

Software Reliability Prediction is not a Science...Yet

Peter B. Lakey

Cognitive Concepts, St. Louis, September 2002

www.cognitiveconcepts.org, plakey01@earthlink.net

I. INTRODUCTION

The purpose of this paper is to address some basic issues associated with software reliability prediction.

What exactly is software reliability prediction, and why do academia and industry bother with it? Below is an adequate definition of software reliability prediction:

A forecast of how reliable an executable software system will be at some point in the future based on metrics (data) available now.

This is based on the industry definition of software reliability, "probability of failure-free execution of a software system for a specified time in a specified operating environment." Note that for purposes of this paper we're talking about early prediction.

Here we distinguish prediction from estimation according to accepted software reliability engineering guidelines. Estimation is an assessment of how reliable a software system is now based on observed test data. Prediction is usually limited to a project period prior to system test. In other words, for prediction a development organization takes information about the system under development and uses some statistical regression model to forecast the level of reliability that will be present at some point in testing. Several prediction models have been documented in a number of sources [1,2].

The question posed here is this: "Do these prediction models serve any useful purpose? Are organizations successful in using software reliability prediction to manage their software projects?"

The thesis of this paper is that software reliability prediction is, or at least should be, a specialized form of software quality management, as defined by the SEI Capability Maturity Model (CMM) [3]. In fact, software reliability prediction is a misnomer. Organizations don't generally predict how reliable a software system will be during system test or upon delivery. Rather, a typical project has an inherent quality (reliability) goal that must be achieved. The software will not be released to the customer or end user for operation, in most cases, until the reliability threshold has been achieved.

Therefore, in most applications, software reliability is a constraint on the project, or a technical performance parameter. The implicit objective then is to achieve the software reliability goals for the system while minimizing cost and/or schedule for the project. Prediction in this sense is useful to support software project management.

II. BACKGROUND

It is the author's experience that performing software reliability prediction is not a particularly worthwhile endeavor. On a recent project the customer imposed a system reliability requirement for the objective (deliverable) system. The project's specialty engineering group decided to allocate the requirement among hardware and software elements, which is a reasonable thing to do. Then the program decided it wanted to use one of the better known models to predict how reliable the system would be upon completion and delivery.

Many painstaking hours were expended by engineers from both the developer and customer side to attempt to work out the details of the parameters and values that would be used to generate the prediction. After several months of haggling, the parties came to a consensus on what the "official" prediction would be. This was done several years before the system was to be fielded. The prediction sat on the shelf for over a year. No one ever used it. The software manager didn't even know it existed. The biggest problem with the prediction is that there was virtually zero confidence in the numbers. It was based on assumptions that had little supporting data.

III. LESSON

So, what's the point of the preceding story? Don't embark on a software reliability prediction unless you can answer the following two questions affirmatively:

1. Do you know exactly how the prediction is going to be used to steer the development and test activities and does the project manager share this view?
2. Do you have sufficient data and experience applying the method to ensure that predicted values have a high degree of confidence?

If the answer to both of these questions is Yes, good. The next recommendation is not to call this activity software reliability prediction. Instead it should be referred to in a CMM Level 4 compliant Software Quality Management Plan, or equivalent. After all, the purpose of such a prediction is to help manage delivered quality.

If the answer to one of these questions is "no" then very little time and effort should be expended on a prediction. Instead, as a software reliability practitioner, the effort could be put to better use by encouraging the organization to develop the capability to manage the project quantitatively with a software metrics program.

IV. RECOMMENDATION

When embarking on a software reliability program for an organization, it can be tempting to try to find simple answers to difficult questions. Typical questions may be "do you know of any good software reliability prediction models?" or "what sort of defect rates can we expect when we start testing?" The author started off his SRE career asking the same types of questions. Over time it became clear that there is no simple answer, no shortcut. If a project or organization wants good models to draw inferences from, they must construct them based on their own processes, people, systems and policies.

Further, the system and software development processes of these organizations need to be mature. Repeatability is necessary to have any reasonable degree of confidence in forecasts that are produced by software prediction models. It is the author's opinion that software reliability prediction is a form of software quality management (SQM). SQM is a CMM Level 4 Key Process Area. In order to be good at Level 4, an organization must first be operating at a solid Level 3 for some period. If not, a project has little chance of making meaningful use of a software reliability prediction.

Level 3 means that a software organization has applied a consistent process and achieved consistent results with that process across projects and within iterations of a single project. With this in hand, the organization can produce a Process Capability Baseline (PCB). The PCB establishes an organization's performance characteristics for cost, schedule and quality. Cost can be measured in labor hours, schedule in calendar months, and quality in defects delivered to system test and to the customer. The quality aspect of the PCB is where the link between the CMM and software reliability engineering exists.

From an SRE perspective, the project wants to know the level of defectiveness of the software code that is delivered to system software or system test. The software reliability engineers can then, theoretically, use this information to help guide test activities so that software reliability goals are achieved during system test.

Most organizations that emphasize the software CMM also have a software engineering process group (SEPG) that's responsible for software processes and other items such as deriving and maintaining the PCB. To be most effective, the SREs should work closely with the SEPG in implementing a software reliability program on a particular project. This means that if a software reliability prediction is going to be performed, the reliability engineer should obtain the process capability baseline values for the organization's processes. It should also utilize the SEPG's model(s) for defect prediction; this is required if the organization is at CMM Level 4.

V. SRE PERSPECTIVE

Based on experience working with several military projects with embedded mission critical software, the capability maturity model is given high visibility while software reliability engineering is low on the priority scale. Another experience on these programs is that software reliability engineering tasks are often handed off to the reliability or specialty engineering group. This sends the message that software reliability is a reliability engineering job as opposed to software engineering.

In reality, SRE tasks are fundamentally linked to both software and test engineering. SRE is just a quantitative perspective of software quality management.

VI. SRE VISION

In future applications, the author urges organizations to recognize the importance and contribution that software reliability engineers can have. Instead of being considered a stand alone discipline, SREs should be an integral part of the system independent verification and validation team. They should be part of the system and software engineering organizations and should be key players in system test planning, execution and analysis.

Ideally there wouldn't be a need to even refer to people as software reliability engineers. Instead, each important SRE task would be embedded in an organization's system and software engineering processes. Predicting and managing quality would be an inherent part of the software development process. Establishing software reliability goals for the system to be delivered would be common practice. Utilizing statistical testing methods to verify that reliability goals have been achieved would be second nature.

To achieve this future vision, software organizations must understand and recognize what software reliability is - a measure of software quality from an external, end user viewpoint. Software reliability practitioners must integrate themselves into the software engineering teams. The author believes that the best way to get there is to incorporate SRE tasks into an organization Software Quality Plan per the CMM. Only then will software reliability prediction become the science that practitioners are in search of.

REFERENCES

- [1] Lyu, Michael R., Handbook of Software Reliability Engineering, IEEE Computer Society Press, 1996.
- [2] Lakey, P. and Neufelder, A., System and Software Reliability Assurance Notebook, produced under contract to Rome Laboratory, 1996.
- [3] Software Engineering Institute Capability Maturity Model for Software, Version 1.1, 1998.