START Selected Topics in Assurance Related Technologies

Volume 5, Number 3

Applying Software Reliability Engineering (SRE) to Build Reliable Software

Introduction

The topic of the guest speaker, Jerry Weinberg, at ISSRE'97 (International Symposium on Software Reliability Engineering) was "Why don't we do what we know how to do?" He went on to explain how the knowledge, the technology, and the methods for building reliable software are known, and have been known for some time - yet few organizations are reporting successful implementation of software reliability programs. He asked the audience, consisting of reliability experts from academia, government, and industry, for their thoughts. Responses included:

- We are too busy fighting alligators - we don't have time to drain the pond.
- It requires "buy-in" from management and from senior staff in order to implement.
- The business process is not aligned with the reliability process.
- Although the knowledge exists, it is not necessarily common or well known or understood by the practitioners.

It became obvious as the discussion continued that "people" issues are at the forefront of our problems in and slow movement along the path to "reliable" software. Each organization needs someone to "champion" and that person need not be the manager, or the department head. The effort need not be full scale but participants need to identify and sustain a common goal.

This START attempts to clarify the common goal, to identify the essential components of the software reliability engineering discipline which, if implemented, can guide an organization to developing more reliable software.

What is Software Reliability Engineering?

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [Lyu 96]. Software reliability is an attribute and key factor in software quality. It is also a system dependability concept. Software Reliability Engineering (SRE) is defined as the quantitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability [Lyu 96]. As such it encompasses all aspects of the software development process. SRE employs proven best practice to ensure that product reliability meets user needs, to speed products to market faster, reduce product cost, improve customer satisfaction, and increase tester and developer productivity.

The essential components of SRE are categorized as follows:

- Establish reliability goals
- Develop operational profile
- Plan and execute tests
- Use test results to drive decisions

These components are sequential. Although this START addresses these SRE elements in isolation the reader should note that in reality they are integrated within the software development process. Reliability goals are part of the requirement definition process. Development of the operational profile occurs in parallel with software design and coding. Testing for reliability is a testing approach which is included in the overall test plan.

Establish Reliability Goals

Reliability goals describe the customer's expectation of satisfactory performance of the software in terms that are meaningful to the customer. This description may be significantly different from the theoretical definition of reliability but the customer has no need for the theoretical definition. The customer must tell you the circumstances under which they will "trust" the system you build. For example, someone who purchases a fax machine wants assurance that 99 out of every 100 faxes received will print properly. The fact that the machine must run for 25 hours without failure in order to demonstrate the specified degree of reliability is irrelevant to the customer. In order to test for reliability we often need to translate expressions that are meaningful to the customer into equivalent time units, such as execution time, but the goal remains as the customer perceives it.

What is important is that these customer needs, or expectations, are described in a quantifiable manner using the customer's terminology. They do not have to be statements of probability in order to be useful for determining product reliability. Some examples of quantified reliability goals are:

- The system will be considered sufficiently reliable if 10 (or less) errors result from 10,000 transactions.
- The customer can tolerate no more than one class 2 operational failure per release and no class 1 failures per release for a software maintenance effort.

All participants in the development (or maintenance) process need to be aware of these reliability goals and their prioritization by the customer. If possible this awareness should come from direct contact with the customer during the requirements gathering phase. This helps to cement the team around common goals. Ideally, reliability goals should be determined up front before design begins; however defining them at any point in the life cycle is better than not having them.

Define Operational Profiles

The operational profile characterizes system (product) usage. Use of this profile is what distinguishes SRE from traditional software development. In order to make a good reliability prediction we must be able to test the product as if it were in the field. Consequently we must define a profile that mirrors field use and then use it to drive testing, in contrast to testing which is driven by the design architecture. The operational profile differs from a traditional functional description in that the elements in the profile are quantified by assignment of a probability of occurrence, and in some cases a criticality factor. Development and test resources are allocated to functions based on these probabilities.

Use of the operational profile as a guide for system testing ensures that if testing is terminated, and the software is shipped because of imperative schedule constraints, the most-used (or most critical) operations will have received the most testing and the reliability will be maximized for the given conditions. It facilitates finding earliest the faults that have the biggest impact on reliability.

The cost of developing an operational profile varies but is non-linear with respect to product size. Even simple and approximate operational profiles have been shown to be beneficial. A single product may entail developing several operational profiles depending on the varying modes of operation it contains and the criticality of some operations. Critical operations are designated for increased or accelerated testing.

Plan and Execute Tests

Under SRE the operational profile drives test planning for reliability. Probability and critical factors are used to allocate test cases to ensure that the testing exercises the most important or most frequently used functions first and in proportion to their significance to the system.

Testing associated with development and maintenance of a software product fits into two broad categories: structure testing and usage testing. Structure testing is based on the software design while usage testing reflects how the software is actually used.
Unit testing, regression testing, and system testing are typically performed by the development team to the extent necessary to verify that the software has been implemented in accordance with the design. Attention to quality in this task facilitates the usage testing that is done later for purposes of determining reliability. SRE focuses on usage testing and often refers to it as reliability growth testing because the purpose is to determine how reliable the software is becoming.

Reliability growth testing is coupled with the removal of faults and is typically implemented when the software is fully developed and in the system test phase. Failures identified in testing are referred to the developers for repair. A new version of the software is built and another test iteration occurs. The testing scenario includes feature, load, and regression tests. Failure intensity (failures per natural or time unit) is tracked in order to guide the test process, and to determine feasibility of release of the software.

The testing procedure executes the test cases in random order - but because of the allocation of test cases based on usage probability, test cases associated with events of greatest importance to the customer are likely to occur more often within the random selection.

Reliability growth testing presumes that a test scenario will be performed more than once, and that deviations from the requirements are identified as failures. Each failure is recorded along with information that identifies where in the test cycle the failure occurred. Failures should be further classified by severity, and by functional area, and by life cycle phase in which they should have been discovered.

In some cases it is more meaningful to track the time to failure, or the inter-fail times than to track total number of failures.

The resources available, your philosophy on testing, and the criticality of meeting the reliability goals determine the amount of testing needed. Some experts feel that you can build reliability into your software with a strong testing program while others feel that the amount of testing needed can be reduced significantly if attention is given up front to ensure complete requirements specification, and that requirements actually drive the design. (We have all experienced situations in which the software is built and then the requirements are defined). Careful review of requirements and subsequent review of design with respect to requirements is tedious but typically finds the most important defects early, allowing corrective action before coding. This process is much more cost effective than waiting for testing to uncover all such errors. Structural testing only tests the software built - it cannot tell you what was not built that should have been.

Use Test Results to Drive Decisions

A normalized reliability growth curve can be used to predict when the software will attain the desired reliability level. It can be used to determine when to stop testing. It can also be used to demonstrate the impact on reliability of a decision to deliver the software on an arbitrary (or early) date. Figure 1 illustrates a typical reliability curve. It plots failure intensity over the test interval.

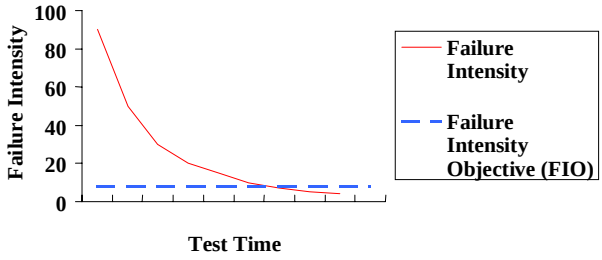


Figure 1: Typical Reliability Growth Curve for Software

The failure intensity figures are obtained from tracking failures during testing. Test time represents iterative tests (with test cases selected randomly based on the operational profile). It is assumed that following each test iteration, identified faults are fixed and a new version of the software is used for the next test iteration. Failure intensity drops and the curve approaches the pre-defined Failure Intensity Objective (reliability goal).

Figure 2 illustrates what may happen when the process for fixing detected errors is not under control, or a major shift in design has occurred as a result of failures detected.

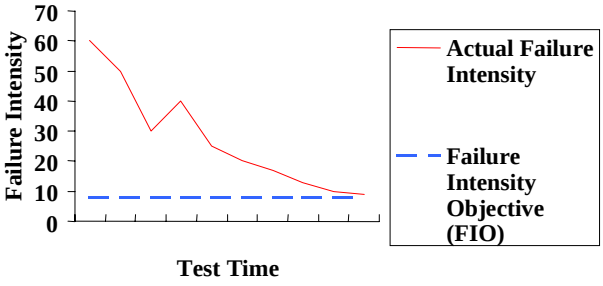
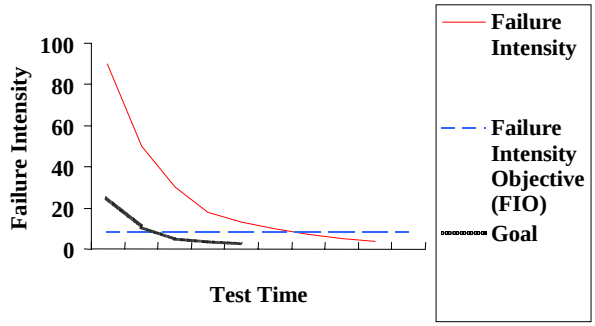


Figure 2: Reliability Growth Curve Reflecting Out-of-Control Process

Failure intensity drops, spikes, and then makes a gradual decline. Any predictions made prior to the 3rd test iteration would be grossly inaccurate because the spike could not be foreseen. The changes actually introduced new errors while attempting to fix the known errors. This graph actually identifies several potential problems. 1) Significant resources are being used for testing. 2) Testing is spanning a significant amount of time and may delay delivery, or be excessive relative to overall project resources. 3) The process for fixing errors may be inadequate. 4) There may be weak areas in the development process (analysis and design) itself which are the root cause of this erratic reliability profile. This type of graph is more likely to occur in efforts where the developer prefers to let the testing find the errors rather than design for defect prevention up front.

Figure 3 overlays the desired outcome (goal) from applying software reliability engineering principles on a typical reliability curve. The goal is to converge quickly to the desired reliability goal thus reducing the rework and the time and resources required for testing and enabling a shorter time to delivery without sacrificing reliability.

Figure 3: Reliability Growth Curve: Desired vs. Typical



This graph suggests that if the goal was achieved the software could be ready for release and testing could stop at least four versions (test iterations) earlier.

SRE is Cost Effective

AT&T combined the operational profile with other quality-improvement initiatives on a PBX switching system. This resulted in customer reported problems and maintenance costs being reduced by a factor of 10, system-test interval by a factor of 2, and product-introduction interval by 30 percent [Abra92]. Hewlett-Packard's application of SRE practices to their system-test process for multiprocessor operating systems resulted in a 50% reduction in the system-test time and cost.

SRE is a specialized subset of software quality assurance and as such is aligned with the high level programs for process improvement such as SEI's capability maturity model (CMM), and requirements for ISO 9000 certification. An organization which is already traveling one of these paths has a head start on SRE.

Conclusion

The key to success with SRE is to start simple. Stay focused on these essential elements, and add more sophistication only after you are comfortable implementing these basic principles. Remember the goal is to build and deliver software that your customer perceives as reliable. The reliability engineering practices described here are simply tools to facilitate your attainment of that goal. In order to obtain meaningful reliability

estimates you must be able to quantify and measure the software behavior with respect to the desired reliability objectives. You must test and testing should be driven by a usage profile. If there are not enough resources to test until the reliability goals are met you will at least know where you are and hopefully will be able to apply lessons learned to your next effort. This article provides only a cursory view of SRE noting the key factors of each topic.

For Further Study:

For further insights the following references are recommended.

1. Lyu, Michael R., Editor, Handbook of Software Reliability Engineering, IEEE Computer Society Press, McGraw-Hill , 1996.
2. Musa, John D., "Tutorial: More Reliable, Faster, Cheaper Testing Through Software Reliability Engineering", 8th International Symposium on Software Reliability Engineering (ISSRE'97), Albuquerque, NM, 1997.
3. Lakey, Peter B., and Neufelder, Ann Marie, System and Software Reliability Assurance, produced for Rome Laboratory, 1997.

Web Sites:

1. http://members.aol.com/JohnDMusa.
   (SRE advice by John D. Musa.)
2. http://www.faqs.org/faqs/software-eng/testing-faq.
   (Answers to questions on software testing.)
3. http://www.rstcorp.com/rst-web-top.html.
   (Organized lists of useful links).
4. http://www.mtsu.edu/~storm.
   (Links to software testing on-line resources).
5. http://www.soft.com/Institute/HotList/.
   (Organized lists of useful links).

About the Author:

Ellen Walker is a RAC specialist in software reliability. She holds Bachelor Degrees in Mathematics and Computer Science and a Masters Degree in Management. In a thirteen-year tenure as a computer scientist, she has worked with all phases of the software development cycle and supported both engineering services and business processes. She has been a facilitator and technical consultant for several long term quality initiatives, and is also a Reviewer for New York's Empire State Advantage (Quality Awards) program.

Other START Sheets Available:

94-1	ISO 9000
95-1	Plastic Encapsulated Microcircuits
95-2	Parts Management Plan
96-1	Creating Robust Designs
96-2	Impacts on Reliability of Recent Changes in DoD Acquisition Reform Policies
96-3	Reliability on the World Wide Web
97-1	Quality Function Deployment
97-2	Reliability Prediction
97-3	Reliability Design for Affordability
98-1	Information Analysis Centers
98-2	Cost as an Independent Variable

To order a free copy of one or all of these START sheets, contact the Reliability Analysis Center (RAC), 201 Mill Street, Rome, NY 13440-6916. Telephone: (800) 526-4802. Fax: (315) 337-9932. E-mail: rac@rome.iitri.com. These START sheets are also available on-line at http://www.rome.iitri.com/RAC/DATA/START in their entirety.

Future Issues:

RAC's Selected Topics in Assurance Related Technologies (START) are intended to get you started in knowledge of a particular subject of immediate interest in reliability, maintainability and quality. Some of the upcoming topics being considered are: Commercial Off-the-Shelf Equipment, Accelerated Testing, and Mechanical Reliability.

Please let us know if there are subjects you would like covered in future issues of START.

Contact Anthony Coppola at:
Telephone: (315) 339-7075
Fax: (315) 337-9932
E-mail: acoppola@iitri.org

or write to:
Reliability Analysis Center
201 Mill Street
Rome, NY 13440-6916

About the Reliability Analysis Center

The Reliability Analysis Center is a Department of Defense Information Analysis Center (IAC). RAC serves as a government and industry focal point for efforts to improve the reliability, maintainability and quality of manufactured components and systems. To this end, RAC collects, analyzes, archives in computerized databases, and publishes data concerning the quality and reliability of equipments and systems, as well as the microcircuit, discrete semiconductor, and electromechanical and mechanical components that comprise them. RAC also evaluates and publishes information on engineering techniques and methods. Information is distributed through data compilations, application guides, data products and programs on computer media, public and private training courses, and consulting services. Located in Rome, NY the Reliability Analysis Center is sponsored by the Defense Technical Information Center (DTIC). Since its inception in 1968, the RAC has been operated by IIT Research Institute (IITRI). Technical management of the RAC is provided by the U.S. Air Force's Research Laboratory (formerly Rome Laboratory).