
Quality Improvement Through Test Automation: A Proposal

Author: Julia Keffer

Date: November 18, 2012



Table of Contents

Executive Summary	1
Introduction	3
Problem	3
Causes	5
Solution	6
Implementation	8
Benefits	8
Cost	8
Alternatives	10
Continue with Only Manual Testing	10
Use a Third-Party Framework	10
Implement Other Quality Improvement Measures	10
Conclusion and Recommendations	10
Appendix	11
Survey - Customer Support	11
References	11

Executive Summary

The purpose of this report is to examine the increasing number of customer complaints related to product quality and to recommend a solution.

Customer defect reports in our flagship SDK product have increased over the past three releases. Customers are becoming frustrated in their efforts to adopt our product. The result is delays in their time to market and in our time to realize revenue. Some customers have abandoned their efforts altogether.

Defects occur when:

- new functionality does not work as intended
- existing functionality breaks when detected defects are fixed

Defects vary in severity; however, undetected defects have increased by 15% over each of the last three releases. We need to stop the trend now.

This proposal addresses defects that escape due to missed failure paths during developer testing and defects that escape because we run out of time to retest defect fixes during product verification.

We must improve the situation using existing staff and minimize the impact to release schedules. The solution must be effective over the long term.

I propose that we build a test framework to enable us to automate interface tests. This functional area accounts for 40% of undetected defects. The tests are high value, well-defined, low maintenance tests.

Testers can implement a test framework and automate a selection of existing tests as part of the upcoming release. To provide an ongoing benefit, software developers will write new tests as part of developer testing for this and all future releases. If we automate 50% of existing tests and write tests for all new functionality, after three releases, the tests will cover 75% of interface functionality.

The automated tests will run as part of the nightly software build, which means that defects related to changes in the interface code are detected and fixed as early as possible, before they escape to product verification or to the field.

The benefits of this approach are:

- Increased customer satisfaction shown by fewer support calls
 - Faster time to market for the customers and we will realize revenue sooner
 - Increased discipline applied to developer testing
 - Increased morale in both the software development and product verification teams
-
-

The budget already covers staffing costs. The upcoming product release schedule will increase by 4 weeks. There is no impact to future release schedules.

If we do nothing, the situation will get worse. Adopting a third-party framework instead of building our own will not save time, costs more, and is less flexible. Other quality improvement measures will take longer to show benefits and the benefits will not accumulate. We can still consider other quality improvement ideas at a later time.

The solution I propose will show the greatest improvement, in the least time, with the lowest cost, and the benefits will accumulate over time.



Introduction



The purpose of this report is to examine the increasing number of customer complaints related to product quality in our flagship SDK product. In this report, I will describe the problem, the impact, and two of the causes. I will discuss the proposed solution, the implementation, and the costs, followed by the conclusion.

Problem

Customer defect reports in our flagship SDK product, which accounts for the majority of our revenue, have increased over the past three releases. Customers are becoming increasingly frustrated in their efforts to upgrade to new releases and develop new applications. The result is a delay in their time to market which causes an increase in the time before we realize significant revenue from a sale. Customers who are unsuccessful during the evaluation process may abandon efforts to use our product, costing us potential sales. This problem is also affecting customer support staff. My interviews with the support team indicate that the time they spend investigating problem reports has increased over this same time period such that it now consumes an average of 50% of their time.



When we implement new features or fix defects, new code can introduce defects in two ways:

- new functionality does not work as intended
- new code breaks existing functionality



The industry typically measures the number of defects in relation to the number of lines of code because a larger code base has more defects. Defects are measured per thousand lines of code (kLOC). The industry standard is somewhere between 6 and 30 defects per kLOC, depending on the complexity of the code. However, it is harder to quantify complexity. To understand this, consider as an example, the SIP functionality we introduced in the last release. We needed to rework the software architecture of 40% of our code, which increased the complexity. It also introduced 50,000 (50 kLOC) new lines of code.

I analyzed the data in our defect tracking database and discovered that in our code base, the number of defects per kLOC raised in the field has risen by 15% over each previous release. The chart below shows the trend.

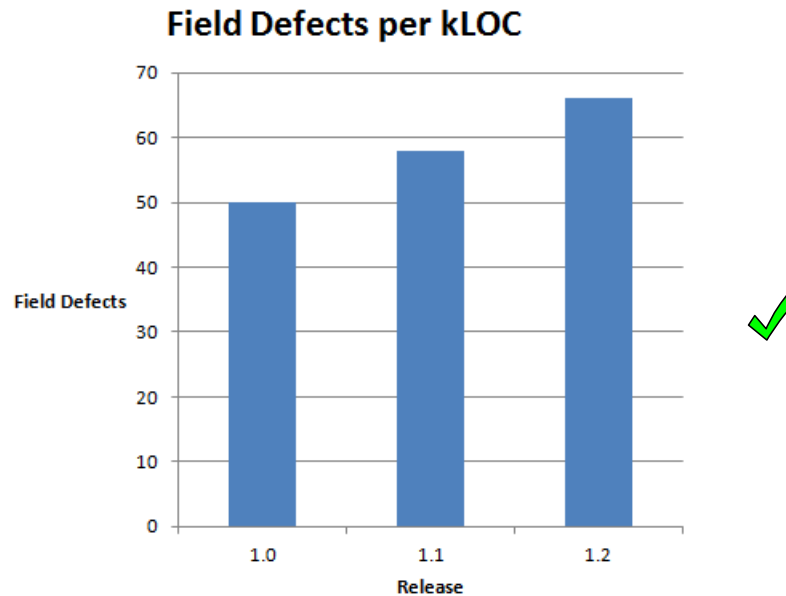


Figure 1: Field Defects

Consider the following when you interpret the numbers:

- Defect reports from the field are a fraction of the total number of defects present.
- All defects do not have the same impact. For example, a misprint in a log does not have the same impact as a crash.
- Customers typically find the highest impact defects.

The key concern is the increase in the number of defects per kLOC. We do not want the trend to continue.

Another concern is the number of patches we issued to fix defects in the field that blocked customer development. We released ten patches for the last release compared to six for the previous release. Each patch released to the field provides an additional opportunity to introduce undetected new defects because patches do not undergo formal product verification. Waiting for patches is also a source of customer dissatisfaction.

Finally, the number of defects found in the product verification phase has also increased. Fixing each defect introduces another opportunity to create new defects, which also may go undetected before the release goes to the field. The chart below illustrates the trend.

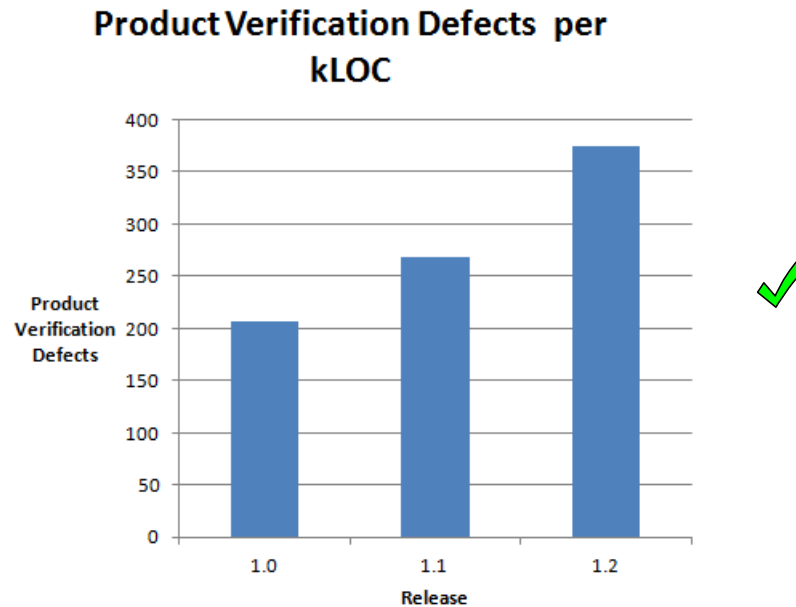


Figure 2: Product Verification Defects

Causes

To proceed with a solution, we need to understand why defects exist. There are a number of reasons; however, there are two causes the proposed solution will address:

- Software developers approach testing differently from product testers. Software developers focus on demonstrating that the software works, whereas the mindset of a tester is to break the software. The developers' mindset can lead to overlooked failure paths during developer unit testing.
- The more defects testers find during product verification, the more time they spend retesting the defect fixes, which takes time away from testing other scenarios. We typically allocate a fixed amount of time for the product verification phase which can result in untested defect fixes. We must make the best possible use of the available time.

The goal is to reduce the number of defects that escape from one phase of the software development process to the next. That is, reduce the number of defects that escape from developer testing through to product verification, and reduce the number of defects that escape to the field.

We need to keep in mind the following constraints:

- There is no money in the budget to hire new staff, therefore, any solution must use only existing staff.
- We need to minimize the impact on product release schedules, now and in the future.



Solution

In addition to meeting the constraints above, a successful solution must address the situation over the longer term.

I propose that we introduce test automation as a solution to reduce the problems described above. The chart below shows my analysis of the defect categories in the defect tracking database. Defects that occur in parameter checking and functional scenarios, which we refer to as the SDK interface, represent 40% of the total defects found during product verification and in the field.

Defect Breakdown

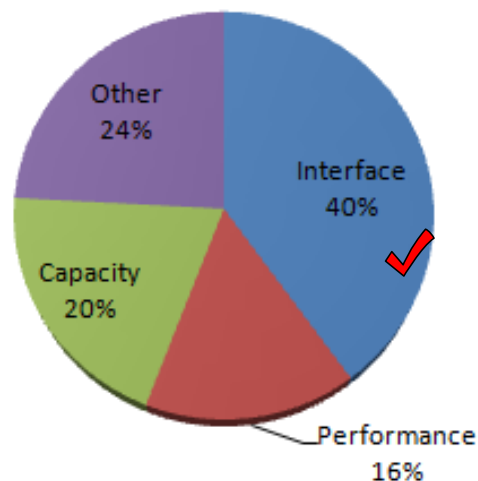


Figure 3: Defect Breakdown

Tests that cover this particular category of defects lend themselves well to automation. The tests have well-defined boundaries and unambiguous pass/fail criteria. They represent actual customer use cases. Our interface must remain backward compatible between releases; therefore, the maintenance costs for these tests should be low.

Because the tests are automated, it is easy to determine the scenarios that the tests cover. Product testers can review the tests, suggest additional scenarios to add to developer testing, and

maximize the use of their testing time by focusing on tests that are not suitable for automation. Designers can learn from the feedback they receive from the testers.

I propose that we implement our own test framework to enable us to customize it for our product. Testers will develop the framework and automate a selection of existing tests as part of the test preparation for the next release. This covers tests they would normally run manually to ensure that existing functionality still works. Software developers will write new tests as part of developer testing for the next and all future releases.

We will run the automated tests nightly, as part of the overnight software build. The diagram below illustrates how test automation works.

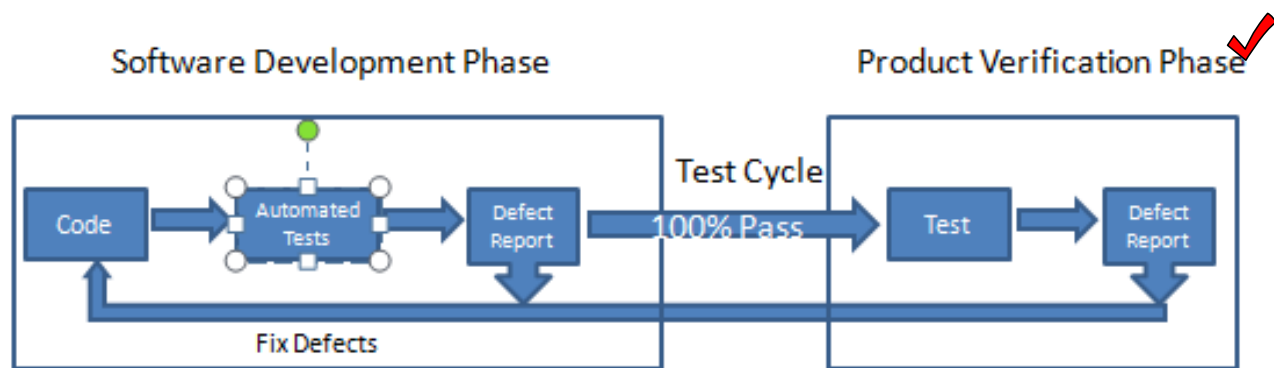


Figure 4: Software Release Cycle

Each day, developers submit new code. The automated tests run as part of the nightly software build and produce a report of test failures which indicate defects introduced by the previous day's code. Developers fix the defects and when the automated tests run again, the results show if the defect is fixed and if the changes introduced new defects. This process occurs during all phases of product development. The nightly test runs catch defects introduced during the development phase and ensures that the fixes for defects found during product verification do not break other functionality.

The diagram above also applies to patches, except that patches are released to the field when all tests pass, instead of moving to the product verification phase.

The strategy works in two ways:

- It reduces the number of defects that escape from software development through to product verification which reduces the chance of undetected defects escaping to the field.
- Running the test suite on patches released to the field provides an opportunity to detect defects introduced in the patch.

We can determine the strategy's success by analyzing the defects in the defect tracking database. We should see fewer defect reports related to the interface from both the field and in product

verification. The strategy meets the solution criteria, because the test suite grows over time, and because the nightly test runs continually detect when new code breaks existing interface-related functionality.

Implementation

The work required to introduce automation includes:

- Design and write software for a new test framework. (8 person weeks)
- Automate existing tests in areas we plan to change in the next release. (6 person weeks)
- Integrate tests into nightly software builds. (2 person weeks)

Jane Smith, our senior tester, will write and design the framework. Two other testers, Steve Jones and Dave Johnson, will automate a selection of existing tests. Jamie Brown, our configuration management specialist, will integrate the tests into the nightly software builds.

The set of existing tests I propose to automate covers approximately 50% of our interface functions. We project that the number of interface functions will expand by 25% as we implement new features over the next three releases. If we add tests as we introduce new functionality and add tests as we fix defects in the interface that are not covered by existing tests, after three releases, the test suite will cover 75% of the functionality.

Benefits

- Increased customer satisfaction shown by fewer support calls and complaints to sales staff.
- Reduced risk of customers giving up on our product because they cannot develop their solution.
- Increased discipline applied to analysis of the software under test and test planning.
- As we automate more tests, the benefits of automation accumulate over time.
- Increased morale in the QA department because it eliminates monotonous defect retesting.
- Increased morale in the software development department because they will spend more time writing new code instead of fixing defects from the field, which is less interesting.
- A framework provides future opportunities to further improve quality by automating additional tests.

Cost

The solution does not impact the budget because it already includes money to cover staffing costs. Implementing the solution will extend the upcoming project schedule by four weeks; however, this is a one-time cost.

The current product release schedule is forecast to require 26 weeks. During the first 13 weeks, software developers write the feature code and perform developer testing. Concurrently, the

✓ testers perform verification preparation activities. The product verification phase starts in week 14.

Jane will spend 8 out of 13 weeks in the preparation phase writing the framework, leaving her with only 5 weeks for test preparation. To minimize the impact on her testing activities, I propose that Jane continues to work on preparation activities after the product verification phase begins. A software developer can assist her by running the tests she writes. This contains the impact on the overall schedule to 4 weeks.

Steve and Dave will each spend 3 out of 13 weeks writing automated tests. They will use time that they would otherwise spend reporting and retesting defects on the interface later in the test cycle. The extra 4 weeks already added to the schedule will cover any additional effort.

✓
Refer to the chart below for more information.

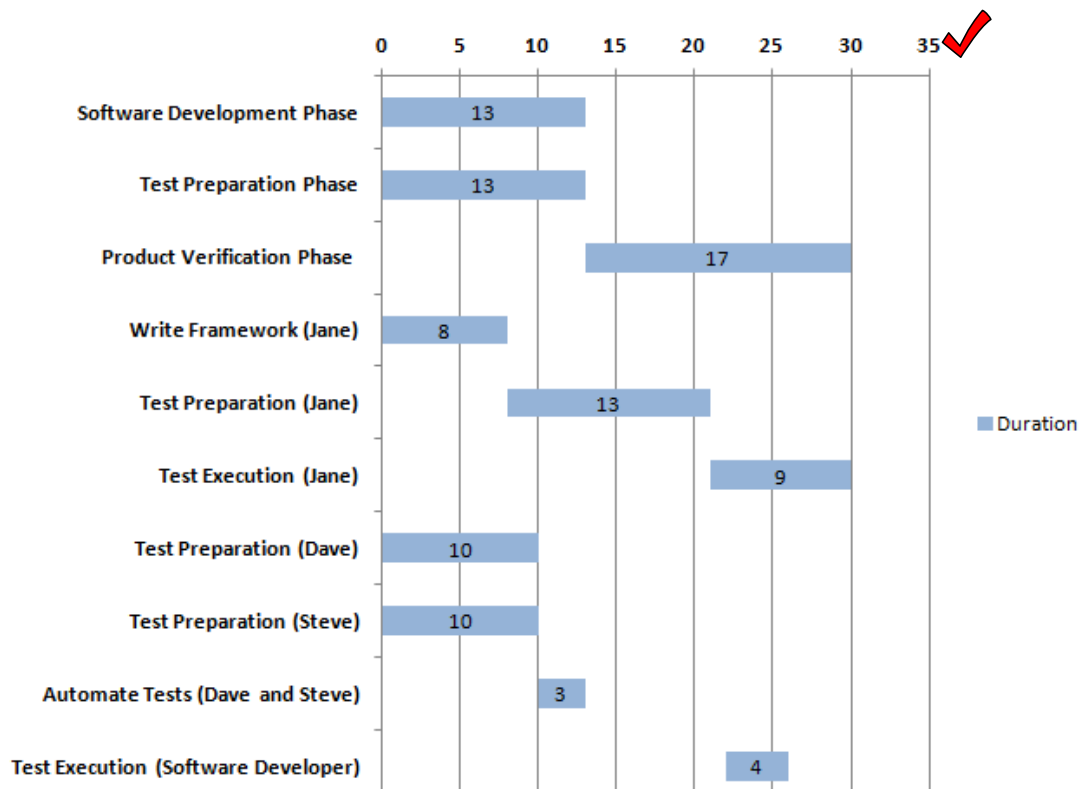



Figure 5: Proposed Product Schedule

The net effect on the software development portion of the schedule will be minimal. Automating developer tests may take slightly longer in this release as the developers become familiar with the test framework. However, I expect that the overall time spent writing automated tests will be approximately equal to the time spent running the tests manually. ✓

Alternatives

- Continue with only manual testing.
- Buy a third-party test framework.
- Implement other quality improvements.

Continue with Only Manual Testing

If we continue with our present course of action, the current trends will continue. Customers will continue to experience the effects of poor quality, with the effects described above. Recall the case of ABC Company. They abandoned their efforts after a month spent trying to get their solution working. The projected yearly revenue from this company was \$100, 000, representing 10% of our current revenue. 

Use a Third-Party Framework

I recommend that we do not use a third-party framework for the following reasons.

- It may not save time because we need to evaluate different products.
- We do not have money in the budget to purchase a commercial product.
- If we use an open source framework, we need to choose one that is actively maintained. Otherwise, we need to maintain the code ourselves, which means we are no further ahead than if we build our own framework.
- A third-party framework is unlikely to meet our exact requirements. We need time to either modify open source code or to work around limitations in a commercial tool.
- Third-party tools upgrade on a schedule that may not align with ours. A custom framework gives us flexibility to make changes only when we need them.
- Developing our own framework allows us to customize the solution to our product.

Implement Other Quality Improvement Measures

There are other quality improvement measures we could implement; however, I believe test automation will show the highest immediate and ongoing benefit with the least schedule impact. Other measures, such as code inspections, may also help. However, the benefit will not accumulate over time, unlike the accumulated affect of continually adding to our automated test suite. Additionally, we can still implement other measures in the future.

Conclusion and Recommendations

We need to take action to improve product quality and customer satisfaction. I propose that we implement test automation with the goal of reducing the number of undetected defects that escape through to product verification and into the field. I recommend that we develop the test framework in-house. My proposal does not require any new staff and the impact to the schedule of the upcoming release is limited to four weeks. I do not expect any impact on future development schedules. The benefits of this proposal accumulate over time.

Appendix

Survey - Customer Support

I interviewed each member of the customer support team which includes 3 front line support staff and 2 Field Application Engineers (FAE). The questions and answers are shown below. The results are the total responses from both groups, unless otherwise indicated.

1. How many customer calls or e-mails do you handle per day?
55
2. How many calls or e-mails result in a trouble ticket?
32
3. How many tickets relate to issues with the SDK interface?
19
4. How many of those trouble tickets do you close without opening a corresponding defect report for the development team?
10
5. What percentage of your time is spent investigating and raising defect reports related to the SDK interface?
Front Line Support: 65%
FAEs: 40%
6. What is the general attitude of the customers you talk to?
 1. Cordial
50%
 2. Frustrated
35%
 3. Angry
15%
4. How many calls or e-mails do you receive each week from the sales staff complaining on behalf of one of their customers?
5

References

<http://www.cigital.com/papers/download/ndia99.pdf>

<http://amartester.blogspot.ca/2007/04/bugs-per-lines-of-code.html>
