Producing More Reliable Software: Mature Software Engineering Process vs. State-of-the-Art Technology?

James C. Widmaier/Department of Defense
410-854-6951, widmaier@ncsc.mil
Dr. Carol Smidts/University of Maryland Reliability Engineering Program
301-405-7314, csmidts@Glue.umd.edu
Xin Huang
301-405-1071, xhuang@Glue.umd.edu

ABSTRACT:

Producing More Reliable Software: Mature Software Engineering Process vs. State-of-the-Art Technology?

A customer of high assurance software recently sponsored a software engineering experiment in which a real-time software system was developed concurrently by two popular software development methodologies. One company specialized in the state-of-the-practice waterfall method rated at a Capability Maturity Model Level 4. A second developer employed his mathematically based formal method with automatic code generation. As specified in separate contracts, C++ code plus development documentation and process and product metrics (errors) were to be delivered.Both companies were given identical functional specs and agreed to a generous and equal cost, schedule, and explicit functional reliability objectives. At conclusion of the experiment an independent third party determined through extensive statistical testing that neither methodology was able to meet the user's reliability objectives within cost and schedule constraints. The metrics collected revealed the strengths and weaknesses of each methodology and why they were not able to reach customer reliability objectives. This paper will explore the specification for the system under development, the two competing development processes, the products and metrics captured during development, the analysis tools and testing techniques by the third party, and the results of a reliability and process analysis.

ICSE 2000, Limerick, Ireland

KEYWORDS:
software reliability
Capability Maturity Model
formal methods
software engineering experiment
software process and product metrics

1. OBJECTIVE OF STUDY

A modest corporate sponsored software development activity was initiated to compare the functional reliability of two software applications built from the same requirements specification using different software engineering methodologies. One method was a classical "waterfall method" perfected to a high level of process maturity (Level 4) defined by Carnegie Mellon's Software Engineering Institute Capability Maturity Model[8]. The other methodology employed a state-of-the-art technology based upon formal methods with theorem proving and automatic code generation. Specware[4] developed by Kestrel Institute embodied this technology.

Each software development effort was managed under separate subcontracts to companies specializing in each methodology. There were a few simple ground rules in the experiment. Each company was given equivalent funding and schedule delivery requirements and well defined reliability objectives for two defined classes of failures. In addition they were allowed unlimited access to the customer to understand,/refine/correct the specification should faults be uncovered. (In actuality the customer embedded two faults in the spec to determine if the developers could discover them through their methodology). Process metrics on skill level, hours, and errors uncovered as a function of development stage were to be gathered. The final delivery was to include the source and executable C++ code for the system, documentation on the employed engineering process, design and development documents, software test plans, procedures, and test results.

An independent third party, the University of Maryland's Reliability Engineering Department, reviewed the customer's requirements, refined the model after all faults were removed from the spec, and developed a test

model of the specified system using Teradyne's Testmaster [1] They then executed a test script using Mercury's tool WinRunner [5]. (SPRE Inc. was hired to independently validate the test models developed by the University of Maryland). Each delivered software system was tested by the University of Maryland using the same test scenario. Failure data were gathered for each developed software application and then used to estimate the operational/functional reliability. Process metrics at discrete stages were gathered from the two software developers and were compared to help understand what led to differences in reliability.(see Figure 1 for the Experimental Process)

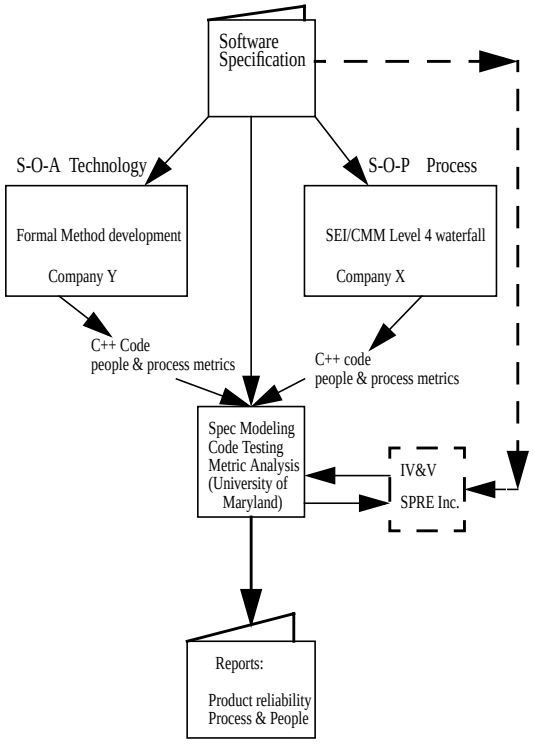
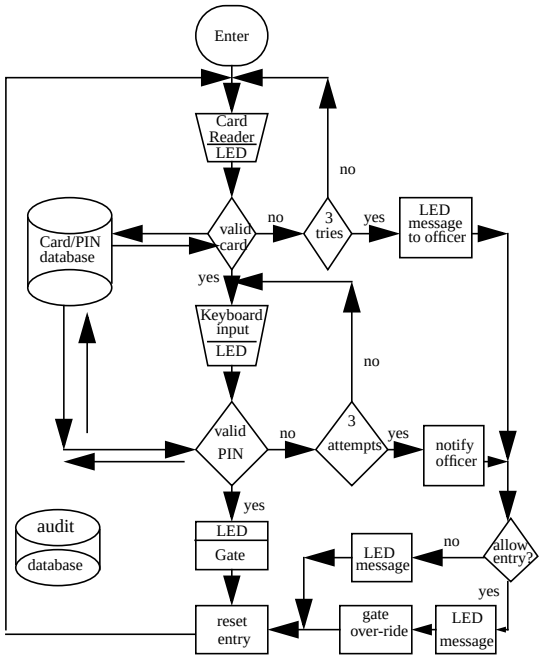


Figure 1: Experimental Process

2.DESCRIPTION OF SOFTWARE UNDER TEST

The software system specified for contracted development was a fictitious version of a personnel badge reader typically found at the entrance to restricted buildings. The Personnel Access Control System (PACS) is an automated entry access (gate) which reads a personal I.D. card containing an individual's name and social security number. The user swipes the card in the reader while the system searches for a match in the database, which may be periodically updated by system administration. If a match occurs, the system allows the user to enter his/her personal identification number (PIN), a four digit code, into a twelve position keyboard display. The system then validates invalidates the PIN and instructs/prevents entry through the gate.

A single linedisplay screen provides instructional messages to the user. An attending security officer monitors a duplicate message on his console with a gate entry over-ride capability. There are six simple hardware components to the PACS system - the card reader, keyboard, single line digital display unit, the guard display unit, guard reset unit, and the gate. Figure 2 summarizes the functional requirements as stated by the customer for the two software system developers.



* Card reader and keyboard share the single line LED display

Figure 2: Personnel Access Control System (PACS)

For simplicity the system was to consist of only one reader which operated twenty-four hours a day, seven days a week. Usage (i.e. system loading) was to vary from light to heavy depending on time of day and day of week. A Level 1 failure of the software was defined to be the condition or conditions in which the software was hung, or valid user cards and valid pins were not processed, invalid users had access, timing requirements were compromised, or the guard could not over-ride the system.

In summary, a Level 1 failure was defined to be one which brought the system to a critical state. The reliability target was specified to be 0.99 per transaction (i.e. on average 99

out of 100 typical gate transactions were to run successfully without Level 1 failures.)

Level 2 failures on the other hand, were defined to be less severe but manifested themselves as the system not working properly. The guard could override these malfunctions and still keep the system running. Anomalies such as an entrant carrying a large package who needs extra time were to constitute conditions for Level 2 system failure. Thus, a Level 2 failure was to have an operational work around. The target reliability was specified to be 0.9 per transaction.(i.e. 9 out of 10 transactions were to run successfully)

The specification also included messages for both the user and guard, auditing log requirements, database size constraints, keyboard timing and response requirements, and a few additional modest performance requirements.

A software solution was estimated by an independent source to be on the order of 800 to 1000 lines of C++ source code statements, excluding embedded comments.

3. RELIABILITY DEFINITION

Both software developers were given specific and identical targets for the reliability of their software as mentioned above. Reliability estimations were to be determined from testing the software using typical user input function frequency profiles (i.e. operational profiles). Operational profiles are defined as particular functions/operations and their frequency of usage [Musa99]. (In this experiment 88 operations comprehensively defined the compete functional profile for the specified system. However, only 35 were needed to cover 99% of all the user's expected operational profiles.) The reliability value in this project was defined as the probability of no failure for each gate transaction. Equivalently, it was the number of successful gate transactions divided by the total attempted gate transactions for the given operational profile.

In actuality, there are numerous ways to estimate the operational reliability taking into account runs which are partially successful or situations where faults are removed and the system operations are continued. However, in this experiment there was no requirement for demonstrating improvement in reliability (reliability growth) by the developers as a function of software evolution (calendar time). Thus, computing the reliability for this experiment by the developers should have equated to the simple success to success plus failure ratio. For example, if the program failed twice in one hundred separate system test runs, the reliability would have been 0.98 per transaction.

4. DEVELOPMENT ACTIVITIES

Conventional SEI Level 4 Approach

A development team which has reached Level 4 in the ascending scale of 1 to 5, is characterized as one which is focused on product and process quality. The team in this approach consisted of a program manager, a requirement spec analyst, a system analyst, a quality assurance and process engineer, one junior and three senior experienced software engineers. Their activities could be categorized into two groups-engineering and management. Engineering activities included the Requirements Management, Software Product Engineering, and Quantitative Process Management. Supporting these major activities were management related activities of: Project Planning, Project Tracking & Oversight, Peer Reviews, Integrated Software Management, Intergroup Coordination, Training, Quality Assurance, Quality Management, and Configuration Management. Each activity had a corresponding documented process or procedure. The design methodology of choice was to introduce object oriented methodology to generate the required C++ source code. The Unified Modeling Language (UML) and its computer based support tool Rational Rose [7] were employed for the first time to specify the system requirements and lead into development. Testing was addressed with only sixteen cases and they were run during system integration. A statement was made at delivery that the C++ "code was 100% reliable"; however no reliability demonstration tests were conducted.

Formal Methods Approach

The formal methods team consisted of a contract manager and two senior mathematicians/computer scientists. Aside from scoping the cost and schedule the two technical people were responsible for all development process activities.

Requirements analysis was a very intense and productive activity which produced multi-level state diagrams. The team opted to a prototyping stage which involved modeling and executing Haskell[2] specifications. Specware as required in the contract could have been used directly to express the hierarchal functionality of the PACS system in a language called SLANG and run as an executable specification. According to the programmer Haskell offered additional features with which to capture features of imperative programming languages as well as being "more user-friendly". The Haskell prototype was automatically converted to Specware specifications so that the automatic code generation feature of Specware could be exercised. The source code generated was in C++ and was compiled by GNU C++ compiler. The development team's integration test set-up was handwritten and only thirty test cases were run. A bug in the automatic code transformation led to a schedule slip before it could be resolved; consequently no theorem proving was done to verify specification correctness and completeness. As with the first contractor, this developer had no reliability demonstration test and assumed a "correct and complete implementation".

5. TESTING SCENARIO

During the subcontracted development activities, the University of Maryland prepared for the reliability assessment of the two operational coded solutions and anal-

ysis of developer process metrics. The first step in the assessment process was the construction of a test model using the tool TestMaster. An outside independent consultant (SPRE Inc.) was hired by the customer to review the TestMaster models constructed by the University of Maryland [3].

TestMaster is a relatively new test design tool based on the model reference test (MRT) technology. It uses the concept of Extended Finite State Machine (ESFM) to graphically and textually represent the specifications and describe the application's desired behavior. Software tests are created automatically using a script generator. The generator develops tests by finding a path through the specified system diagram from the starting to the exit state. The path will be a sequence of events and actions that traverse the diagram, defining an actual -use scenario. Each transition in the path may contain events, output actions, control information (predicates and constraints), variable assignments, and testing information. Once a path has been defined, the test generator creates a test script for that path by concatenating all of test action statements (in the language of the test execution environment) and data values required to move the system from its current state to the next state [1]. When the test executor applies this script to the system under test, the system should follow the sequence defined by the path if the system's implementation is correct. Figure 3 shows a high level view of the PACS TestMaster Model.

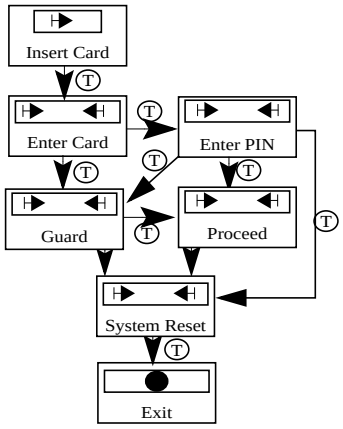


Figure 3: High Level TestMaster Model of PACS

To fully automate the entire testing process, the test generation tool had to be followed by a test execution tool. In this study, Mercury's tool WinRunner was chosen. WinRunner's test executor recognizes its own Test Script Language (TSL) which was used to construct TestMaster snippets. To calculate the reliability of PACS from the reliability of its functional and nonfunctional elements (performance), it is necessary to determine the probability with which a particular functional element will be executed. The probability can be determined by identifying the likelihood of different user software operations and the functions involved in these operations. (An operation being a logical system usage scenario which returns control to the system when complete and whose processing is different from other operations). An operational profile is simply the set of operations which users will employ and their probability of occurrence [Musa]. The PACS operational profile contained a theoretical 88 operations with frequencies ranging from 2.7E-10 to 0.38. A total of 35 out of the 88 operations contributed to a system reliability of 0.999, a value greater than the specified 0.99 requirement. Modeling 0.995 probability space would still have been sufficient. Testing proceeded along different operations by constraining the TestMaster model in each operation with an identical number of tests per operation. The testing approach could have considered a random selection of test cases; however, this would have precluded the in-depth investigation of the low probability operations and the study of behavior of the code in this context. In all 200 test were run on each of the 35 operations for a total of 7000 test cases.

6. RESULTS

C++ software was designed, developed, and delivered under the two methodology development subcontracts. However, what was delivered for product and process requirements failed to meet all programmatic expectations in many areas-from functionality to the lack of explicit reliability estimates to the required process metrics. Ironically, both developers claimed they had satisfied contractual requirements from both the process and software product standpoints. Process metrics concerned with manpower and skill levels were captured for all stages during development by both subcontractors. These process metrics are summarized in Table 1.

In the SEI/CMM waterfall methodology much care was taken for initial project preparation and estimation. The cost and schedule estimation predicted both early delivery and cost under-run. Project management was greatly involved at this and subsequent stages and as seen in Table 1 amounted to almost 50% of all man-hours. Requirements analysis, ironically, was minimized after having initially judged the customer's Requirements Spec to be "one of the best seen". However, lack of involvement of the design, development, and testing team may have been the reason for not finding the "two hidden faults" in the Requirements Spec. However, this team did tabulate defects found at each of their major development stages. As time progressed the initial CMM Level 4 (Quantitative Processes and Quality Management) contractor degraded to a "maturity level" somewhere between Level 1 and Level 2 (ad hoc/ repeatable respectively) when schedule and funding grew short. Technical heroes were conscripted to meet cost and schedule when the junior engineer failed to solve a seemingly easy problem with new Object Oriented tools. System level testing was sacrificed to meet final delivery deadlines; thus,

reliability based testing and consequential estimates were never made other than a statement that: “The software is 100% reliable”. When the University of Maryland ran 7000 user profile tests on this contractor’s 296 lines of code, reliability was determined to be 0.56 for Level 1 failures (on the average 44 of 100 test cases caused Level 1 failures). Processes and methodology were, however, well documented and re-traceable through most of the “non-panic” portion of the experimental development.

The formal methods process was surprisingly thorough in the requirements analysis stage as evidenced by the amount of time expended in this area (reference Table 1).

Summary of Analysis Results		
	SEI/CMM Level 4	**Formal Methods**
People	Company X 1 Entry Level Engineer 3 Senior S/W Engineer 1 Process Engineer 1 Q.A. Engineer 1 Program Manager	Company Y 1 Ph.D. Compute Scientist 1 Computer Scientist
Process Requirements System & S/W Design Implementation Integration & Test (Management) **Total**	SEI/CMM Level 4 83 hrs 211 47 36 385 **762**	Undocumented 178 hrs 52 283 36 8 **557**
Technology	* Object Oriented * Rational Rose * Unified Modeling Language	* Formal Specs * Specware
S/W RELIABILITY	0.56	0.77

Table 1 Summary of Analysis Results

It was in this stage that both the incomplete and conflicting requirements in the GFE specification were found. (One ground rule of the experiment was that the two subcontractors could not share information, such as shortcomings in the requirements provided by the customer. Thus, the SEI/CMM Level 4 subcontractor never had the benefit of resolving the shortcomings uncovered by the formalists). There was also continuity throughout the development from one stage to another since the two scientists “did it all”. A prototype was developed by the mathematicians in Haskell, a simple functional language with which they were intimately familiar. This intermediate stage was neither expected nor required, however, it was translated to the required Specware specifications, and then to C++ code. The C++ code generation was a major undertaking, fraught with technical incompleteness found within the Specware tool. The code generation stage consumed time and resources beyond what was expected. Theorem proving was consequently not done on the specifications because of escalating cost and schedule constraints. Unfortunately, this would have been an opportunity to uncover any possible specification faults or inconsistencies (of which there were two). Testing to enable reliability estimations was not conducted, which consequently, would have also led them to discovery of the two simple temporal errors. However, the 7000 operational profile tests by the University of Maryland on their 3986 lines of C++ code led to a reliability estimation of 0.77 (on the average 23 Level 1 failures within a 100 test case set). If the two errors found during the requirements review had been documented and removed, the reliability could have jumped to 98% instead of the measured 77 %!.The development process was poorly documented including the errors they found with the customer during a requirements review. Their engineering process was more of a research and development activity i.e. difficult to reproduce (a common short coming of most leading edge technologies).Table 1 summarizes the reliability estimates and performance metrics,

7. CONCLUSIONS

From the system reliability perspective, the formal approach produces better results than the classical software engineering approach; however, the customer’s system reliability requirements were not met by either approach. The reliability advantage realized by the formal methods occurred in the requirements stage where more than twice the man hours were devoted to finding requirements errors, inconsistencies, and incompleteness. The very nature of rigorous formal specifications and the executable prototyping capability were responsible for this advantage. Due to time constraints, the formal approach never had the opportunity to employ the theorem proving features of the tool during which additional shortcomings in the original specification may have been revealed. However, the formal approach lacked the overall process environment for good project management and a documented approach for re-engineering and/or maintenance. In addition a bug related to Specware’s auto code generator’s inability to represent nested logic, cost the developers precious time. The formal approach also suffers from a code optimization problem. While program performance never became an issue in this simple target system, more realistically sized real-time software system performance requirements may stress Specware’s ability to generate efficient code.

The classical approach fell short of the stated reliability objectives for a number of reasons as the process metrics revealed. Junior people being assigned to lead an apparently easy project got in technical trouble with new OO tools and technology. The primary weakness was that requirements analysis was not sufficient nor did it involve all “next-step” parties to detect inconsistencies or incompleteness. Consequently, the developed product reflected only what the coder interpreted to be the requirements. Inadequate scaling of the Level 4 process down to a small project may have also contributed to the degraded performance as witnessed by the heavy loading of management on the resource consumption. Documentation of the process

was very good and faults in the code could be easily corrected.

In conclusion, it might be apparent that a hybrid software development methodology be optimum for improved reliability by building on the strengths of requirements analysis by the formalists and the process management and documentation of the conventionalists. In any case, no claims about the reliability of software products should ever be made without having made deliberate testing attempts and reliability estimations.

References:

1. Apfelbaum L., Spec-based tests make sure telecom works, IEEE Spectrum, November 1997.

2. Haskell, The Haskell 98 Report, http://www.haskell.org/.

3. Huang, X., A Comparison Between Standard and Formal Mathematical Development, Master's Thesis, University of Maryland, Department of Nuclear Materials and Reliability Engineering, 1998.

4. Kestrel Institute, SPECWARE Users Guide, Version 2.01, Kestrel Institute, 1996.

5. Mercury, WinRunner User's Guide, Version 4.0, Mercury Interactive Corporation.

6. Musa J., Software Reliability Engineering: More Reliable Software, Faster Development and Testing, McGraw-Hill, New York, 1999.

7. Rose, Rational Rose 98 Edition, http://www.rational.com/rose/.

8. Software Engineering Institute, The Capability Maturity Model: Guidelines for Improving the Software Process, SEI Series in Software Engineering, Addison Wesley, 1995.