Development of Decision Models for Best Use of Software Testing Resources

Charles J. Campbell
University of Memphis
ccampbell@Memphis.edu

Judith C. Simon
University of Memphis
jsimon@memphis.edu

Ronald B. Wilkes
University of Memphis
rbwilkes@memphis.edu

Abstract

This focus of this paper is to present the first phase of development of decision models that could be used to determine the best uses of software testing resources. This model is a prototype that will evolve. As data and process information becomes available, the preliminary model presented in this paper will be expanded, with multiple tools used to determine the most appropriate and complete model. The ultimate model could be used to reduce overall costs while applying resources where they provide the greatest value throughout the systems development process.

1. Introduction

This project is designed to study software defect data as a means toward identifying where resources should be allocated most effectively to provide the highest quality of software product while reducing the overall cost of software testing. The mechanism for this study is development of a series of simulation models. The first phase of model development is presented in this paper. Ongoing work will involve extensive data collection regarding business processes followed by the use of simulation in the development of decision models.

2. Need for Research

Cost to an organization (both in dollars and in image) is significant when software defects are identified after installation at a client site. This project intends to identify areas where improvements in software testing resource allocations would provide added value to the organization.

Organizations that engage in software development and testing benefit significantly if their management team has tools to assist them in determining the most effective use of financial resources that might result in the fewest software errors in delivered systems. To be most effective, this tool needs to be developed after a thorough review of the specific organization’s testing data. Once developed, the tool will identify the specific phases and processes during the development life cycle where additional resources would provide the best return on investment and highest software quality. The use of this tool will provide a major reduction in the number and severity of software defects that exist after software testing. It will also reduce the overall cost of software testing by focusing on the appropriate process for a specific organizational environment.

To summarize, the purpose of this research is to increase software quality and reduce overall costs of software testing by focusing resources where they provide the most value.

3. Theoretical Foundations

The use of simulation modeling to ascertain best investment of software testing resources derives from two commonly known truths. First, variable analytic studies are often poor at facilitating understanding of complex contexts. As Coetzee [1] noted,

“We understand by immersing ourselves and our intelligence in complexity. There is something selfsustified in the way in which scientific behaviourism recoils from the complexity of life.”

Second, finding and fixing problems early in their life is easier and costs less than finding and fixing them later. McKeen [2] studied the relationship between where an error occurred and where it was caught, as well as the cost to remediate, and determined that the cost was lower when errors were caught earlier in the process.
The project explores optimal methods and timing of software testing in a very rich context defined by the nature of related business processes. Although there is strong rationale for using simulation to ascertain best methods and timing of resource allocations in software testing, no discussion of this approach or theory development related to this approach has been found in existing software testing literature.

4. Research Method

Initial modeling is based on known procedures for software testing, described in numerous publications as well as in interviews. Actual organizational data on software testing defects will be studied in the future for this model, including:

- types of software defect incidents
- numbers of software defect incidents
- locations where defects were identified
- locations where defects actually occurred
- specific developers/testers involved
- inspection cost per code segment at each phase
- delay/downtime cost
- rework/repair cost
  - process descriptions related to the entire development/testing/retesting effort to determine how defect issues at various stages are resolved

The data will be studied in detail to look for locations of greatest problems, as well as associated costs for correcting defects. This review will include a search for patterns, such as common defects, number of errors per tester, number of errors during a particular process, and combinations that might be targets for additional review regarding resource needs.

Simulation is the proposed tool for analysis of complex and interrelated systems such as the testing system under study. Complex simulation models are usually easily explained and understood. Further, once the existing model is built and verified, additional proposed scenarios may be built and tested to attempt to improve on the existing system.

Classic research models are frequently confirmatory and imply that the answers are known and just need to be validated. In contrast, simulation is exploratory and allows researchers to alter the model in a variety of ways and perform sensitivity analysis in developing a model. Simulation is better suited to the rich, complex, multivariable context of this particular environment.

5. Expected Contributions to Both Theory and Practice

This project contributes to research on software testing practices as they relate to management of the processes involved. Most existing research focuses on the coding and re-coding/testing aspects of software development rather than on effective management of the entire software testing process.

The development of this model for software testing will provide financial value for software testing management and improve the quality of the final software product. Once developed, tested, implemented, and evaluated, this model could be applied to the entire software development processes, resulting in additional financial benefit.

Due to the absence of existing theoretical work directly related to this project’s use of simulation, a major benefit of this somewhat unique project will be the contribution of a theoretical basis for future work in this area.

6. Model Development

The proposed model begins with jobs being generated according to some probability distribution. The job is then given two attributes. First, the number of errors, from one error to a maximum number of errors in the job, is generated according to another probability distribution and then the job is given a low testing priority. These two attributes, along with the time the job has been in the system, will stay with the job until it is completed and exits this phase of testing.

The job then enters a prioritized backlog (or waiting line). If there are no jobs in backlog and there is room for the job at the tester, the job immediately enters testing. If there is more than one job in the waiting line and the tester is ready for a job, then the jobs waiting are checked for priority. If there are jobs with high priority waiting, then the job with high priority and the longest waiting time in backlog will go into testing. If not, the low priority job with the longest waiting time in backlog will enter testing.

The time spent in the tester will be determined by a probability distribution. Upon exiting the tester, the attribute indicating the number of errors will be checked. If zero errors are indicated, then the job will exit this phase of testing and will serve as input to the next phase of testing or will be a completed job and ready for implementation. If one or more errors are indicated, the job will enter a waiting line for a reworking or correcting. Again, if the reworker is idle
and no jobs are waiting, the job will immediately enter service. The time spent in rework will be generated by another probability distribution. Upon exiting the rework station, the attributes of the job will be adjusted to indicate how many of the errors of this type were caught and the remaining number of errors will be adjusted. In addition, the priority of the job will be set to high and the job will reenter the backlog of jobs waiting for the tester.

For example, suppose it was determined that the number of jobs follows a Poisson distribution with mean 1.5 weeks. Further, suppose the number of errors can range from 0 to 10 such that it is equally likely for any of the outcomes to occur. The testing time follows an exponential distribution with mean 2 weeks, and the rework time follows an exponential distribution with mean 2.5 weeks. Finally, the number of errors repaired at the rework station follows a discrete distribution with outcomes from 1 to the number of errors in the attribute with probability that each is equally likely to occur.

Without loss of generality, let us assume that the process begins with the first job occurring at time 0. The error distribution is sampled and the number of errors in the job, say 6, is attached to the job. In addition, the priority of the job is set at a low level. The job then enters the backlog and checks to see if there are other jobs waiting. If so, it is placed at the end of the line for testing. The tester then polls the backlog of jobs waiting to be tested. Given there are jobs waiting, it will select the job with highest priority and longest waiting time to service. If there are no jobs waiting, the tester will be idle until a job arrives. Once a job is in testing service, the testing time for the job will be generated which, in turn, will determine the time the job exits the tester. The number of errors attribute is then checked. If there are zero remaining errors, the job exits this phase and is ready to enter the next phase of testing.

If there are still errors to be corrected, the job enters the backlog list for the reworking station. The reworker polls the backlog and if there is a job present takes it into service. The rework time is generated for the job to determine when it will exit the rework station. When the job leaves the rework station, the number of errors corrected is determined and the remaining number if errors is changed in the attribute. The job then reenters the backlog of testing. (See Figure 1.)

![Figure 1. Preliminary Model](image)

7. Follow-up Research

This study is a first stage in the development of simulation of business processes encompassing the
entire systems development process. The model is currently being expanded to account for the variety of interpretations of the meaning of software testing. Problems related to communications of requirements specifications can be more deadly than code errors. Determination of the full variety of fail points in the total process is essential in avoiding suboptimal solutions.

Additional complexity regarding the nature of software development artifacts will be incorporated into the model. Differential distributions of defects and defect types will be analyzed to account for the development of different system types in different organizational contexts.

Development of the model will continue in an effort to capture the richness of the software development environment.

8. References
