemX increase over time, the rate at which defects in the system is uncovered should be increasing.

The final class of parameterized mathematical models we examine is Weibull and Gamma models like those used by Kenny [6], Yamada and Ohba [17]. (Row five and six in Table 3) Like exponential models, the number of defects is assumed finite. However, unlike both exponential and infinite models, this class accurately describes the migration characteristic. This class of models can describe the ramping up process due to migration seen in commercial systems, using the expressive power offered by the models' parameters.

The Weibull model has been shown to be able to describe the defect occurrence patterns seen in widely–used commercial systems[6]. We not aware of any other papers that attempts to fit other model to defect occurrence data from a system with our characteristics.

5.2 Ability to incorporate prior information

The main drawback of parameterized mathematical models is the need to wait until defect data from the current release is available. However, if we want to plan for maintenance, we need to make defect predictions before release to the field. We cannot use defects occurrences during testing to tune the model before release, because due to the widely-used condition of SystemX defect rates seen during testing will not reflect occurrence characteristics seen in the field. However, we can use the multi-release characteristic of SystemX to help us to over come this difficulty.

Defect information from previous releases can be used by Bayesian models in Table 4 to construct prior distribution for their model parameters before field defect information from the current release is available. The resulting model will be able to estimate defect occurrence rates and defect occurrence counts for the current release before release to the field. The Bayesian process also uses defect information as it becomes available to update the parameter estimates. At any stage during the Bayesian process, the data collected up to that point can be assumed to be prior information and all the parameters can be re-estimated. Intuitively this means that whenever defects are reported and repaired for SystemX, a Bayesian model can re-computed the number of defects in the system to reflect the fact that repairs could introduce new defect into the system.

The Bayesian Gamma model used by Littlewood should be able to describe the characteristics of a commercial system, but the total number of defects for the current release is a model parameter that is estimated using defect occurrence data from previous release. Intuitively this does not make sense because the number of defects seen in any given release of SystemX is a result of the amount change in the current release and the status of development organization, not a result of the field defect occurrence characteristics of previous releases.

There is not at present a Bayesian Weibull model. However, we feel that the machinery available through Bayesian methods and the ability of the Bayesian framework to update estimates be combined with the Weibull model.

5.3 Ability to account for product/production differences

Up to this point, we have not considered variations due to the product or the organization that developed the product. Since information about the changes a release implements and the development effort is available before release, product/process models use this information to estimate defect occurences.

COQUALMO, the extension into the software quality domain of Bohem's work with COCOMO estimates the number of residual defects in a system using lines of code and process drivers related to the characteristics of the particular organization concerning various defect introduction stages and defect removal methods. These drivers are derived from expert estimates at first, then tuned to match actual defect counts.[3] COQUALMO assumes that the number of defects at release time is the result of two processes during development, defect introduction and defect removal. The drawbacks of COQUALMO are that it estimates only the total number of defects and lines of code. Lines of code alone are not very good estimators of complexity. In addition, maintenance planning and software insurance for SystemX require defect occurrence estimations at any time t, not merely the total number of defect.

A promising alternative is to use change data as captured by change management systems. In addition to lines of change data, this data includes number of changes, feature requests, and developer who made the change. Mockus uses the non-intrusively recorded feature/fix requests, which can be seen as a change at a high level, along with lines of code and number of changes to estimate efforts [13]. The method uses a delay factor to determine when the estimated effort will occur and a ratio to describe the defect repair effort relative to development effort. However, this method does not adoption characteristics, which could vary between releases depending on the features implemented. In addition, this method does not take in to consideration process changes.

Currently, there does not exist a model that incorporates research in parameterized mathematical models, Bayesian models, and process and product models.

6. CONCLUSION AND THE ROAD AHEAD

Widely used and multi-release commercial software systems are an important class of software systems. Reliance on these systems has created a need for maintenance contracts to assure defect resolution and for software insurance to provide compensation in the event of a defect. Both techniques require a defect occurrence model, but no model is currently available that can account for all possible causes of variation. We find existing solutions lacking in their considerations of organizational changes and user adoption characteristics.

Current modeling techniques do not adequately consider organizational changes. Process changes and tool changes can affect defect occurrence characteristics as well as influential personnel changes, such as a lead designer [4]. We think evaluations of the development process from developers who have worked on a release and can better evaluate the process and product variations as shown by postmortems and experience reports. However, currently no model attempts to quickly capture and use this information.

Current approaches to defect modeling do not account for differences in customer adoption characteristics. We feel these differences are important because in the commercial setting the customers are the ones who exercise the code to detect defects. The number of customers and the configurations they use has a direct effect on the number of defects found. Customer support divisions and sales departments could provide the needed information.

Research in customer adoption is not to be confused with customer usage profiles and defect profiles as researched by Wyuker [16] and Bassin and Santhanam [1]. Customer usage profiles describe the frequency with which a piece of code is exercised. Defect profiles describe the type and order in which defects occur. While both are interesting, we are interested in rates of adoption: the number and type of customers who adopt a release and the time when they adopt.

Improved defect estimation model in our setting needs to be validated by showing that it is generally applicable to a wide range of commercial software systems like operating system, servers, office suites, or web browsers. In addition, it has be an improvement over existing solutions such as ones listed in Table 6.

Table 6. Summary of existing solutions

Model Grouping	Best solution	Desirable features	Main Drawbacks
Parameterized mathematical models	Weibull model	-Describes the ramping up pattern	-Assumes no defects are introduced into the system -Descriptive, not predictive thus cannot be used for predict before release.
Bayesian models	Bayesian Gamma model	-Predictive by using priors -Parameters (like defect totals) can change after release	-Does not consider release differences
Process/ Product models	Product change model	-Describes release differences though change data	-Does not consider organizational changes -Does not consider adoption characteristics

Once a defect occurrence estimation model is available, can we move towards structured maintenance planning and begin to evaluate the possibility of software insurance.

7. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Grand CCR-0086003, by the Carnegie Mellon Sloan Software Center, by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298.

The authors would like to thank Peter Santhanam and Bonnie Ray of IBM Research for their contribution to this work.

The authors would like to thank Audris Mockus for his insights, and Vahe Poladian for his valuable input.

8. REFERENCES

[1] Kathryn Bassin, Peter Santhanam. “Use of Software Triggers to Evaluate Software Process Effectiveness and Capture Customer Usage Profile.” Symposium on Software Reliability Engineering, 1997.

[2] Ram Chillarage, Shriram Biyani, Jeanette Rosenthal. “Measurement of Failure Rate in Widely Distributed Software.”

[3] Sunita Chulani. “COQUALMO9 COnstructive QUAlity MOdel) A Software Defect Density Prediction Model.” Project Control for Software Quality. Editors: Kusters, Cowderoy, Heemstra, and van Veenendaal. Shaker Publishing, 1999.

[4] B, Curtis. H. Krasner, et al. "A field study of the software design process for large systems." Communications of the ACM. 31(11): 1268-1287. 1988.

[5] A.L. Goel, K. Okumoto “Time-Dependent Error –Detection Rate Model for Software and Other Performance Measures.” IEEE Transaction on Reliability,

[6] Garrison Q. Kenny. “Estimating defects in commercial software during operational use.” IEEE 1993.

[7] M.M. Lehman, L.A. Belady. *Program Evolution: Process of Evolution Change*. Academic Press Inc. 1985.

[8] Paul Luo Li, Mary Shaw, Kevin Stolarick, and Kurt Wallnau. “The Potential for Synergy Between Certification and Insurance” Special edition of ACM SIGSOFT from the IWRE in conjunction (ICSR7), April 2002.

[9] Yahswant Malaiya, Nachimuthu Karunanithi, Pradeep Verma. “Predictability of Software-Reliability Models.” IEEE Transaction on Reliability Vol 41. No 4. 1992.

[10] Michael Lyu. *Software Reliability Engineering*. McGraw-Hill, 1996.

[11] P.L Moranda, Z. Jelinski. Final Report on Software Reliability Study. McDonnell Douglas Astronautics Company, MADC Report Number 62921. 1972.

[12] P.L. Moranda. “Event-Altered Rate Models for General Reliability Analysis.” IEEE Transaction on Reliability. Vol R-28,1979 pp376-381.

[13] Audris Mockus, David M. Weiss, Ping Zhang "Understanding and Predicting Effort in Software Projects." ICSE 2003 Proceedings

[14] J.D. Musa. “Validity of Execution-Time Theory of Software Reliability.” IEEE Transactions of Reliability, Vol R-28, 1979. pp181-191.

[15] J.D. Musa, K. Okumoto. “A Logarithmic Poisson Execution Time Model for Software Reliability Measurement.” Proceedings Seventh International Conference on Software Engineering, 1984.pp 230-238.

[16] Elaine Weyuker, S. Rapps. "Data Flow Analysis Techniques For Test Data Selection." Proceedings of the 6th International Conference on Software Engineering (ICSE), Tokyo, Japan, September 1982, pp.272-278.

[17] S. Yamanda, M. Ohba, S. Osaki. “S-Shaped Reliability Growth Modeling for Software Error Detection.” IEEE Transaction on Reliability, Vol R-32. 1983. pp 475-478.