Leveraging Defect Prediction Metrics in Software Program Management

Alagappan. V
Program Manager
Information Technology Industry

ABSTRACT

Software reliability assurance is a challenging task. The accuracy of predicting the defects in the software is based on many factors such as the estimated size of the software, estimated defect density and software complexity. Numerous software metrics and statistical models have been developed to address this but there are limitations of these models to different software development projects.

For successful completion of software projects, the program managers need relevant metrics to help in planning, monitoring and controlling functions. While Software program managers rely upon Plan Vs Actual metrics for project management attributes of Schedule, effort and cost, but often do not track these metrics with respect to quality(defects).

This paper discusses the experience of using defect prediction model across software development projects and how the defect metrics (Predicted Vs Actual) help program managers in planning resources and schedule, stakeholder expectation management, risk mitigation and project control.

General Terms

Management, Measurement, Reliability, Software Maintenance, Project Schedule Planning, Project Resource Planning, Program Management

Keywords

Keywords are your own designated keywords which can be Defect prediction, software development, project planning, Program management

1. INTRODUCTION

Lot of research and theories are available with respect to software reliability and software defect prediction models. Each model has inherent limitations and hence the effectiveness in deployment of these models/measurements in software development projects are varied and limited. This experience report summarizes the benefits of adopting a defect prediction model and discusses the considerations and limitations when such a model is deployed for a software development project.

2. SOFTWARE RELIABILITY

According to ANSI, Software Reliability is defined as: the probability of failure-free software operation for a specified period of time in a specified environment. Although Software Reliability is defined as a probabilistic function, and comes with the notion of time, we must note that, different from traditional Hardware Reliability, Software Reliability is not a direct function of time[5].

Software reliability is a key component in software quality. The study of software reliability can be categorized into three parts: modeling, measurement and improvement.

There are many defect prediction models that exist, but no single model that is universal to all the situations and can capture a necessary amount of the software and project specific characteristics. Assumptions and abstractions must be made to simplify the problem. Software modeling techniques can be divided into two subcategories: prediction modeling and estimation modeling[5].

Software reliability modeling has matured to the point that meaningful results can be obtained by applying relevant models to the problem. The challenge for the software development program manager is to choose the right model and in cases where the prediction is not accurate the defect prediction is not adopted.

Software reliability measurement is naive. Software reliability cannot be directly measured, so other related factors are measured to estimate software reliability and compare it among products.. The current practices of software reliability measurement can be divided into four categories: Product metrics, process metrics, project management metrics, and defect metrics.[5]

Software reliability improvement is hard. The difficulty of the problem stems from insufficient understanding of software reliability and in general, the characteristics of software. Until now there is no good way to conquer the complexity problem of software.[5] Complete testing of a moderately complex software module is proved impractical amidst cost and schedule constraints. Defect-free software product cannot be assured. Realistic constraints of time and budget severely limit the effort put into software reliability improvement and priorities are given by software practitioners for delivery of functionalities and timely project completion.

3. SOFTWARE DEFECT PREDICTION MODELS

A proliferation of software reliability models have emerged as people try to understand the characteristics of how and why software fails, and try to quantify software reliability. Over 200 models have been developed since the early 1970s, but how to quantify software reliability still remains largely unsolved. As many models as there are and many more emerging, none of the models can capture a satisfying amount of the complexity of software; constraints and assumptions have to be made for the quantifying process. Therefore, there is no single model that can be used in all situations. No model is complete or even representative. One model may work well for a set of certain software, but may be completely off track for other kinds of problems. [5]

Most software models contain the following parts: assumptions, factors, and a mathematical function that relates the reliability with the factors.[5]

Representative prediction models include Musa's Execution Time Model, Putnam's Model. and Rome Laboratory models TR-92-51 and TR-92-15, etc. Using prediction models, software reliability can be predicted early in the development phase and enhancements can be initiated to improve the reliability. [5]

Representative estimation models include exponential distribution models, Weibull distribution model, Thompson and Chelson's model, etc. Exponential models and Weibull distribution model are usually named as classical fault count/fault rate estimation models, while Thompson and Chelson's model belong to Bayesian fault rate estimation models.[5]

Rayleigh Defect prediction model, among the family of Weibull distribution, is found to be most suitable for predicting reliability of software product. It predicts the expected value of defect density at different stages of life cycle of the project, once parameters (total number of defects or total cumulative defect rate and peak of the curve in terms of unit of time) for the curve are decided. The number of defects recovered during various life cycle stages of a project conforms to a numerical distribution, which is represented by Rayleigh equation. Estimation of overall defect density of the entire project can be obtained by carrying out non-linear regression analysis using this equation with observed defect data for design reviews and code reviews. Estimation for number of defects for any stage (e.g., Unit Testing) thereafter can be obtained through Probability Distribution Function..[6]

The nature of curve indicates the pattern of defect removal rate in the life cycle of the project. The steeper it is, the less defect prone it is when delivered to customer, on the other hand, if it is flatter, it indicates inefficient defect removal rate and hence lot of defects leaked to customer.[6]

The field has matured to the point that software models can be applied in practical situations and give meaningful results and, second, that there is no one model that is best in all situations. Because of the complexity of software, any model has to have extra assumptions. Only limited factors can be put into consideration. Most software reliability models ignore the software development process and focus on the results -- the observed faults and/or failures [5]. By doing so, complexity is reduced and abstraction is achieved, however, the models tend to become specialized that can be applied to only specific situations and a certain class of the problems. We have to carefully choose the right model that suits our specific case. Furthermore, the modeling results cannot be blindly believed and applied.

4. EXPERIENCE OVERVIEW

Our experience of the defect prediction model deployment is for large software system development projects with duration of around 8 to 12 months and the software development methodology is carried out using waterfall model. The defect prediction model deployed is based on Rayleigh defect prediction model and the parameter used as an input to this model is the software size in function points and assumed defect density per function point.

Based on the experience in applying the Rayleigh defect prediction model across software development projects, we observed three different patterns and the insights from the defect metrics (Predicted Vs Actual) helped the program manager to better plan the project and manage stakeholder expectations effectively. Detailed below are how defects were predicted in each phase and the formula that was used for defect prediction

Est. Defect Density in Phase DDp =

$$= E * \left(e^{-B(P-1)^2} - e^{-BP^2}\right)$$

Where, E = total defects per Function Point

B = efficiency of discovery process

P = phase number

B' value is arrived based on the past data on completed projects

'E' value is decided based on the Technology, Size of the Project, Skill of Resource pool available, Initial Risks perceived and the past experience on similar projects

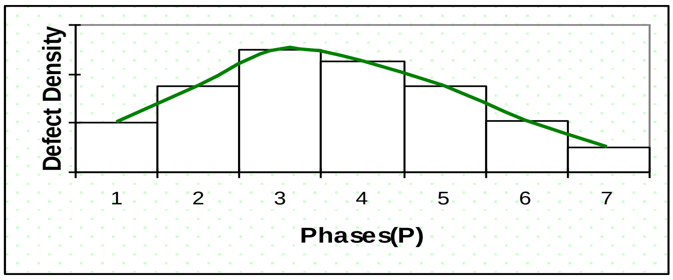

Figure 1 :Rayleigh Defect prediction

Pattern 1: Fluctuating defect trend

The reported defects fluctuate across different phases of the project and more defects are identified in the later phases of the project. This is more a typical trend we see in projects with schedule overrun, cost overrun and more change requests/defects to be fixed before the launch date. This is the case where there is lack of internal reviews, changing requirements and inadequate testing of the product before being delivered to end user .

Program manager could have leveraged the insights from defect metrics (predicted Vs actual) to increase the number of test iterations before delivering the product to customer.

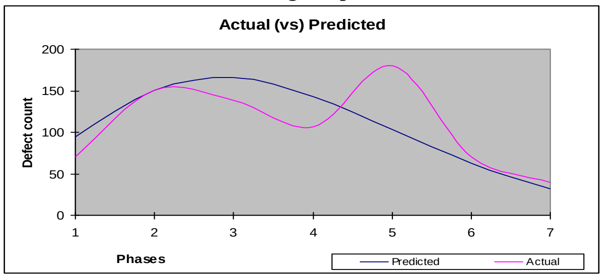

Figure 2 :Actual defects fluctuating over predicted defects

Pattern 2: Lower actual defect density

The reported defects are lesser than the predicted defects. Further analysis revealed that the defects are not recorded in each phase and also customer was involved in Early visibility testing (during the course of development, project team has demonstrated system to customer and received feedback). So for the study we adjusted the defects and the same is shown in figure 4. It is important to record all defects across the phases of the project.

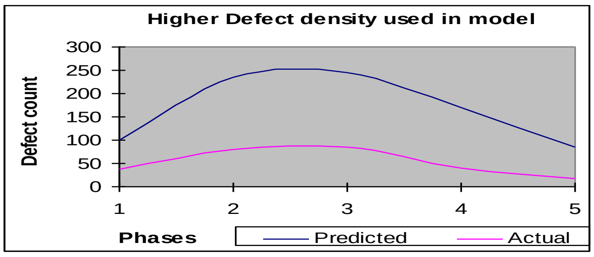

Figure 3: Actual defects lower than the predicted defects

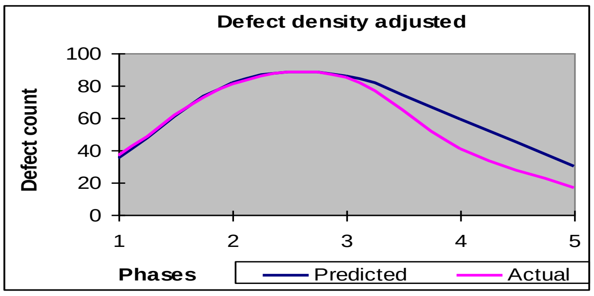

Figure 4: Defect density adjusted

Pattern 3: Effective Defect management

The reported defects are more than the predicted defects especially in the earlier phases of the project. These patterns usually result in less rework and better customer satisfaction during acceptance testing phase of the project and also less maintenance effort. Program managers leveraged the defect metrics for effective planning and customer communication. We could capture some of the best practices in planning and customer expectation management with the inputs of defect prediction model..

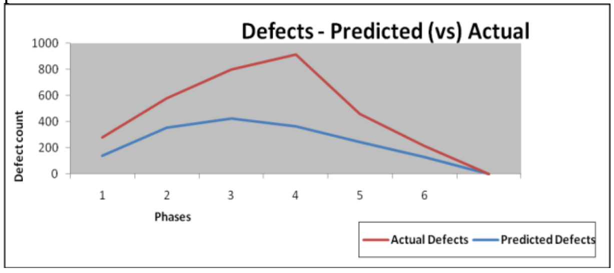

Figure 5: Actual defects more than the predicted defects

In most of the software development projects, the program management team will have less visibility of the quality of the product until the acceptance testing phase. The use of defect prediction model helped identifying defects at the earlier stages of SDLC (Software development Life Cycle), reduced the rework due to defects in later stages of SDLC and thereby reduced the COQ(Cost of Quality).

Author has seen the application of this very effective in projects with multiple iterations where the defect prediction model can be improved with past data from the same project

4.1 Benefits of leveraging defect prediction in Program management

As the defect prediction acts as a forecast, the model can be used as a planning tool in the following ways

a) Resource and schedule planning for different phases of the project

b) Number of iterations for the testing phase

c) Estimate the maintenance effort

4.2 Resource and schedule planning

The early estimate of defect helps in better utilization of resources by re-deployment of manpower to the appropriate modules and also the resource mix during various phases of the project. Typical allocation of resources for a module is based on the complexity and effort required for that. During testing phase the defect prediction should determine the number of testing resources and development resources required for each module.

Number of defects to be identified in each phase is one of the critical inputs to determine the schedule and revalidation of the schedule for different phases of the project.

4.3 Test Iterations

In most of the projects the number of test iterations is dependent on the duration of the testing phase, the number of testers and availability of the system builds. In most cases the testing duration is compromised due to the delivery deadlines and delay in development completion. With defect prediction model it helps to plan the appropriate number of test iterations. For example, in the first iteration we couldn't identify the estimated number of defects, hencewe need to plan for further iterations before delivery. We can also prioritize the iterations for the modules where lesser number of defects are identified.

4.4 Maintenance Effort

The thumb rule to estimate the maintenance effort is based on the project size or development effort. Software practitioners need to estimate the staffing requirements during support and warranty phases

If the defects data (predicted and actual defects for each phase) is available then we can estimate the residual defects in the system and thereby determine the maintenance effort.

5. LIMITATIONS

The number of defects alone cannot be sufficient information to provide the basis for planning quality assurance activities and assessing them during execution. That is, for project management to be improved, we need to predict other possible information about software quality such as usability, performance requirements, ease of maintenance and so on. Defect prediction models are based on the estimates and hence the accuracy of prediction depends on the accuracy of estimates (we mean the size of the project). Also to manage stakeholder expectations the program manager should convince them on the reliability of this model and that is key for him to provide rationale for the planning decisions (number of development and testing resources in each phase, number of testing iterations, duration of testing phase)

This is applicable only for software development projects that are of Water fall or modified water fall type. This is not applicable for package implementation or upgrade projects.

6. CONCLUSION

Generally, efforts have tended to concentrate on the following three problem perspectives as far as defect prediction is concerned [2].:

1. predicting the number of defects in the system
2. estimating the reliability of the system in terms of time to failure
3. understanding the impact of design and testing processes on defect counts and failure densities

The author has experienced the benefits of defect prediction as a useful program management tool that helps in project planning and better stakeholder expectation management. Program managers who intent to adopt this as a planning tool should necessarily ensure the reliability of this model for their project with the results in earlier phases of the project/iterations and also focus on software estimations and change requests for the project that impacts the defect prediction. Further work can be done to study the suitability of defect prediction models in other software development methodologies and the extent the defect metrics can be leveraged by software development program manager.

7. ACKNOWLEDGMENTS
My thanks to all the project stakeholders, team members, colleagues, customers who have provided the required background for me to assimilate the thoughts and ideas to summarize my experience in this report.

8. REFERENCES
[1] Smidts, C., Li, B., Li, M. and Li, Z. 2002. Software Reliability Models. Encyclopedia of Software Engineering.

[2] Norman Fenton,Martin Neil,Centre, “A Critique of Software Defect Prediction Models, for Software Reliability”, 0098-5589, IEEE Transactions on Software Engineering, Oct 1999

[3] Fenton, N.E. and Ohlsson, N. "Quantitative analysis of faults and failures in a complex software system" IEEE transactions on software Engineering, Vol 26, No 8, August 2000, 797-814

[4] W. Florac, "Software Quality Measurement: A Framework for Counting Problems and Defects," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-92-TR-022, 1992. http://www.sei.cmu.edu/library/abstracts/reports/92tr022.cfm

[5] Jiantao Pan , "Software Reliability" Carnegie Mellon University, http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/, 1999

[6] M.Thangarajan, Software Reliability Predicition Model Whitepaper, Tata Elxsi, http://www.tataelxsi.com/htmls/pds/pds_whitepaper.php

[7] Kehan Gao,Taghi M. Khoshgoftaar,Huanjing Wang,Naeem Seliya , “Choosing software metrics for defect prediction: an investigation on feature selection techniques”, Software—Practice & Experience, Volume 41 Issue 5, April 2011