Quality Improvement Through Test Automation: A Proposal

Author: Julia Keffer

For: PT Company Inc.

Date: June 12, 2013

Table of Contents

Executive Summary

The purpose of this proposal is to demonstrate the effect that an increase in the number of defects in our SDK (Software Development Kit) product is having on our customers and our company, and to recommend a solution to resolve this problem.

Customer defect reports in our flagship SDK product have increased over the past three years, causing customers to become frustrated with using our product in their development. 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. The increase in defects needs to stop - now.

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

The solution must use existing staff, minimize the impact to release schedules, and be effective over the long term.

The proposed solution is to build a test framework to enable automated interface tests. This functional area accounts for 40% of undetected defects. The tests are high value, well-defined, low maintenance tests.

Product 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 can write new tests as part of developer testing for this and all future releases. If 50% of existing tests are automated and tests are written to cover 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, to detect and fix defects related to changes in the interface code as early as possible, before they escape to product verification or to the field.

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

Without action, the situation will get worse. Adopting a third-party framework instead of building one 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.

The proposed solution 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 proposal is to examine the increasing number of customer complaints related to product quality in the company's flagship SDK product. This proposal describes:

- the problem
- the impact
- two of the causes

It presents:

- the proposed solution
- the implementation
- the costs
- the conclusion

Problem

Customer defect reports in the company's flagship SDK product, which accounts for the majority of the company's 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 increases the time before the company realizes significant revenue from a sale. Customers who are unsuccessful during the evaluation process may abandon efforts to use the SDK, costing us potential sales.

This problem is also affecting customer support staff. Interviews with the support team indicate that the time they spend investigating problem reports has increased over this same time period and it now consumes an average of 50% of their time.

Implementing new features or fixing defects can introduce defects in two ways:

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

The software 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.

As an example, the SIP functionality implemented in the last release required reworking the software architecture of 40% of the code, which increased the complexity. It also introduced 50,000 (50 kLOC) new lines of code.

An analysis of the data in the defect tracking database shows that in the code base, the number of defects per kLOC raised in the field has risen by 15% over each previous release. Figure 1 shows the trend.

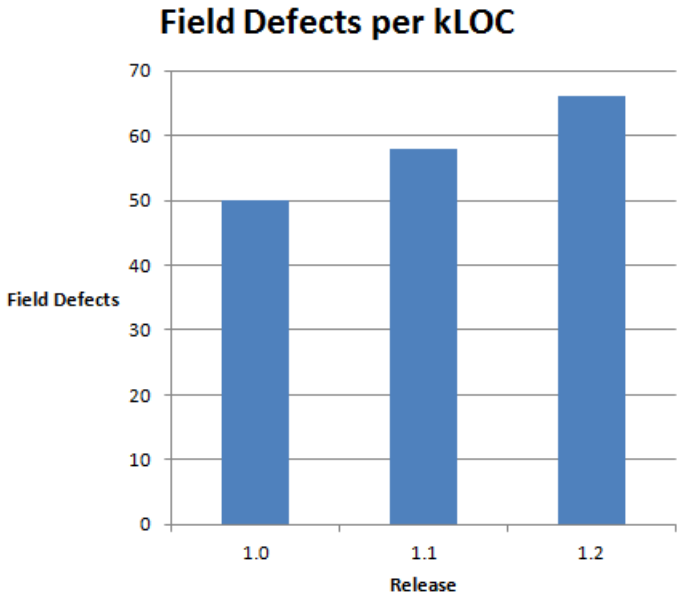

Figure 1: Field Defects

Keep in mind the following:

- 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. The trend cannot continue.

Another concern is the number of patches issued to fix defects in the field that blocked customer development. Ten patches were issued 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. Figure 2 illustrates the trend.

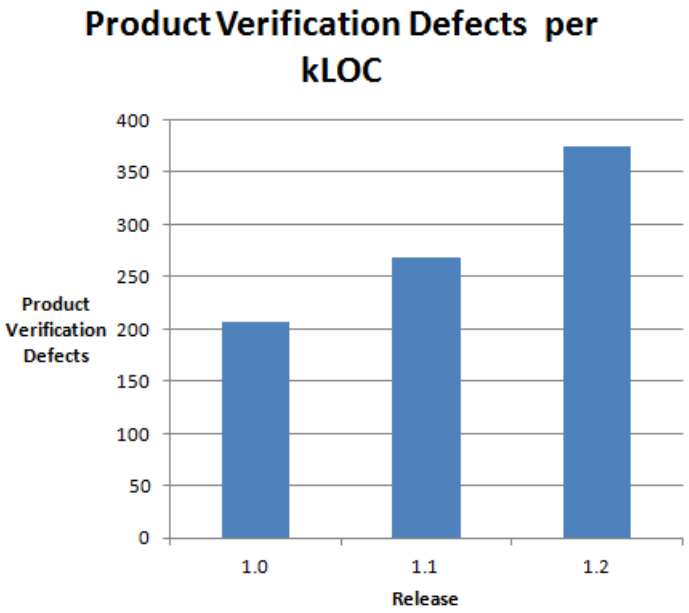

Figure 2: Product Verification Defects

Causes

To proceed with a solution, it is necessary to understand why defects exist. There are a number of reasons; however, there are two causes that 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 product tester is to break the software. The software developers' mindset can lead to overlooked failure paths during software unit testing.
- The more defects product testers find during product verification, they more time they spend retesting the defect fixes, which takes time away from testing other scenarios. There is typically a fixed amount of time allocated for the product verification phase, which can result in untested defect fixes and inefficient use of time.

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

Solution Criteria

A successful solution meets the following criteria:

- No impact to the existing budget

- Minimal impact on current and future product release schedules
- Addresses the situation over the longer term

Solution

The proposed solution is to introduce test automation to reduce the problems described above. Figure 3 shows the analysis of the defect categories in the defect tracking database. Defects that occur in parameter checking and functional scenarios (the SDK interface) represent 40% of the total defects found during product verification and in the field.

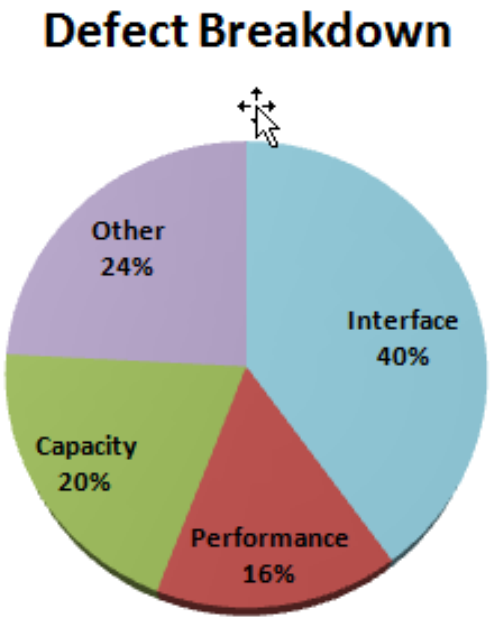

Figure 3: Defect Breakdown

Tests to cover this category of defects lend themselves well to automation because they:

- have well-defined boundaries
- have unambiguous pass/fail criteria
- represent actual customer use cases

The 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 and suggest additional scenarios to add to software developer testing. They can then maximize the use of their testing time by focusing on tests that are not suitable for automation. Software developers can learn from the feedback they receive from the product testers.

The proposed solution involves implementing a customized test framework. 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.

Automated tests will run nightly, as part of the overnight software build. Figure 4 illustrates how test automation works.

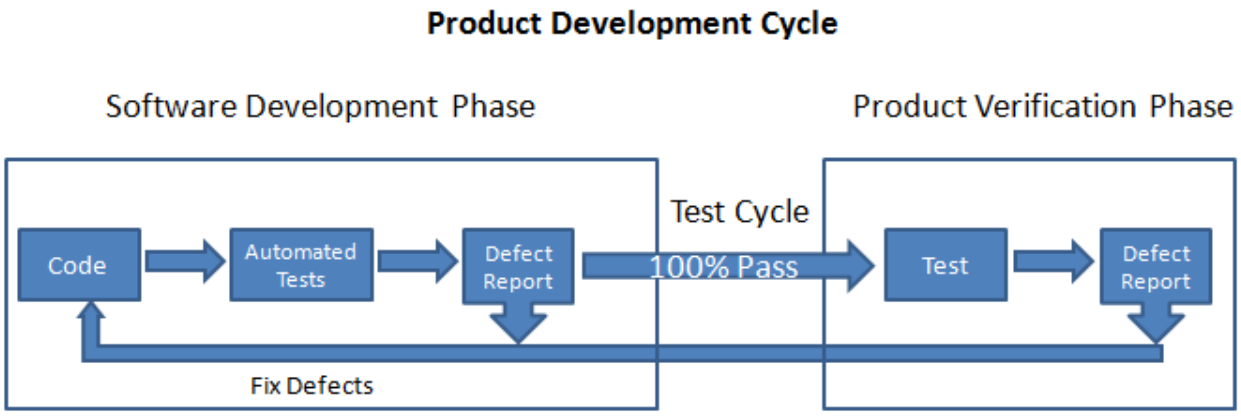

Figure 4: Product Development Cycle

Each day, software developers submit new code. The automated tests run as part of the nightly software build and produce a report of test failures that indicate defects introduced by the previous day's code. Software 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 defect that escape from software development through to product verification, which reduces the chance of undetected defects escaping to the field.
- It provides an opportunity to detect defects in patches before they are released to the field.

The strategy's success can be determined by analyzing the defects in the defect tracking database. There should be fewer defect reports related to the interface from both the field and in product verification.

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 expected to change in the next release (6 person weeks)
- Integrate tests into nightly software builds (2 person weeks)

Figure shows the proposed schedule.

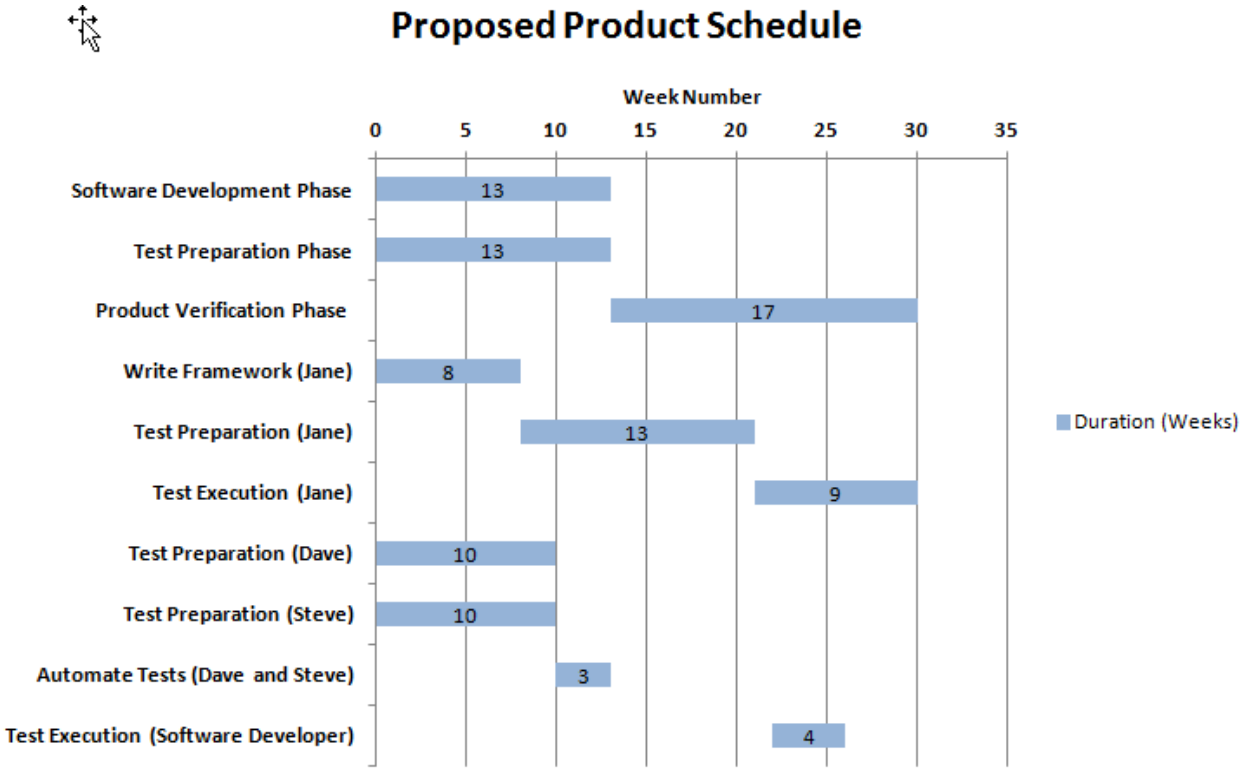

Figure 5: Proposed Product Schedule

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

The set of existing tests to automate covers approximately 50% of the interface functions. The number of interface functions is projected to expand by 25% as new features are introduced over the next three releases. If software developers add tests as they introduce new functionality and add tests as they fix defects in the interface that are not covered by existing tests, after three releases, the test suite will cover 75% of the functionality.

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 product 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, Jane can continue 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.

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 software developers become familiar with the test framework. However, the overall time spent writing automated tests should be approximately equal to the time spent running the tests manually.

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.

Solution Criteria

The strategy meets the solution criteria:

- There is no impact to the budget because existing software development and product verification staff implement the solution and their salaries are already included in the budget.
- It does not involve any new purchases that are not included in the current budget.
- Four weeks is only a 15% increase to the current schedule and there are no changes anticipated to future schedules.
- The benefits accumulate because the test suite grows over time, and because the nightly test runs continually detect when new code breaks existing interface-related functionality.

Benefits

- Increased customer satisfaction shown by fewer support calls and complaints to sales staff.
- Reduced risk of customers giving up on the product because they cannot develop their solution.
- Increased discipline applied to analysis of the software under test and test planning.
- As the automated test suite grows, the benefits of automation accumulate over time.
- Increased morale in the product testing 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.

Alternatives

To help evaluate the proposed solution, this section describes and analyses the following alternative solutions to the problem:

- Continue with only manual testing.

- Buy a third-party test framework.

Continue with Only Manual Testing

If no action is taken, the current trends will continue. Customers will continue to experience the effects of poor quality, with the impacts 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 the company's current revenue.

It does not address the solution criteria because there is no future benefit and ongoing quality problems will increase the time needed for future releases.

Use a Third-Party Framework

A third-party framework is not recommended for the following reasons.

- It may not save time because of the need to evaluate different products.
- There is no money in the budget to purchase a commercial product.
- To avoid the cost of maintaining the code base, it is necessary to choose an open source framework that has active development by the open source community. Otherwise, there is no advantage over building a custom framework.
- A third-party framework is unlikely to meet our exact requirements. It takes extra 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 allows the flexibility to make changes only when necessary.
- Developing an in-house framework enables a solution customized to our product.

It does not address the solution criteria because a commercial tool requires money that is not in the current budget and the time to evaluate a new product will increase the current schedule.

Conclusion and Recommendations

Action is needed to improve product quality and customer satisfaction. The increasing number of defects in the SDK is causing an increase in customer complaints and customers are having difficulty adopting the product. This is delaying or reducing revenue and customer support staff are spending time handling trouble tickets when they could instead spend time helping customers adopt the SDK more quickly.

The proposed solution is to introduce test automation with the goal of reducing the number of undetected defects that escape through to product verification and into the field.

The solution involves developing a test framework in-house, which provides the following benefits:

- Increased customer satisfaction shown by fewer support calls
- Faster time to market for the customers and reduced time to realize revenue
- Increased discipline applied to developer testing

- Increased morale in both the software development and product verification teams

The proposal:

- Requires **no** new staff
- **Minimizes** the impact on the schedule of the upcoming release to four weeks; there should be **no** impact on future development schedules
- The benefits of this proposal **accumulate** over time.

The next step is to organize a meeting with the key software developers and product testers to begin analyzing the requirements and planning the design of the new framework.

Appendix

Survey - Customer Support

Each member of the customer support team, which includes three front line support staff and two Field Application Engineers (FAE), filled in the following survey. 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

Glossary

FAE -- Field Application Engineer

kLOC -- Thousand lines of code

Nightly software build -- overnight compile of the software to build the software installation program

Product Verification -- Product level testing that uses no knowledge of the software implementation

SDK -- Software Development Kit

Unit testing -- Testing by software developers to verify that a component of the software is free of defects

References

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

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