# On the Feasibility of Using Operational Profiles to Determine Software Reliability in Extreme Programming

**Nachiappan Nagappan**
**Mark Sherriff**
**Laurie Williams**

# On the Feasibility of Using Operational Profiles to Determine Software Reliability in Extreme Programming

Nachiappan Nagappan, Mark Sherriff, Laurie Williams
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
{nnagapp, mssherri, lawilli3}@unity.ncsu.edu

## Abstract

*By using the test-driven development practice of the Extreme Programming methodology, programmers write extensive automated unit and acceptance tests. Our research objective is to further leverage the benefits of these extensive automated test cases by funneling the TDD test results to an automated means of estimating project reliability. Our initial research approach was to use an operational-profile model for reliability estimation. We investigated the feasibility of the operational profile approach by visiting three industrial locations where XP was practiced. These case studies revealed the operational profile approach is not feasible.*

## 1.0 Introduction

Agile software development methodologies [5, 10], such as Extreme Programming (XP) [2] are rapidly growing in popularity. These methodologies are based on iterative enhancement, a technique which was introduced in the 1970-ies, e.g. [1]. Of recent, organizations long known for producing high-quality safety- and mission-critical applications, such as Motorola [4], IBM [11], and NASA [25], are experimenting with XP and/or XP-like methodologies. In this section, we describe Extreme Programming, Test-Driven Development and Software Reliability Engineering.

### 1.1 Extreme Programming

Extreme Programming (XP) is a set of agile software development practices that focuses on developing software whose requirements could be volatile and on continuously meeting the needs of the customer in this changing environment [2]. The methodology is called "Extreme" because each of the practices is to be taken to its logical extreme to better meet the customer's needs. There are twelve main practices that are considered the core of XP, ranging from how to do initial planning to how to utilize a programmer's time. They are:

- Small releases – An XP project should be released often, with enhancements coming over time.
- System metaphor – Using a common metaphor, developers have an overall idea how the project will come together.
- Planning game – Developers and customers and developers jointly determine the product requirements and the plan for the next iteration.
- Sustainable pace – Working an excessive amout of time can increase the number of defects in a programmer's code. Thus, the work week is limited to minimize this risk.
- Pair programming – Two programmers work together on the same code at the same time at the same computer.
- Test-driven development – Through unit testing of individual components, code is verified immediately. Additionally, automated acceptance test are written to validate the successful completion of customer requirements.
- Simple design – XP programmers do the simplest thing that will still work so as to not over-complicate a program.

- Refactoring – During code implementation, extra, duplicate, or poor-ly structured code might be written. It should be continuously removed and/or restructured.
- Collective code ownership – A team member can make a change to any piece of code in the project at any time as long as all of the tests still pass.
- Continuous integration – Code is routinely integrated (at least daily) into the code base so that collective code ownership is possible.
- On-site customer – An on-site customer is needed in order to clarify or change any of the requirements at any time.
- Coding standards – Since anyone can work on any piece of code, all developers have to be able to understand what has been written easily. [2]

The core practices that are focused on in this report is the planning game and test-driven development. During initial phases of planning, the developers and the customers work together to create user stories. These user stories generally contain one to threes sentence that describe a function of the system. To help clarify these user stories, the customers define acceptance test cases. These test cases describe a sequence of actions which would simulate the fulfillment of a customer requirement and what thr desired outcome of the sequence should be. The developers use this information to better understand the user stories that they are associated with in order to develop the system more closely to what the customer wants. [2]

### 1.2 Test-Driven Development

The XP software development methodology [2] is a test-centric approach to software development. With XP, developers follow the test-driven development (TDD) [3] practice, incrementally writing unit and acceptance test cases throughout the software development cycle. The writing of these test cases is based upon the implementation of user stories. These test cases are generally written using one of the xUnit[1] automated testing tools.

TDD software engineers develop production code through rapid iterations of the following (as indicated in Figure 1). TDD software engineers develop production code through rapid iterations of the following:

- writing a small number of automated unit test cases;
- running these unit test cases to ensure they fail (since there is no code to run yet);
- implementing code which should allow the unit test cases to pass;
- re-running the unit test cases to ensure they now pass with the new code;
- refactoring [8] implementation and test code, as necessary; and
- periodically (e.g., once a day if the code base is small enough) re-running all the test cases in the code base to ensure the new code does not break any previously-running test cases (i.e. frequent regression testing).

[1] e.g. JUnit (www.junit.org) for Java.

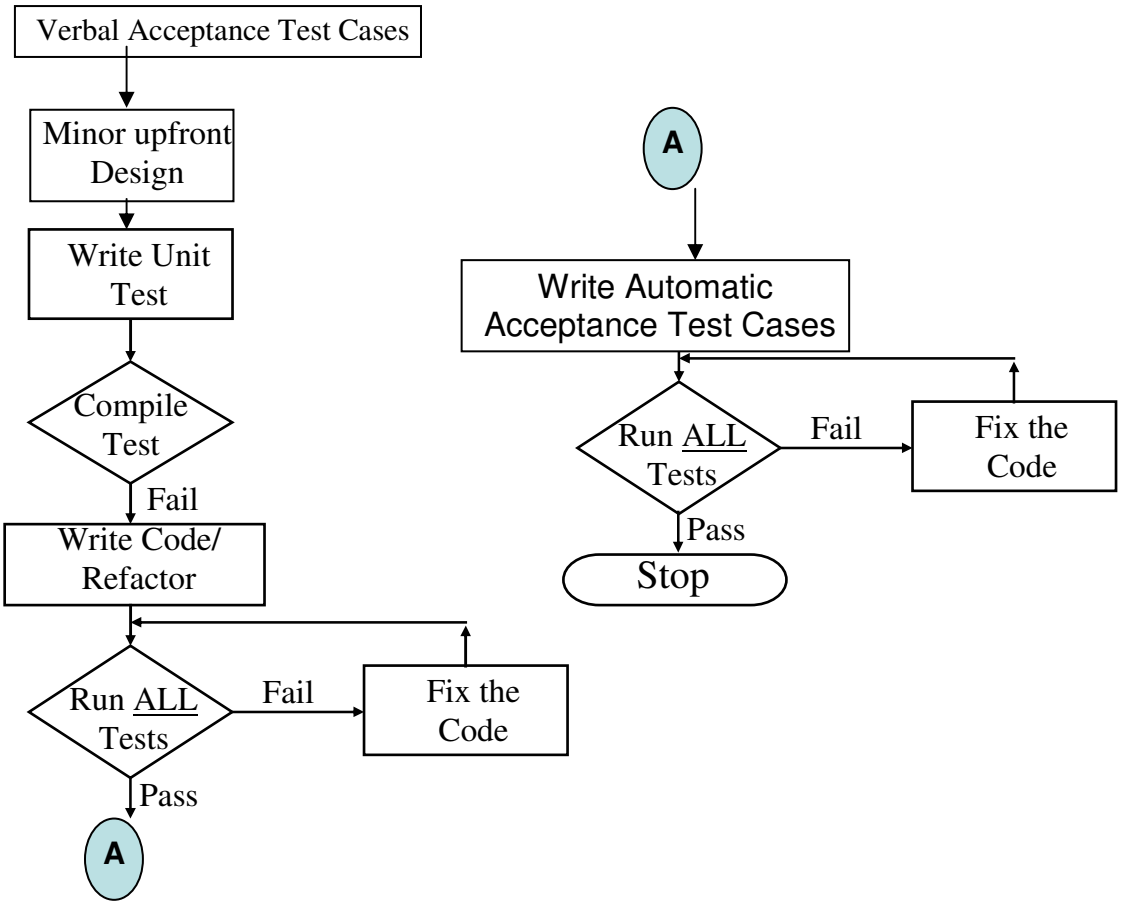



**Figure 1: Test-Driven Development**

### 1.3 Software Reliability

Software reliability is defined as the probability that the software will work without failure for a specified period of time [14]. There are several software reliability models available, each of which has their own advantages and disadvantages. Several software reliability growth models are described by Non-homogenous Poisson Processes (NHPP) [26]. These models, like the Musa [15, 16] and the Goel-Okumoto[9, 16] models, require failure intensity or an inherent number of faults. Since in XP, there are "no failures", reliability models based on failure rates cannot be used as our reliability estimation model. Instead, "no failure" estimation models need to be developed, as described in [6, 7, 12].

Researchers have developed and utilized several different models for estimating reliability. They range from a very simplistic Nelson's type model, to more sophisticated hypergeometric coverage-based models, to component-based models, and object-oriented models [17, 19]. We found the Markov Chain Reliability Model (MCRM) especially well suited for the satisfied measure of reliability of XP projects. The MCRM is a finite-state, discrete-parameter model based on statistical testing; MCRM is often used as a means of estimating reliability for Cleanroom Software Engineering [13] projects. A full description of MCRM is beyond the scope of this paper, but can be found in [18, 20-23].

## 2.0 GERT Model

An operational profile is a set of operations and their probabilities of occurrence [14]. We developed initial model for estimating reliability of XP projects and incoroporated it into a tool called the "Good Enough Reliability Tool" (GERT) for XP [24]. The model was based on the operational profile of the software tied to the customer user stories and the developer's acceptance test cases by the following relationship:

$$\hat{R} = \sum_{j=1}^{m} \sum_{i=1}^{n_j} us_j \, at_{ij} \, x_{ij}$$

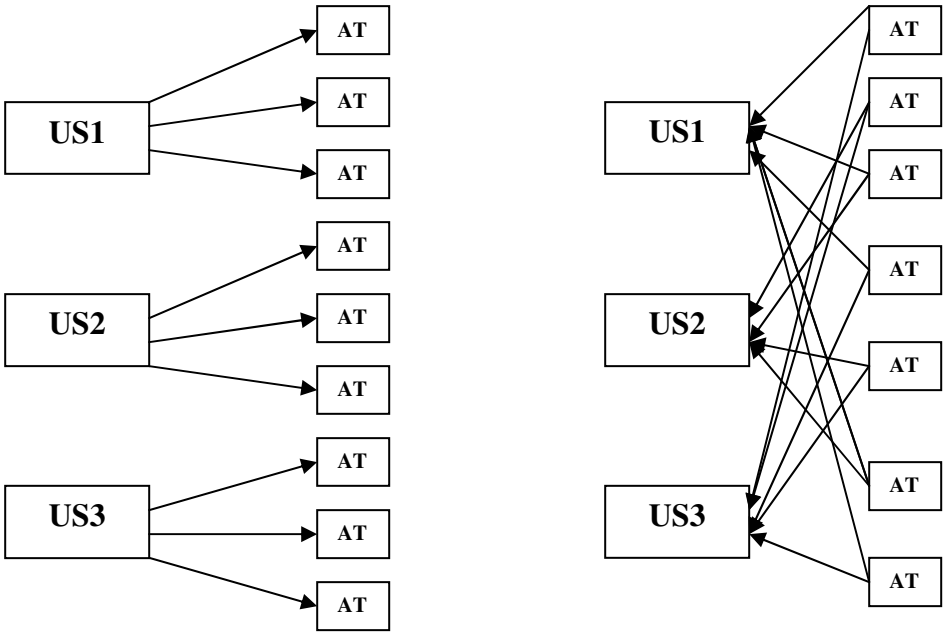
where the acceptance test execution score is $x_{ij}$. If Acceptance Test Case i of User Story j passes, the score is 1, otherwise it is 0. In order to further illustrate the working of the GERT model we present an example application of the GERT model to sample banking problem in the following section.

## 3.0 Case Studies

To determine the feasibility of using the GERT tool with XP practitioners, we performed three case studies. The case studies were carried out at three industrial locations: RoleModel Software, John Deere and Nortel Networks. The meeting minutes and other observations are found in Appendix A, B.1, B.2, and C respectively.

In our model, we had assumed there was a correspondence between user stories and their automated acceptance test cases. These came from the idea that, during the planning game process, the developers and customers would create the user stories and then write acceptance tests that dealt specifically with that certain user story. This way, developers could prove to the customer that a given user story was completed by demonstrating that all of the acceptance tests passed. Thus, we expected that there would be a set of user stories and acceptance tests that were always available to prove that any given story was completed.

However, we found that developers tend to aggregate a minimal set of acceptance test cases, each used to satisfy several of the customer's user stories. For example, the developers could have an acceptance test that tested a certain initial condition of the program. After this test passes, the developers move on to other requirements that need to be developed, following the next sets of user stories and acceptance tests. However, we found that once they have passed a test to demonstrate the completion of a user story, developers consider that they do not need to keep that test in its current form any more. As more functionality is added to the system, developers may alter/add additional conditions to acceptance test cases so that one acceptance test may be used for multiple user stories, as shown in Figure 2. Therefore, developing a mapping between user stories and acceptance test cases was shown to not be practical.




**Initial GERT Model Assumption** **Reality**

**Figure 2: Feasibility of Operational Reliability Model**

## 4.0 Conclusions and future work

Based on our three industrial case studies, there are several problems with the feasibility of using an operational profile-based model for estimating reliability with XP:

- Developers would need to change their automated acceptance testing habits to eliminate the reuse and alteration of test cases, as described in Section 4. We do not feel that XP practitioners would accept this change because their focus is on getting as much done as quickly as possible, minimizing overhead.
- XP advocates small (no more than three-month) releases. The operational profiles may change drastically in each of these releases. For example a feature that had a high operational occurrence in the previous release may be eliminated because of the customers change in requirements. Allowing for such frequent change in the operational profile would be difficult to maintain in our model.
- It is almost impossible to tie together the customer user stories and developer's acceptance test cases as developers tend to aggregate acceptance test cases to satisfy several of the customer's user stories.
- The GERT model assumes all the user stories are independent. There exists a certain degree of parallelism in the test cases, and this might cause the GERT model to be inaccurate.

We intend to use a non-operational model in future to further leverage the benefits of these extensive automated test cases by funneling the TDD test results to an automated means (1) of estimating project reliability and (2) of providing feedback to developers on the quality of their test cases based upon historical values.

## References


1. Basili, V.R. and A.J. Turner, *Iterative Enhancement: A Practical Technique for Software Development.* IEEE Transactions on Software Engineering, 1975. **1**(4).
2. Beck, K., *Extreme Programming Explained: Embrace Change*. 2000, Reading, Massachusetts: Addison-Wesley.
3. Beck, K., *Test Driven Development- by Example*. 2003, Boston: Addison-Wesley.
4. Bowers, J., et al. *Tailoring XP for Large System Mission-Critical Software Development*. in *Extreme Programming/Agile Universe*. 2002. Chicago, IL: Springer.
5. Cockburn, A., *Agile Software Development*. The Agile Software Development Series, ed. J. Highsmith. 2001, Reading, Massachusetts: Addison Wesley Longman.
6. Duane, J.T., *Learning Curve Approach to Reliability Monitoring.* IEEE Transactions on Aerospace, 1964(563-566).
7. Ehrenberger, W., *Statistical Testing of Real-Time Software*, in *Verification and Validation of Real-Time Software*, W. Quirk, J., Editor. 1985, Springer-Verlag: New York. p. 147-178.
8. Fowler, M., et al., *Refactoring: Improving the Design of Existing Code*. 1999, Reading, Massachusetts: Addison Wesley.
9. Goel, A., Okumoto, K., *Time-Dependant Error-Detection Rate Model for Software Reliability and other Performance Measures.* IEEE Transactions on Reliability, 1979: p. 206-211.
10. Highsmith, J., *Agile Software Development Ecosystems*. The Agile Software Development Series, ed. J. Highsmith. 2002, Boston, MA: Addison-Wesley.
11. Krebs, W. *Turning the Knobs: A Coaching Pattern for XP Through Agile Metrics*. in *Extreme Programming/Agile Universe*. 2002. Chicago, IL: Springer.
12. Miller, K.W., Morell, L.J., Noonan, R.E., Park, S.K., Nicol, D.M., Murrill, B.W., Voas, J.M., *Estimating the Probability of Failure When Testing Reveals no Failures.* IEEE transactions on Software Engineering, 1992: p. 33-43.
13. Mills, H.D., R.C. Linger, and A.R. Hevner, *Box Structured Information Systems.* IBM Systems Journal, 1987. **26**(4): p. 395-413.
14. Musa, J., *Software Reliability Engineering*. 1998: McGraw-Hill.
15. Musa, J., *Theory of Software Reliability and its Applications.* IEEE Transactions on Software Engineering, 1975: p. 312-327.
16. Musa, J., Ianino, A.,Okumoto, K., *Software Reliability:Measurement, Prediction, Application*. 1987, New York: McGraw Hill.
17. Musa, J.D. and W.W. Everett, *Software-Reliability Engineering: Technology for the 1990s.* IEEE Software, 1990. **7**(November): p. 36-43.
18. Rauakorpi, M., *Application of Markov Chain Techniques in Certification of Software, Master's Thesis.* Department of Mathematics and Systems Analysis, Helsinki University, 1995.
19. Vouk, M.A. *Software Reliability Engineering, a Tutorial*. in *1998, 1999, and 2000 Annual Reliability and Maintainability Symposium*. Anaheim, CA, January 1998; Washington, DC, January 1999; Los Angeles, January 2000.
20. Whittaker, J.A., *Certification Practices*, in *Cleanroom Software Engineering Practices*, J.A. Whittaker, Editor. 1997, Idea Group Publishing: Harrisburg.
21. Whittaker, J.A., *Markov Chain Techniques for Software Testing and Reliability Analysis, PhD Dissertation.* Department of Computer Science, University of Tenn., 1992.
22. Whittaker, J.A. and J.H. Poore, *Markov Analysis of Software Specifications.* ACM Transactions on Software Engineering Methodology, 1993. **2**(1): p. 92-106.

23. Whittaker, J.A. and M.G. Thomason, *A Markov Chain Model for Statistical Software Testing*. IEEE Transactions on Software Engineering, 1994. **20**(10): p. 812-824.
24. Williams, L., Wang, L., Vouk, M.A. *"Good Enough" Reliability for Extreme Programming, FastAbstract*. in *International Symposium on Software Reliability Engineering*. 2002. Anapolis,MD.
25. Wood, W.A. and W.L. Kleb. *Extreme Programming in a Research Environment*. in *Extreme Programming/Agile Universe*. 2002. Chicago, IL: Springer.
26. Yamada, S., Osaki,S., *Software Reliability Growth Modeling:Models and Applications.* IEEE Transactions on Software Engineering, Dec 1985: p. 1431-1437.

# Appendix A

Company : RoleModel Software[2], Holly Springs, NC

Date: April 9 2003

NC State Participants:

Dr.Laurie Williams
Nachiappan Nagappan
Mark Sherriff

RoleModel Software Contact:

Ken Auer

Brief Outline

The project we were interested in was done by RoleModel for CipherOptics . This project followed the Extreme Programming framework (XP) and employed pair programming.

Points of the Meeting:

- It is difficult to find a direct relationship or to retrace the relationship between the user stories and the test cases. There exists a varying degree of parallelism that keeps changing. The testing of user stories gets accumulated or combined together in the acceptance testing stage, and there is no way of tying the tests back to the initial user stories.
- The developers used the unit test cases to drive the framework of the system.
- They also demonstrated the use of SWTUnit, a program that runs/simulates the actual execution of the program. The code for this is open source.
- As an example to highlight point 1, they showed an example of the test case to *add an appliance.* Though the requirement to add an appliance appears in the user stories, a test case which validates only this requirement cannot be found in the testing framework because this *add an appliance* functionality is used in several test cases that need to use this functionality. Thus, the individual user story is almost impossible to trace to a test case. This is further complicated as there are a number of releases and once a releases passes all it test cases then developers further combining/modifying these test cases to check for the new version.
- There was a discussion about the relationship between the acceptance test cases and the unit test cases. When the unit test cases become very complex, then either the unit test case is very tough or it is not possible at all.
- Several questions arose as to:
    1. How much more system testing is to be done before the software is delivered?
    2. What are good guidelines for domain-specific projects?
    3. What are valid and useful code coverage metrics?

[2] www.rolemodelsoft.com

Appendix B.1

Company : John Deere[3], Cary, NC

Date : April 14 2003.

NC State Participants:

Dr.Laurie Williams
Nachiappan Nagappan
Mark Sherriff

John Deere Contact:

Doug Taylor

Brief Outline:

All the discussions below were done with the software division of John Deere. The entire discussion on reliability also included hardware reliability as an important component. This is because most of the software that John Deere develops is for use by their hardware products namely Tractors and Farm Equipment

Points of the Meeting:

- JSP is one of the primary languages used in Mr. Taylor's department. The use of JSP leads to having little control over the browsers used to run their programs. The interface must be from a user aspect at the domain level.
- Downtime is very critical to the company, and thus manual testing is done in addition to automated testing.
- The software part of the company is technology centric. They mainly use JMS/Swing user interfaces. A proof of concept and manual testing is also done to verify their initial plans.
- Despite their careful planning, there is no acceptance check by user for each feature. Due to high system demand, more formal testing (i.e. testing under real load conditions) is required than usual.
- The developers gave a demonstration of their current reliability tool. It is browser based with a database backend. Both hardware and software component reliability is integrated into the tool. The reliability keeps on growing or decreasing depending upon the values/results of the test carried out. In short, the more test cases that pass, the more is its contribution towards the reliability of that component.
- The tool also lets users measure how the actual reliability of the component and the overall product reliability is moving with respect to the expected value. The tool also provides information on how many additional tests are required to attain the desired reliability metric.

[3] www.deere.com

Appendix B.2

Company : John Deere, Cary, NC

Date: April 26 2003

NC State Participants:

Dr.Laurie Williams
Nachiappan Nagappan
Mark Sherriff

John Deere Contact:

Doug Taylor
Dwight DeDoncker

Brief Outline:

Dwight is a reliability expert at John Deere who deals primarily with the concept of reliability of both hardware and software components. He provides guidance to development groups from a mathematical standpoint as to the type of reliability model/models that can be used for evaluating the reliability of individual components (both hardware and software) and estimating reliability of systems upon integration of the hardware and software components.

Points of the Meeting:

- They have five categories of testing - review, design analysis, lab tests, field performance tests (may be lab with components for interaction), and field durability (full machine).
- All the mathematical models discussed were based on work done by Dr. Larry Crow. Most of this work has been published at RAMS (Reliability and Maintainability Symposium). It deals with the reliability growth of repairable systems (Fix and Play).
- Systems are classified either as repairable or non-repairable systems. Mostly the Weibull process is the only one used for reliability estimation. However, all system components finally relate to blocks of non-repairable systems.
- BlockSim by Reliasoft is used by the team. Accelerated testing is performed which is essentially simulation of high loads (or) simulation of a long period of operation accelerated (i.e. contracted) over a small time interval to observe the system performance. Reliability growth of repairable systems is studied. AMS is contracted as the software arm that puts GPS systems in combines. There is always a difficulty in integrating the software and hardware components.
- Other models that are commonly used for reliability estimation (by both hardware and software systems) are NHPP (Non Homogenous Poission Process) and Rayleigh distribution (Weibull with $\beta=2$).
- The zero-failure test model is also used for reliability estimation when no failures are detected during testing. The reliability is quoted with the level of confidence in the reliability metric. For example, if someone says we can say with 65% confidence that we will detect 3.5% errors in the product, this must be considered relative to the fact that tossing a coin is a 50% chance event
- The future holds ideas for John Deere like prototyping their equipment virtually so that the first machine/model that is built can be sold.
- The Mean time before failure (MTBF), Maximum likelihood errors (MLE), regression testing is also carried out on the products.

Appendix C

Company Name: Nortel Networks[4], NC

Date : April 22, 2003.

NC State Participants:

Dr.Laurie Williams
Nachiappan Nagappan
Mark Sherriff

Brief Outline:

The discussion with Nortel focused on current and future projects in which North Carolina State University (NCSU) and Nortel will collaborate. These projects involve a great deal of automated testing, both unit tests and acceptance tests. This information was critical in devising a way to leverage the information gathered and created during testing.

Points of the Meeting:

- Nortel is extremely interested in automating user acceptance tests to decrease testing time and increase the number of acceptance tests with no time penalty.
- Their projects revolved around technologies that could potentially make this more difficult: permissions, servers, email, installers, socket interfaces on same service on server.
- At NC State, we wanted to create additional automated tests to feed into the NCSU reliability study and to increase automation.
- Automated unit testing was already being done during development of new features. Nortel uses methods such as UI testing and JUnit testing for new classes.
- We all wanted to look at a way to create tractability from requirements to use cases to test cases.
- One strategy that was put forth was a full automation of "sanity" tests (30 or so most common operations). This would hopefully be under full automation – UI Workflows and scripting stuff / data-IO combinations
- Test-driven Development was discussed, but not seen as viable in the immediate future.
- Nortel currently uses a scripting method for automation/reliability testing. However, there was not way to test the scripting commands to ensure that they were running properly.
- We suggested using several tools that were becoming popular for testing in addition to the JUnit tool, such as SWTUnit (along with other xUnit tools), Abbot, and Marathon.
- Using the above testing tools would require developers to do additional work/different work, so this overhead should have a trade off by indicating to developers the extent to which they should carry out the testing.

[4] www.nortel.com