Quality-Focused Software Testing in Critical Infrastructure

Zoë Oens, *Schweitzer Engineering Laboratories, Inc.*

zoe_oens@selinc.com

Abstract

When testing software, it is typical to hear that 100 percent coverage is unachievable. What happens, then, when we are asked to provide 100 percent? If the software being tested is used in critical infrastructure (e.g., power systems, water, medical, or banking), then an escaped bug is not a trivial thing. The quality of critical infrastructure software needs to have as close to 100 percent coverage as possible. This is not easy to achieve. It is even more difficult to achieve in a time-effective manner. This paper tackles the dilemma of building quality without exceeding a schedule and budget when it comes to critical infrastructure.

Biography

Zoë Oens graduated from Eastern Washington University in 2014 where she studied electrical engineering. She joined Schweitzer Engineering Laboratories, Inc. (SEL) that same year as a manufacturing test engineer. After a year and a half, she transferred to the research and development division as a firmware test engineer, where she has applied her knowledge of quality-focused testing into SEL's software development cycle. She has given many presentations and trainings on automation and quality throughout the company.

1 Introduction

Software produced for critical infrastructure is held to standards that ensure the safety of not only those who use the software but also those who are directly affected by the software being used. Software protecting the power system does not only affect the power company that uses it but also all people who receive electricity from that company. Development of such software requires the use of the best quality practices to avoid major malfunctions while continuing to deliver new features in a cost-effective and time-efficient manner. The iron triangle (or project management triangle) shown in Fig. 1 is a good way to visualize this problem. The triangle represents the tradeoffs made between scope, time, and cost and the effect they have on the quality of a product. Scope represents the work assigned to complete the project, time represents the timeline allotted to complete the project, and cost represents the project's budget (Schenkelberg n.d., 11–12).

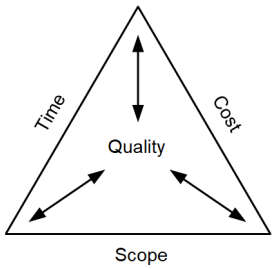

Fig. 1 The Iron Triangle

If quality is set as a static value, a set point that is to be achieved, then increasing one of the variables can decrease the others if needed. However, if the goal is to increase the overall test quality, separating the process into each area allows for a more focused approach to improvement. When focusing on the scope of testing, objective methods for determining test coverage can be put in place on a feature-by-feature basis. Reducing the amount of time spent not only running tests but also writing and reviewing test results can allow more time for exploratory testing. Building trust in testing and providing proof of its value helps to convince others that investing in testing is worthwhile. Focused incremental improvements in each area can improve the quality of the test process and, as a result, improve the quality of the software.

2 Scope: Ranking Functions by Criticality

For products used in critical infrastructure, removing the wrong tests or limiting testing on the wrong feature could result in a bug escape, which could lead to serious damage, injury, or even death. Therefore, it is important to determine the proper amount of testing needed to ensure a quality product without over testing. To determine this, there should be an objective way to analyze the testing investment for each feature released. For example, in network security, there is a method used to rank the amount of protection needed for a new device on the network. Predetermined inputs such as information value, device value, and vulnerability are used to determine how much effort is put into an asset's protection. This same method can be applied to determining the proper amount of testing that should be done on a new feature.

A common method for calculating such ranking is called the Kim and Kang method (Kellett 2016, 5). Using this method, a list of decision factors is created to encompass the important possible shortcomings of testing. This list is used across all features to make comparisons of test importance between features possible. Each decision factor is given a weighted value. The higher the weight, the more important the decision factor is to the software process. Features are evaluated one decision factor at a time, ranked according to each factor. The weighted values are multiplied by each rank and summed into a deterministic ranking that reflects the importance that should be considered when testing each feature.

For example, some decision factors for testing might be:

- Safety: “If this fails, what is the safety risk?”
- Reach: “If this fails, how many devices fail?”
- Usability: “How much would a bug from this feature affect the end user?”
- Complexity: “How complex is this feature?”

The decision factors are then assigned a weight:

- Safety: 1.0
- Reach: 0.8
- Usability: 0.7
- Complexity: 0.7

With all the setup done, a feature can now be evaluated. As a group, the ranking that is most appropriate for each decision factor is assigned. A weighted rank is calculated for each additional decision factor by multiplying the weight by the assigned rank, and then the weighted ranks are summed together, resulting in the feature risk (see Table I).

TABLE I
CALCULATING FEATURE RISK

Decision Factor	Weight	Assigned Rank (0–5)	Weighted Rank (Weight • assigned rank)
Safety	1.0	5	5
Reach	0.8	3	2.4
Usability	0.7	3	2.1
Complexity	0.7	1	0.7
Feature Risk			10.2

Feature risk is an objective value that is used to determine the amount of testing that needs to take place. There are a couple of ways this number can be used. A simple implementation involves predetermined tiered decisions (Wikoff 2006, 9). Test engineers must preset a testing intensity based on a threshold system. When a score is above a threshold, they can test to the stringency of that threshold. Another option is to use this number to objectively define the coverage that is needed to meet the test plan for this feature, using a percent of coverage as the final target.

Using this method or another objective ranking method helps to evaluate the priority of a feature. Having a consistent way to determine test scope ensures that the same decision-making process is used for all features. If issues with the calculation method itself occur, the weights or decision factors can be fine-tuned until they meet the needs of those using them. Since everyone on the testing team was involved in ranking the feature risk, there should be no uncertainty regarding the amount of testing required. For these reasons, the Kim and Kang method is a valuable tool for evaluating and properly limiting the scope of testing.

3 Time: Reducing Time of Critical Tests

A key component of being able to expand test coverage is reducing the amount of time spent developing, executing, and recording tests. The faster each test is completed, the more tests can be run. The problems faced in a manufacturing environment are similar. Therefore, there are a lot of lessons that can be learned from examining how extremely complex manufacturing processes are simplified and automated. Those lessons can then be applied in any aspect of software development to understand how best to improve processes. Using that knowledge, along with analyzing where in the process to focus efforts and knowing when it is best to implement automation, will ensure that the best improvement is made.

In manufacturing, a key part of any time-saving project is to properly understand the flow of the product through all parts of the factory. The assembly line receives parts, assembles them, and sends a product down the line to their customer. In the same way, software testing can be thought of as a single station on the assembly line. Code and specifications are received from suppliers upstream, and then test results are produced and shipped downstream to customers. Even in a rapid development environment where specifications, coding, and test writing are happening at the same time, it is helpful to understand which part of the process might cause a slowdown or stoppage in testing. Within the station itself, there are multiple tasks that can be improved individually to help the overall process.

Examining the test process from an external perspective can be very helpful in finding places for improvement (see Fig. 2). Creating good relationships with those who supply code and specifications can eliminate a lot of miscommunication and vague expectations. A design-for-test document can be created to communicate the coding practices to be followed that benefit testers and make automation easier. Creating consistent test results that are easy to evaluate is a good way to eliminate rework. In summary, when looking at testing from an external perspective, focus on communicating, trusting, and understanding group members to make huge strides in reducing test times.

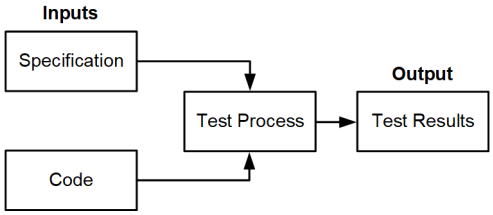

Fig. 2. External Test Process

First, the expectation of what a station on the line will receive from the supplier upstream and deliver to the customer downstream should be well-documented. This documentation should not cover the content of what is being delivered, but rather how the delivery should happen. Doing so allows for clear understanding of how best to approach reducing the time it takes to accomplish the task at the station. This can be accomplished for test suppliers by using a design-for-test document that outlines consistent software design and documentation practices. Similarly, creating template results that can be easily filled out and used consistently reduces the number of errors in test result creation, making the job of the reviewers easier.

When there is a problem at the station, it is important to document what caused the problem, how long it stopped work, and what the solution was (Schonberger 1986, 18–20). This allows the station to reduce downtime and keep work stoppages to a minimum. A frequently used term to describe this process is root cause analysis. It is important to share this analysis with both suppliers and customers. Sharing this information frequently allows others working on the project to eliminate work stoppages and avoid the same issues in the future. Using a whiteboard visible by the whole team to collect issues is a good way to share information with everyone in real time. Teams should discuss these issues together to come up with solutions to the most pressing matters.

When an issue arises that could put a stop to all work on the project, it is important for all parts of the assembly line to know, so that all who are affected can work to fix the issue to get everyone back up and running. Helping other stations on the assembly line continue to run smoothly helps keep the assembly line running as a seamless unit. Communicating directly with someone using a real-time communication method (e.g., face to face, via instant messenger, or over the phone) is a much faster resolution than logging a bug or sending an email (Schonberger 1986, 40).

Once there is consistent communication between stations, work can be done to reduce the time it takes to execute a test. Testing can be broken into four categories: test write, test setup, test execution, and results creation (see Fig. 3). Each area has unique challenges that provide opportunities for improvements. Clearly defining these different sections can be helpful when planning to improve the test execution overall. This paper uses the following definitions:

- **Write** is the act of taking a specification and defining repeatable test steps. This could be in the form of a test case specification, writing test code, or a manual test write.
- **Setup** is any required action that needs to occur before a test is executed. This could range from hardware changes to software installations.
- **Execution** is any action that results in the test passing or failing. When focusing on improvements in speed, this is usually where most of the work takes place, and it is the section most likely to be automated.
- **Results creation** is the documentation of the execution results.

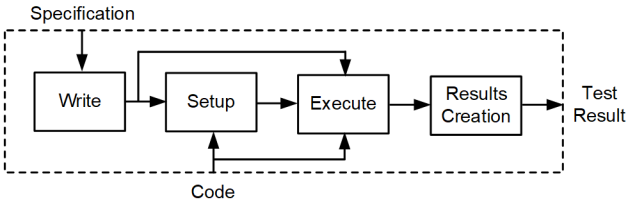

Fig. 3. Internal Test Process

3.1 Test Write

Test write can be thought of as a data conversion. Using the specification as input, the raw data outlining the operation of a feature can be transformed into a sequence of steps for testing that feature. Thinking this way makes it easier to find features that are similar and makes it possible to devise a way of automatically converting the specification into tests. Consider the following example. A testing team receives specifications for buttons on different webpages. There are hundreds of buttons and all of them must be tested the same way. The specifications read something like this: “Button 4 should be placed in the top left of the page. Its size should be 100x200 pixels. It should read ‘Okay,’ and when pressed, it should navigate to webpage.com.”

In this case, someone would be required to write a unique test for each button that is specified in this way. If the specification is machine-readable and consistent, though, then the conversion from specification to test could be automated, saving time in test write and allowing more time for execution. The test code needs to be implemented in a way that will handle all possible behaviors displayed by a button, regardless of its location, size, or function. The specification can then be passed to the test code directly. In this case of button testing, a test engineer could start by defining the required testing the button needs to undergo. Based on the previous specification, they could run tests on the location of the button, its size, the text on the button, and its function. If the specification is written in a data storage language like XML or JSON, then no conversion is needed. Otherwise, if all button specifications are structured and worded consistently, then a parser could be used to interpret existing specifications into such a language.

Either way, the goal is to have a file that can easily be converted to a structure in a given testing language. The test code then oversees interpreting the specification and executing the proper battery of predefined tests. Automating the data transfer requires open communication and cooperation with partners upstream who supply the specification. Once the test writes are automated, it is equally important to have the testing infrastructure ready so that the test can be started quickly.

3.2 Test Setup

Preparing a test environment capable of handling tests can be a daunting task. It can be thought of as similar to setting up an assembly line station, where having all the tools needed to complete the job of the station is crucial. Tools that are used most are positioned closer to the assembler, whereas tools that are used less often are positioned farther away. The positioning of tools is the same between stations. Similarly, if software testing requires hardware, it is helpful to follow the same rule and organize the equipment so that the most commonly used equipment is easily accessible. Program installation locations between machines may seem trivial, but when there is a problem, removing even the smallest variable can reduce complexity. If the same set of testing is run consistently enough, it may warrant a separate station to only run that one test. A well-developed setup creates a smooth transition into test execution.

3.3 Test Execution

Most of the focus of reducing test time goes into the test execution stage. Implementing automation for test execution is important in reducing test time. However, it is important to consider all the factors before investing time into automating a test. The most important factors to consider are the time it takes to run a test manually, the frequency of test runs, the frequency of test changes, and the time it takes to automate the test. It is also important to be consistent. The more people who work on automation, the more important it is to create consistency in how the scripts look and execute, the functions they use, and the layout of the results they may create. Creating this consistency will make maintenance of automation easier as time goes on and the amount of automation grows.

3.4 Test Results Creation

Results creation is often overlooked when it comes to reducing the time it takes to run a test. Receiving a request to make changes or discuss a test result and its formatting consumes time that could be used doing more testing. Having consistent test results and formatting the results in a way that the reviewers can easily understand are two main ways to reduce rework for results. When a test is automated, results creation can often be automated as well to help ensure the consistency. The automated results should still be formatted in a human-readable manner. The results should be easy to evaluate when all steps pass and easy to debug when a step fails.

As a final note, automation is extremely important when reducing the amount of time spent on testing. It is also an effective way to increase the consistency of testing across similar features and ensure coverage. However, in any step of the test execution that uses automation, it is important to be prepared for two of the many automation pitfalls. First, no automation system is perfect. It is always necessary to spend time keeping the system running. There will always be a new feature or a change that breaks the test automation. Second, it is important to train others to use and maintain any automation created. It is best to train everyone who is willing to learn. The more people who know how the automation works, the faster any future problems within the system can be fixed.

With all the focus on reducing time, it is important to not lose sight of the end goal of improving test coverage. The faster a test can be run, the more tests can be run in the time given. Finally, the more quality testing is done, the more confident a team can be that the product being released will be ready to be used, as if someone’s life depended on it.

4 Cost: Selling Quality

A lot of work is put into deciding how much testing should be done to balance cost, time, and scope to ensure that critical infrastructure software has zero defects. This is clear to testers; however, it can be difficult to convey this to other team members. It is critical for the entirety of the team to understand the importance of zero defects when it comes to software in critical infrastructure. Sometimes, that means helping them understand the cost of a failure. Often, a good way to help others understand how critical a failure could be is by showing them the estimated cost of a potential failure. This includes the cost of running a bug fix project; the cost of retesting; the cost to the end user to update software for a new release; and often, in critical infrastructure, the cost of someone being injured. These are all effective ways of convincing a team to be more quality focused.

The next step is to measure quality in an objective way. Tracking quality metrics, specifically focused on tests to determine the effects of any test decision, is important when conveying the results of the decision. When focusing on quality, it is important to be able to measure quality as the team is making strides to improve. Every team is different, and there is no universal answer for what metrics should be used to measure a testing group. However, there are a few that are worth noting here:

- **Change failure rate** measures how many failures occur after release that require immediate attention (Duvall 2018, 17). Tracking this number will allow the team to make changes and, after the fact, determine whether the change has an effect. The downside to this is that it can only be measured after the issues have already happened. Within tests, this process can be used to track false failures and to identify weak points of automation.
- **Business value delivered** can be preemptive and used in a pitch when suggesting new testing techniques or suggesting expanding coverage (Phillipy 2014, 3). It can also be used to put a price on a specific bug that was found or missed to explain the necessity of testing.
- **Customer satisfaction** involves solid channels of communication and giving opportunity for internal customer feedback. Gaining the trust of other team members and internal customers helps to strengthen the drive toward quality. This metric can be tracked through surveys or one-on-one communication with customers to determine what are perceived to be the pain points in testing.

These are simple metrics that are easy to implement and maintain, but the value of any applied metric should be consistently scrutinized to find what works best for each organization. As there are no two people who are the same, no two organizations are the same, and so the focus of improvement for each should be individualized.

5 Conclusion

When testing software in critical infrastructure, it is important to analyze all parts of the testing process to improve the overall quality, so understanding what drives quality is essential. Testers who strive for quality have the responsibility to increase the quality of testing while simultaneously reducing the variables that drive quality. Improving the selection of test priorities ensures that coverage goals are met while avoiding excessive testing. Using an objective method for determining priorities increases traceability and understanding of the necessity of testing. Understanding the flow of the test process and breaking that process down into its individual parts improve the overall process. Finally, convincing all involved in the software process to make quality the highest priority ensures that everyone on the testing team is on the same page about quality expectations. Exploring each of these options to improve quality is the best way to ensure testing does not exceed a schedule and budget in critical infrastructure.

References

Duvall, Paul. 2018. “Measuring DevOps Success with Four Key Metrics.” Stelligent. Accessed May 26, 2020. http://stelligent.com/2018/12/21/measuring-devops-success-with-four-key-metrics/.

Kellett, Matthew. 2016. “Kim and Kang’s Approach.” In “Ranking Assets Based on Criticality and Adversarial Interest,” 5–9. *Defence Research and Development Canada* 168.

Phillipy, Mark A. 2014. “Delivering Business Value: The Most Important Aspect of Project Management.” Proceedings of PMI Global Congress, Phoenix, AZ.

Schenkelberg, Fred. n.d. “Introduction to the Quality Triangle.” Accendo Reliability. Accessed May 21, 2020. http://accendoreliability.com/introduction-quality-triangle/.

Schonberger, Richard J. 1986. *World Class Manufacturing: The Lessons of Simplicity Applied*. New York: Free Press.

Wikoff, Darrin. 2006. “Managing Assets by Criticality.” Plant Services. Accessed May 26, 2020. http://www.plantservices.com/articles/2006/125/?stage=Live.

20200908 • TP6995-01