Continuous Evolutionary One-Step-Ahead Testing

Mechelle Gittens  
University of Waterloo  
mgittens@math.uwaterloo.ca

Hanan Lutfiyya  
University of Western Ontario  
hanan@csd.uwo.ca

Mike Bauer  
University of Western Ontario  
bauer@csd.uwo.ca

1. ABSTRACT
The traditional software development life cycle considers testing to be an activity that occurs between the implementation phase of development and software release [4]. With this approach any testing subsequent to release is done in reaction to failures reported by software users. The realities of software in operation however causes questions about this approach to arise. Adams [1] showed that organizations developing significant software applications often provide several fixes after their software has been released as the result of errors found in the field. This work also showed that the most serious and frequently recurring errors are usually found by users soon after a product has been released. These are referred to by Adams [1] as virulent errors. The negative effects of remaining defects implies that post-release activities should be proactive. These post-release activities must include continued testing by the vendor to find errors even after release. This paper proposes a solution to this requirement.

2. INTRODUCTION
Adams [1] recommends that preventive post-release approach be taken to mitigate virulent field errors. A continued testing process for this purpose has not yet currently been suggested in the research literature, however the Ziff Davis publication Baseline for corporate technology management [7], suggests that error-prevention middleware that tracks software activity and coding updates should be used by the customer organization to prevent the occurrence of errors. The article states that this middleware needs to monitor the interaction of the software with other applications and platforms for problems on a daily basis and suggests that a company monitoring its own software use, should monitor the use of the same software by other organizations so that they can prevent these errors in the local organization. This is a form of continuous testing, albeit distributing the testing to customers in the field, and allowing the use by some of the users to reveal the errors for others. Our proposal is that the vendor should act as a customer that uses the software in a variety of representative ways, and is therefore able to reveal errors for others. Some researchers have supported a continued process of collecting customer usage information to facilitate adequate pre-release usage-based testing [6, 11]. We believe that an approach that collects and uses customer information for continuous testing will both preempt error discovery in the field and enhance testing across versions of the software that represent continuous updates.

3. BACKGROUND
Since software testing after release would be focused on approximating usage in order to stay steps ahead of the user, it is necessary to employ a form of testing based on usage. We have chosen the operational profile model approach since it is recommended in [13, 12] as a suitable basis for testing that can properly benchmark customer usage. The traditional definition of the operational profile captures the set of operations for an application with their probabilities of occurrence[6]. Our research in [2] and the work in [11, 14, 8] motivated the need for an extension of the work done in [6]. As a result we have developed the Extended Operational Profile (EOP) model. This new model is more representative of observed software operational environments than the traditional operational profile model. The EOP model is composed of three parts: 1) The processes and the frequencies with which they occur in a typical customer application (process profile); 2) The structure of the platform on which the application is running, and characteristics of the dynamic data structures (structural profile); 3) The values of the data inputs to the application from one or more customer (data profile).

4. THE EVOLUTION OF TESTING
In this section we describe how the extended operational profile is used in an evolutionary process for any software development life cycle. This methodology is called the Continuous Evolutionary One-Step-Ahead (CEO) testing process, shown in Figure 1, since it undertakes a view of software development which recognizes that today’s products go through several versions that are upgraded continuously (an evolution) and are tested as such (continuously). The one-step ahead aspect follows from the proactive aspects that will be described as we go through the process. This enhanced life cycle modifies the traditional waterfall life cycle presented by Royce [9] with the extended operational profile model[1]. The CEO process provides an entire cycle that car-

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM Copyright 2004.

Copyright 2004 ACM X-XXXX-XX-XX/XX ...$5.00.
ries the user through iterations of a product and operational profile enhancement.

4.1 The CEO Process

In the Requirements Definition phase, there are three steps that are relevant to implementing the continuous testing process. Firstly, establishing at least one suitable measure of satisfactory testing (software quality factors) based on the company’s organizational goals. These may be based on measures such as the completeness of the testing in terms of code coverage [5]. Secondly, setting a goal using at least one of the measures deemed suitable at which the testing activity is considered to be satisfactory [6]. Define the aspects of expected customer use of the product, using the extended operational profile (EOP) model summarized in Section 3 to quantify the structural and data characteristics.

The Integration and System Testing phase may lead to increased knowledge relevant to the refinement of an extended operational profile. As a result, there will be feedback to the operational profile definition. A set of test cases is created that covers the expected customer usage according to the extended operational profile. These test cases are used for load testing, which attempts recreation of the product’s field environment, creating a mixture of test cases proportionate to the probabilities (profile values) of occurrence in processes for the customer being studied, which are run on representative data structures with data similar to that used in data structures.

On entering the Operation and Maintenance phase, which occurs at product deployment, information should be gathered about the value of the software measure chosen in the Requirements Definition phase (the software quality factors) [6]. Report on which set of tests were drawn from the test set in the representative test selection. This tracks the test cases that were used to record what has been tested and what has not (Step 8). The Vendor Stop Test Decision phase (Step 7) requires a check of the software quality factor. If the goal has not been met for satisfactory testing, if there are other tests remaining, continue the testing, since something may have been missed. Continue until the set quality goal has been met.

Within the Operation and Maintenance phase, in the Customer Monitoring phase (Step 10), more information about how the customer is instantiating and using the current release of the product is gathered in the field. This is done simultaneously with the continued in-house testing.

Subsequent to the on-site continued testing and customer defect report collection, if a vendor stop-test decision has not been made, documentation should be generated to record the Continued Testing Error Reporting (Step 9) which is done by the vendor on-site based on in-house test results, and Customer Error Reporting (Step 11) based on customer field use and problem reporting. These defect reports are helpful in triggering fixes. This trigger occurs when there is Combined Error Reporting, Step 19 shown in Figure 1, and translates through the steps starting with developing the necessary code correction (Step 14) through to the regression testing (Step 17) and back into the operational phase.

In the Customer Profile Data Parsing phase (Step 12), information about how a customer is using the product is then translated to operations, frequencies, structural information, and a summary of the data and hence an extended operational profile for the customer.

In the Customer Profile Refinement phase (Step 13), there is feedback from the recorded customer profile for the SUT, and the structural and data characteristics are compared to the extended operational profile for the test instance to discern how the customer information has: (i) changed from expected customer profile information, as exercised in the test instance; (ii) changed from previously recorded use of the product, as exercised in the test instance. If the processes, structures and/or data are significantly different from the expected or previously recorded processes, data or structures, the test instances should be updated, and
the profile of the typical system should also be augmented. The newly modified operational profiles are used, as with the original operational profile report, to produce new test cases. Any new test cases that exercise new behaviour should be added to the general test case repository.

The product fixes can be triggered from a Combined Error Report (Step 19). This means, from the reports generated by the customers and from on-site testing, as opposed to primarily from the customer and hence no major preemptive effort.

These steps from integration and system testing through combined error reporting and the cycle of Step 14 to 17 should be continued until there is a major release of the product. With a major release the entire software development life cycle would be typically followed from the requirements phase.

5. A CASE EXAMPLE AND RESULTS

We had the opportunity to work with the system testing team starting a new program of testing based on operational profiles for a commercial application. The process outlined here is based on observed practice in the case example from the requirements definition phase through the design, implementation, integration and testing phases. The derivation of the extended operational profile depicted through customer profile data parsing and customer profile refinement, are currently implemented. The vendor stop test decision step, continued system testing in-house, the creation of a combined error report, and the preemptive feedback into the build-integrate, have all fostered interest from our industrial partner, since this is partially done in a reactive manner.

As an item of future work, we intend to incorporate the latter phases into a controlled industrial experiment where we conduct a study of a release of the SUT prior to following the process, follow the entire process for a subsequent release, and measure any changes in in-house and field defects to evaluate improvements in defect occurrence that may result.

To conduct the customer monitoring shown in Step 10, we went to a customer site that made heavy use of the software. We collected the log files containing information on the configuration of dynamic data structures, a history of commands performed by the application, the data used with the application, and the hardware and software platforms at the customer site.

6. THE EMPIRICAL EVIDENCE

There is evidence that this approach which utilizes the collection of customer usage information is useful. We conducted a post-mortem analysis of the collected information about customer use to determine if there were defects that could have been prevented using the CEO approach. Our resulting observations can be categorized in terms of each of the parts of the EOP. As an example of the usefulness of collecting customer information for testing we extracted the probabilities (profile values) for the operations from a log file recording the use of the functionality of the application. This was done for the test instance and the customer instances of the application. A number of line graphs were generated illustrating the relationship between the occurrence probabilities for each of the operations in each mode as they occurred for the customer, along with the occurrence probabilities for each of these operations as they occurred in testing. Figure 2 shows one of the graphs of the results of plotting the process profile values for each operation within the administration mode (administration functionality), as it occurs in the customer instance, along with its process profile values in the test instance. The process profile values or overall system occurrence probabilities correspond to the y-axis, and the operations are enumerated on the x-axis. The figure shows a visible difference in profile values between the test system operational profile, and the customer system operational profile for operations enumerated in the range 70 to 80 and 85 to 95. There were a number of customer reported defects associated with each of the operations falling within these ranges.

More specifically, there are a number of defects that result from errors that can be categorized under process, structure, and data. These defects were reported by customers for later versions of the software and many of them can be related to the differences between the customer instances and test instances in the use of operations, the structural characteristics and data. We believe that exercising the test instance with knowledge of how customers use the software application may have preempted some of these defects. There are a multitude of defects that fall into the categories of process, structure, and data. To succinctly illustrate this point, Table 1 is presented, and it gives the total number of defects that can be placed into the categories of process, structure and data for characteristics where there was difference between customer use and testing.

7. SUMMARY AND CONCLUSIONS

We propose that conventional testing approaches where the activity is done only in the maintenance phase as a reactive measure should be modified to take a proactive stance by attempting to anticipate customer behaviour. Table 1 shows that there are errors attributed to customer use where differences between test activities and field activities have been recorded but not taken into consideration in tests. Table 1 adds to the findings of [1], and highlights two major points. These are (1) that software release occurred before testing that was representative of customer activities in the operational areas of process, data and structure could be done; and (2) that the current reactive approach will not remedy the problems experienced in at least 830 cases.

The summary of defects given here implies that an approach that captures customer use of an application and incorporates it into post release testing at the vendor site could mitigate some recorded defects. This defect analysis is a post mortem analysis, and this approach provides a connection between defects observed in usage of a release of the software subsequent to the release that we studied, and the components of the extended operational profile model, and the stages of the CEO approach. This study is considered adequate because it discovers that there are areas of use that are not generally captured by the approach to testing using the studied test instance of the software. It is likely that at least some of these defects could have been found with additional knowledge of realistic use. An alternative approach, which was not feasible due to industrial schedules, would have been to use the extended operational profile measurement derived in this work to derive test cases and execute them in the industrial test organization. This is future work.

There are challenges in conducting industrial experiments due to issues of schedules and the associated expense relating to dedicating person hours to validating a process such as the

\[2\] This subcycle is denoted in Figure 1 with the grey lines.
CEO approach [10]. We would therefore suggest mitigating such concerns by first using a more controlled and less costly environment such as a student environment to test and refine this approach [10]. However due to concerns for realism, an environment that is already a production environment in the university context is suggested. For example, in most Computer Science departments, support environments are created by teaching assistants for students in a course to work on and submit assignments. We suggest that such an environment is a production environment and the students in the course are the users. The faculty member instructing the course is the customer and the teaching assistants are the development staff responsible for the software development life cycle. Additionally the course should be repeatedly offered course support environment reused. This will create a controlled experimental environment that can be monitored, operational profiles can be derived and refined, changes can be integrated into the system and defects can be recorded. Using this approach experimental difficulties can be noted and improved for the larger industrial context.

8. REFERENCES