

**ABSTRACT # 011-0028**

**SOFTWARE DEVELOPMENT WHEN TESTING IS IMPERFECT OR INCOMPLETE**

Tapan P Bagchi

*Indian Institute of Technology Kharagpur, Kharagpur, WB India 721302*

[bagchi@vgsom.iitkgp.ernet.in](mailto:bagchi@vgsom.iitkgp.ernet.in)

+91-9775194566

**POMS 20<sup>TH</sup> ANNUAL COONFERENCE**

**ORLANDO, FLORIDA USA**

**MAY 1 TO MAY 4, 2009**

# SOFTWARE DEVELOPMENT WHEN TESTING IS IMPERFECT OR INCOMPLETE

Tapan P Bagchi

*Indian Institute of Technology Kharagpur, Kharagpur, WB India 721302*

*Abstract* This study analytically assesses the statistical risk of releasing defective software that cannot be exhaustively tested, and of needlessly testing defect-free software. Specifically, it quantifies the probability of committing Type I ( $\alpha$ ) and Type II errors ( $\beta$ ) in software development when one may release software that still is faulty or do needless testing since the test methods themselves may not be perfect. The study uses Truncated-Poisson and geometric distributed path lengths and Bernoulli-type inspection errors to link  $\alpha$  and  $\beta$  to software design features, the development philosophies employed, and certain aspects that include code quality, cyclomatic complexity and the average length of basis paths. For risk reduction this study finds quantitative justification for raising test coverage, perfecting the test methods, the adoption of recent innovations and programming methods such as component-based design, SOA and XP as ways to raise the likelihood that the product developed will be fault free. Results are relatively robust with respect to the probability distributions assumed.

**Keywords:** Defect modeling, Imperfect inspection, Software Engineering, SW Testing, Type I and Type II Errors

## 1. Introduction

The unending frustration with SW disasters, release of buggy SW, and in general the large contrast between current SW development methods and hardware design and manufacture has led to a great

deal of soul-searching in the IT community [1]. The focus within IT is primarily on improving the effectiveness of testing (inspection) methods. Models of SW reliability as it is influenced by design and development methods have started to appear. Still, the risk of releasing a defective SW or needlessly debugging a perfect SW is yet to be linked to SW design parameters, and to the quality of the test methods employed to check the codes. This paper attempts to partially fill that gap by analytically modeling the SW development and testing processes—not done to date.

## **2. Risks in passing judgment on Software based on Limited Testing**

In statistical hypothesis testing it is assumed that sometimes it is not possible to completely remove uncertainties surrounding a decision situation [2]. This leads to working with limited information about the “state of nature”, which in turn can lead to four distinct possibilities in the decision situation confronting a decision maker. First, if the true state of nature is that a hypothesis is correct, and the test accepts that hypothesis as true, no error is committed. In this paper’s view, the risk in not testing all basis paths and terminating the test is the expected loss from the *probability* of committing a Type II error. The reverse (committing a Type I error in SW testing) is possible if the testing methods are not perfect. In setting up statistical hypothesis testing one attempts to design the tests so as to minimize the two risks—by minimizing  $\alpha$ , the probability of committing a Type I error, as well as  $\beta$ , the probability of committing a Type II error.

## **3. The Probability of committing Type I Error**

As noted in Table 1, a Type I error is committed when on the basis of limited or faulty testing one erroneously rejects SW that is truly defect-free. This can happen if the test cases are improperly designed or if the tests are improperly run and therefore yield incorrect results. The consequence of committing a Type I error is a false signal, delays and waste of resources on pointless “debugging”

adding to development cost, hence product cost, thereby hurting the software developer's competitiveness through higher cost of quality [3].

Table 1 Type I and Type II errors in hypothesis testing

		True State of Nature	
		Software is truly defect-free	Software has defects
Your decision based on the tests conducted on the software	Declare that "Software is defect-free"	Decision is correct	Type II error: You have erroneously accepted a defective SW as good
	Declare that "Software is defective"	Type I error: you have incorrectly rejected a good SW	Decision is correct

#### 4. The Probability of committing Type II Error

In contrast to Type I error, committing a Type II error is quite serious as it may lead to the release of a defective SW in the marketplace. A defective SW may lead to rework and returns, laborious maintenance, and it may even harm the client's operations financially, cause unsafe actions, or delay operationalizing critical functions [2]. Therefore, one must make every effort to prevent or minimize the probability of committing a Type II error, i.e., releasing a SW with unsatisfactory or unacceptable performance. We address its estimation in this section.

If the Software's cyclomatic complexity (the number of basis paths present in the code [1]) is CCN. Assume that the code has been partially tested up to T paths (T simulating test coverage [4]).

Hence, the number of basis paths yet to be tested is  $CCN - T$ ,  $\varepsilon$  is average code quality or the probability that a given node is defective and  $\eta$  is the average length (measured by the count of nodes on it) of a randomly selected and untested basis path, then  $P[\text{Type II error in partially testing the SW}]$  will be  $P[\text{one or more defective nodes in } \eta(CCN - T) \text{ nodes}]$

$$= \sum_{r=1}^{\eta(CCN-T)} P[r \text{ defectives in } \eta(CCN - T) \text{ nodes}] \quad (1)$$

$$= 1 - (1 - \varepsilon)^{\eta(CCN - T)} \quad (2)$$

It is trivial to see that  $\beta$  reduces as  $\varepsilon$  is reduced (i.e., code quality is improved), or if cyclomatic complexity (CCN) is lowered.

If the exhaustive testing is successfully completed, i.e., all CCN basis paths have been successfully tested,  $T = CCN$ , hence

$$P[\text{Committing a Type II error}] = 0.$$

If no tests have been run yet,

$$P[\text{Committing a Type II error}] = 1 - (1 - \varepsilon)^{\eta \text{ CCN}}$$

And if  $\varepsilon = 0$ ,

$$P[\text{Committing a Type II error}] = 1 - (1)^{\eta(CCN - T)} = 0.$$

Each result above is consistent with our expectations. Figure 1 displays the typical sensitivity of  $P[\text{Type II error}]$  to  $\varepsilon$  and the fraction of total basis paths tested. Figure 2 shows the sensitivity of  $P[\text{Type II Error}]$  to CCN and  $\eta$ .

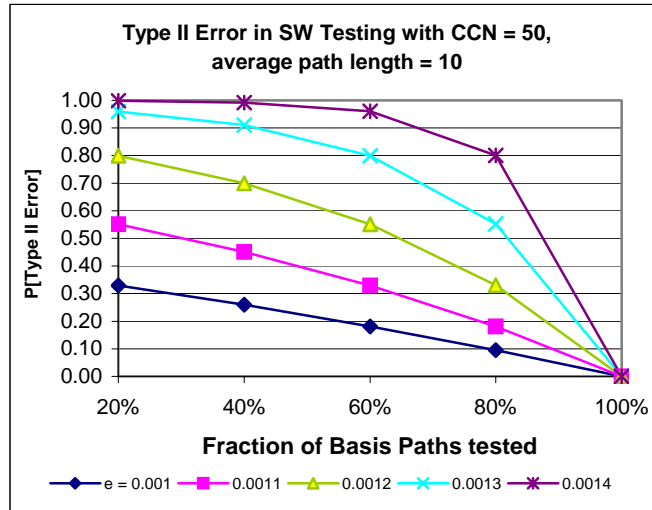


Figure 1 Probability of accepting a defective SW when testing has not covered all CCN basis paths

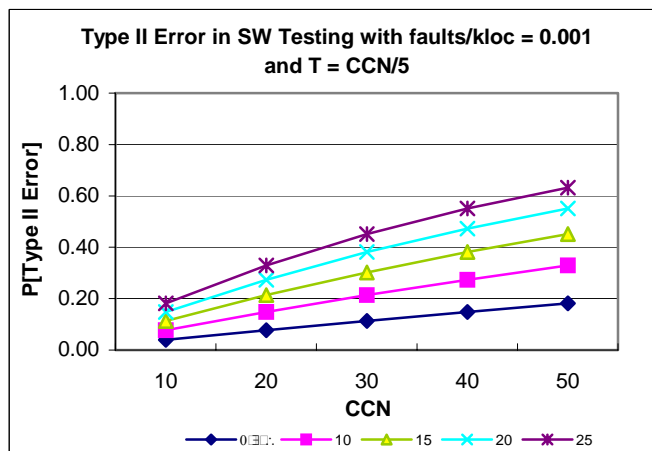


Figure 2 Probability of declaring a defective SW to be “good” as function of CCN and Average Length ( $\eta$ ) of Basis Paths

## 5. A Model for SW Defects and Testing Strategies based on Size

### 5.1 Testing a Path

We investigate a different issue in this section—the probability of missing defects in a single randomly run test. In such a test, only one basis path is executed.

The initiator of an execution path (Figure 3) is the *starting node* where the initial input is given. Subsequently, based on the SW's response/behavior/design, the user interacts and provides further inputs. In normal operation, through state transitions, the SW moves from decision node to decision node along paths and produces the desired output (Figure 3). We use here the definitions given in Section 4.3.1 by Jorgensen [4] of nodes and edges. Figure 3 is an adaptation of various program graphs provided by Jorgensen [4].

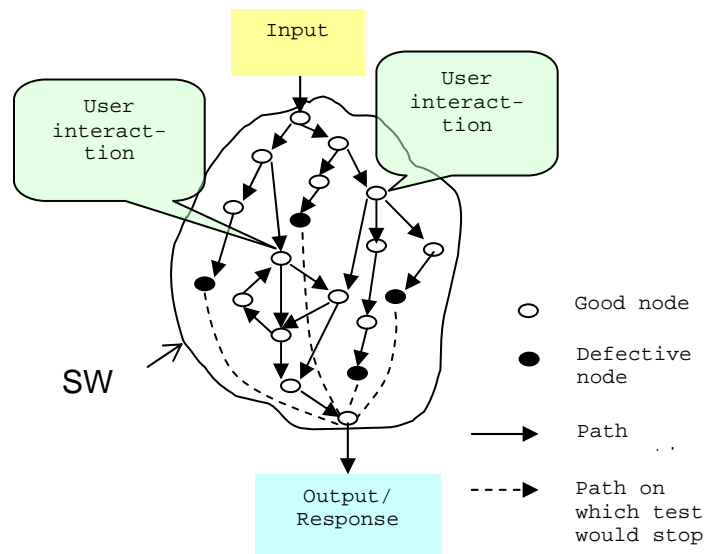


Figure 3 A Simplified “Node-Path” Representation of a SW

## 5.2 Assumptions

1. Basis paths traverse every possible execution paths connecting input to output through the maze of nodes existing in the SW system.

2. A SW is defective if any node in it is defective (shown as ● in Figure 3). This implies that if some path is found to have a defective node on it, it would make little sense to proceed further without rectifying that defective node.
3. A SW is acceptable if *every* basis path contains only good nodes (○ in Figure 3).

In limited budget testing or in testing a large SW in which all possible basis paths may not be tested, the exercise can be a “lucky” one in that the few paths tested do not contain any defective node(s). Such a “lucky” test will falsely declare the faulty SW to be good (Type II error in testing).

## 6. Derivation of P[Type II Error] for a single randomly run test

As shown in Figure 3, we assume that defective nodes in a SW to be subjected to white box testing are randomly distributed within the coded bulk. Code quality determines the likelihood ( $\epsilon$ ) of a node being defective, which again for simplicity is assumed to be identical for all nodes. Further, we stipulate that a faulty SW will contain at least one defective node in it. Note that the length of basis path (i.e. the number of nodes it traverses in going from input to output) cannot be zero. Hence by eliminating the pathological case of  $n = 0$  one may model path length by the Truncated Poisson distribution [5],

$$P_n = \frac{\lambda^n e^{-\lambda}}{1 - P_0}, \quad n = 1, 2, 3, \dots \quad (3)$$

Here  $P_0 = e^{-\lambda}$  is used to normalize  $P_n$  as suggested by Rider [4] such that

$$\sum_{n=0}^{\infty} P_n = 1$$

The distribution given by (3) has the mean  $\eta$ , which equals  $\lambda/(1 - e^{-\lambda})$ . Therefore,

$$\begin{aligned}
& \text{P[SW with code quality } \varepsilon \text{ passes a single test tracing a random path]} \\
&= \sum \text{P[path length} = n] \times \text{P[path is free of defects/path length} = n] \\
&= \sum P_n \times \text{P[path is free of defects/path length} = n] \\
&= \sum_{n=1}^{\infty} \frac{\lambda^n e^{-\lambda}}{1 - P_0} (1 - \varepsilon)^n
\end{aligned}$$

On rearranging the terms we get

$$\begin{aligned}
& \text{P[a SW with quality } \varepsilon \text{ passes a single random test]} \\
&= \beta = \frac{e^{-\lambda\varepsilon}}{1 - P_0} [1 - e^{-\lambda(1-\varepsilon)}] \tag{4}
\end{aligned}$$

where  $\varepsilon = \text{Prob[a node is bad]}$ , and  $P_0 = e^{-\lambda}$ .

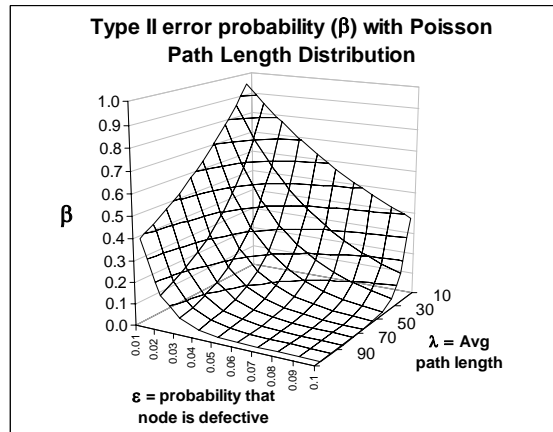


Figure 4 Sensitivity of Probability of Type II Error to  $\varepsilon$  and  $\eta$

Figure 4 shows the sensitivity of  $\beta$  to  $\eta$  here is the average number of nodes on a typical basis path and  $\varepsilon$  representing code quality (faults/kloc). Note that the effects of  $\varepsilon$  and  $\eta$  are *not* additive—they *interact* [2], hence would require special methods to optimize actions for defect

reduction. We recall that we assumed the distribution of path lengths (i.e., the number of nodes on a basis path) to be Truncated Poisson and the quality of a node to be a Bernoulli distribution with parameter  $\varepsilon$ . It is theoretically possible to derive expressions resembling (4) if some other distributions govern these random dispositions of the code.

## 7. Imperfect Inspection

SW quality assurance (SQA) remains still at the very early stages of QA when one observes where hardware QA or process control methods are today. Many practitioners and users say that SW failures are no less serious now even in air or space travel than a defective O-ring or protective tile. A critical step in SW development continues to be *testing*, executed by pre-designed test “cases”, often themselves less than perfect [6]. To study this we set up a simple Bernoulli-type [7] model for imperfect inspection (Figure 5). Thus  $\theta$  = Probability[misclassify a good node as defective] and  $\phi$  = Probability[misclassify a defective node as good]. When basis paths are Truncated Poisson distributed with parameter  $\lambda$ ,

$$P[\text{Type I error}] = \alpha = \left[ e^{\lambda(1+(1-\varepsilon)\theta)} - 1 \right] \frac{e^{-\lambda}}{1 - e^{-\lambda}} - 1 \quad (5)$$

And P[Type II error in testing a single basis path] equals  $\beta$  when

$$\beta = 1 - \frac{1 - e^{-\lambda(1-\varepsilon\phi)}}{e^{\lambda\varepsilon\phi} (1 - e^{-\lambda})} \quad (6)$$

Allowable bounds on testing errors  $\theta$  and  $\phi$  may then be established as

$$\theta \leq \frac{\ln(1 + \alpha(1 - e^{-\lambda}))}{\lambda(1 - \varepsilon)} \quad (7)$$

$$\phi \leq -\frac{\ln(1 - \beta(1 - e^{-\lambda}))}{\lambda\varepsilon} \quad (8)$$

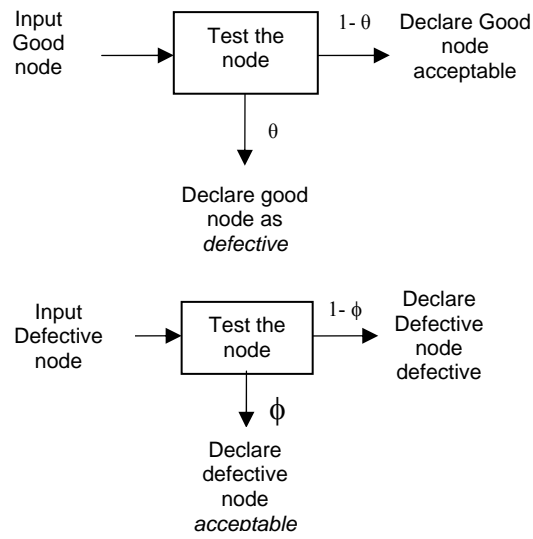


Figure 5 Node misclassification probabilities

## 8. Conclusions

This study was motivated by existing unaddressed issues in the cost of poor software quality [8].

The actionables are:

1. Minimize mistakes in test case design.
2. Maximize the coverage in SW testing.
3. Build SW in small components using well-crafted and well-tested reusable parts (~ 100% coverage) and then assemble them into a SW.
4. Strive to keep SW testing errors and node misclassifications low. This paper provides quantitative allowable bounds on testing errors similar to Gage R&R studies for hardware quality assurance [7].

## References

- [1] R.S. Pressman: *Software Engineering, A Practitioner's Approach* (McGraw-Hill, 2005).
- [2] John A. Rice: *Mathematical Statistics and Data Analysis*, 3<sup>rd</sup> ed., (Thomson, 2007).
- [3] Quality Cost Analysis: Benefits and Risks <http://www.kaner.com/qualcost.htm> opened January 4, 2009.
- [4] P.C. Jorgensen: *Software Testing: A Craftsman's Approach*, 2<sup>nd</sup> ed, (CRC Press, 2002).
- [5] P.R. Rider: Truncated Poisson Distributions, *Journal of American Statistical Association*, Vol. 48(264).
- [6] [http://www.nws.noaa.gov/ost/SEC/AE/Testing/to8/TO8-NWS-TestCases/IV&V\\_Test\\_Case\\_TO8\\_8004.pdf](http://www.nws.noaa.gov/ost/SEC/AE/Testing/to8/TO8-NWS-TestCases/IV&V_Test_Case_TO8_8004.pdf)
- [7] F.M. Gryna, R. Chua and J. Defeo: *Juran's QPA for Enterprise Quality*, 5<sup>th</sup> ed (Tata-McGraw-Hill, 2007).
- [8] <http://www.cse.buffalo.edu/~hungngo/SCE-Papers/p67-slaughter.pdf> accessed January 29, 2009.