

STeP-IN SUMMIT 2008

5th International Conference On
Software Testing

Structured Testing approach for improving Product Quality – A practical case study

by

Sudhir Patnaik
Group Manager - QA

&

Balasubrahmanyam Pillalamarri
Senior Software QA Engineer

Intuit India Development Center

Copyright: STeP-IN Forum and Quality Solutions for Information Technology Pvt. Ltd.

Published with permission for restricted use in STeP-IN SUMMIT 2008 in agreement with full copyrights from owner(s) / author(s) of material. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior consent of the owner(s) / author(s). This edition is manufactured in India and is authorized for distribution only during STeP-IN SUMMIT 2008 as per the applicable conditions.

Practices Experience Knowledge Automation

Produced By

STeP-IN
Forum

www.stepinforum.org

Hosted By



www.qsitglobal.com

Assessing improvement in product quality quantitatively has always been a subject of debate for many years and there are still methods, techniques, formulae to arrive at quantitatively measuring this aspect of quality. Some methods have evolved over the years but yet, on the verge of release or even after the release, we still ponder on the topic “if the product met the quality criteria”, noting the fact that defects are still seen from the field after the product release. This brings to the question as to “what is an acceptable quality criterion” for any product to go out of release and for an existing release, how to assess that there is an improvement in quality of the product from its previous release? If the answer to this question is yes, then we need to know what methods, techniques were adopted or improved upon in the product development cycle including testing, which resulted in this improvement? Can this be benchmarked? Or can it be made a best practice in the Organizations so that all the products developed across silos can follow this best practice to be able to – not only improving the product quality but also being able to quantitatively measure them as well?

In this paper, the authors will share one of such best practices in end-to-end product development including development-build-test phases, which resulted in improvement of product quality that can be tracked, measured and improved. One such case study which helped achieve greater product quality is a project on re-engineering to replace an existing legacy platform framework to achieve better performance improvements and quality of the code delivered to the application development teams for GUI development. The tests existed for legacy platform code needed to be re-engineered to be able to run them on the new technology framework. It involved developing an entirely new test framework using C# and .NET that eventually led to an ambitious quality goal of 80% code coverage. Basically, it involved certain process discipline in the entire development process such as (a) Execution of DBTs (developer build tests) prior to code check-ins to ensure that the added code has not broken the existing functionality (b) Upon successful check-in, the build goes through Feature Sanity Tests (FSTs) and Build Acceptance Tests (BATs) automatically on a QA infrastructure (called TCoE – Test Center of Excellence) and (c) System testing subject to a successful build pass of these tests. There are other aspects of the end-to-end product development process that the authors would like to touch upon in this article, which could help teams involved in product development to adapt this process discipline.

How the audience would benefit out of this paper:

1. How can product development teams achieve greater quality by “building in” quality through a “bottom up” approach?
2. What factors contribute to quality of the product?
3. How these factors can practically contribute to the product quality?

Introduction

Assessing improvement in product quality quantitatively has always been a subject of debate for many years and there are still methods, techniques, formulae to arrive at quantitatively measuring this aspect of quality. Some methods have evolved over the years but yet, on the verge of release or even after the release, we still ponder on the topic “if the product met the quality criteria”, noting the fact that defects are still seen from the field after the product release. This brings to the question as to “what is an acceptable quality criterion” for any product to go out of release and for an existing release, how to assess that there is an improvement in quality of the product from its previous release? If the answer to this question is yes, then we need to know what methods, techniques were adopted or improved upon in the product development cycle including testing, which resulted in this improvement? Can this be benchmarked? Or can it be made a best practice in the Organizations so that all the products developed across silos can follow this best practice to be able to – not only improving the product quality but also being able to quantitatively measure them as well?

The project referring to here is a re-engineering effort to replace an existing legacy platform framework to achieve better performance improvements and quality of the code delivered to the application development teams for GUI development. The tests existed for legacy platform code needed to be reengineered to be able to run them on the new technology framework. It involved developing an entirely new test framework using C# and .NET that eventually led to an ambitious quality goal of 80% code coverage.

A brief about the quality goals stated at the start of the project:

- 90% of code will be reviewed
- 90% of all feature changes will be design reviewed
- All test plans/Quality plans will be reviewed
- Unit tests to cover >70% of code path
- Post “code complete” defects at X% or less compared to previous release
- Defect density at System test 50% less than company average
- 95% or more build success on group branch (one level above team branch)

The authors would like to share their experience in achieving the aforesaid quality goals.

Benefits of Team Software Process

Team Software Process (TSP) supported the team in going about their tasks. Team sat together and detailed all the tasks into 5-15 hr/task. All the dependencies were noted. The same was informed to the management which gave a go-ahead to the project based on the task details provided by the team. Team used to meet every week to understand any issues/roadblocks being faced by the team. Team management

used to take note of the issues and tried to resolve them as early as possible. Risks were identified pretty early in the life cycle which gave the team ample time to tide over them. TSP process allowed the team to calculate the Earned Value (EV) and the task hours achieved by the team every week. This helped team manage the work schedule effectively through out the product life cycle and achieving low defect density. TSP allowed enhanced communication with all the relevant stake holders. Peer reviews and inspections were done for all the new code added to the product.

Benefits of following PSP/TSP:

- Helps engineers build time in their plan for design reviews, code reviews and inspections
- Reduced over-commitments by software engineers and the team
- Earlier the defects found, less time it takes to fix them and also saves System Test time/effort
- Improved productivity since engineers are in better control of their own work since they estimate the task hours
- PSP gets software engineers involved in process improvement and boosts organization's process improvement effort

The Product Development Process

Developers were made to write unit tests to test every line of code added to the product. Team also set a goal not to allow the code coverage fall below the value set at the start of product development. Any new check-in should have a DBT (Would be discussed later) pass result associated with it. This made the QA team's job easy as every check-in was thoroughly tested even before the build was triggered. Incremental integration testing approach was followed which allowed the engineering team release code drops every month. This in turn helped the team identify and fix bugs very early in the product development life cycle.

Product Quality Improvement

Developer Build Tests

Developer Build Test (DBT) is a tool that allows a developer to run automated tests against their local QuickBooks build. This way an engineer can verify that their changes do not adversely affect the build prior to checking in their changes.

- DBT helps developers
- Verify changes do not break BATS, FST's or other test areas
- Verify bug fixes to BATS, FST's or other test areas
- Verify changes do not unexpectedly break existing functionality
- Verify changes on different skus/flavors of QuickBooks
- Verify changes on different Operating Systems
- Verify changes in a non-developer environment

- Give Tools/QA a heads up at coming UI changes which may adversely affect the existing tests, allowing time for automation engineers to update affected areas prior to the change appearing in the build.

As discussed previously, all the code check-ins would undergo a round of DBT. This helps the team in two ways. Firstly, it allows the flexibility for the developer to run the existing tests against newly developed code even before the build is triggered. Secondly, it gives a sense of confidence to the developers that their check-ins would not adversely affect the quality of the product. Once the developers are done with their changes after successful execution of the DBTs, code gets checked-in to the Version Control System. Build triggers were also customized. Nightly builds were generated. Build Acceptance Test System (BATs) were automatically triggered by the existing infrastructure once the build was successfully built.

Build Acceptance Tests

All test scripts were stored in Perforce (Version Control System). They were stored by release and by branch. One needs to understand the structure so that he can find the scripts for a particular branch. A series of simple tests covering major functionalities are called as BATs. These tests were run on all the builds across all the branches. In case of the product we tested, it included some simple tests like updating a company file, creating a new company file or making a backup of the existing file and checking if the functionality is not broken.

Ex: File Operation Tests which include the following:

- Update a company file to QuickBooks
- Create a new company file
- Make a backup

Feature and Sanity Tests

FSTs are scheduled to run automatically after a build passes BATs. Results of the BATs and FSTs can be checked on the Intranet Web center. Feature Sanity Tests (FSTs) covers major areas of the product with tests identified based on the feature. Sample list of the FSTs include:

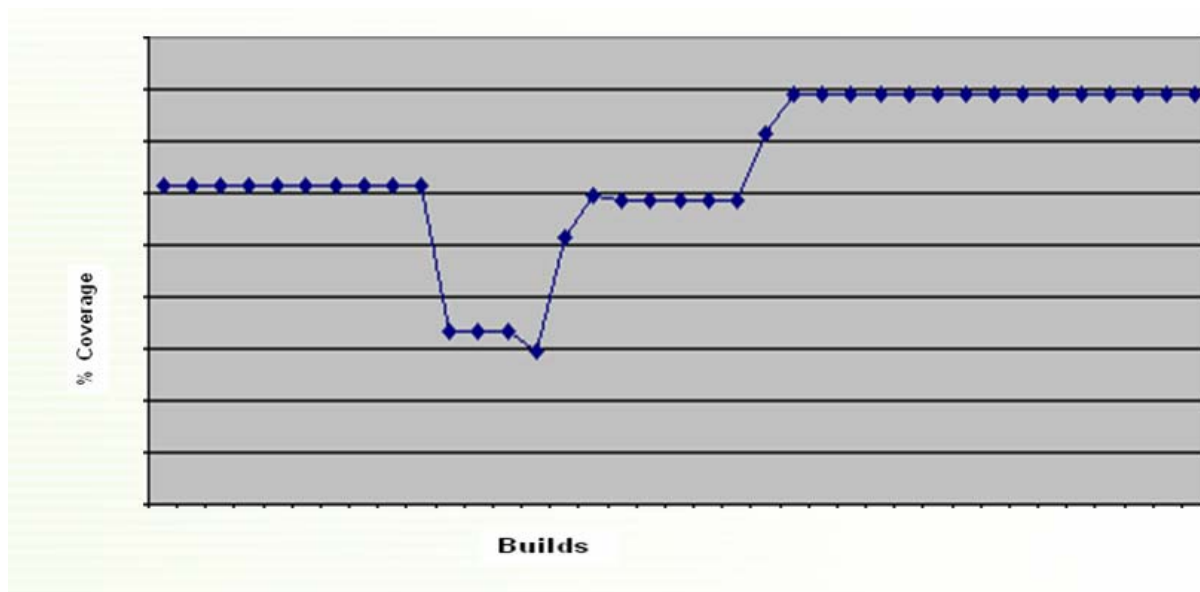
- Core FST
- Atom FST
- Payroll FST
- Multi User FST
- Reports FST
- IMS FST

Build would be considered for subsequent test execution only if the BATs and FSTs are passed. Infrastructure also provides various options through which one can triage a failure of the test. Once the build clears BATs and FSTs, it was accepted for further testing by the QA team. Results were stored on a shared location so that engineers can access them and understand the results. A series of tests were run through On-Demand system. On-Demand tests were the tests which were integrated into the Testing Center of Excellence (TCoE) infrastructure. These tests were run in an automated fashion. These tests included Unit tests, Smoke tests, Performance tests and Regression tests.

Unit Tests

Unit tests were executed on the build. An automated unit-test framework was developed using open source unit tests in C# and .Net framework on a new development project. Prior to this implementing this framework, products at Intuit were tested for code-coverage using heavy tools like McCabe IQ. But, using these heavy weight tools and subsequently triaging them involved a lot of time and resources and hence the need to start using light weight tools (NUnit, NCover and NCoverExplorer) which would give results of code coverage within no time and very well fit in to the agile development process. A test framework was built using these tools which is reliable, scalable and can generate reports based on function coverage as well as sequence-point coverage (statement coverage). This gave a better idea of where our code stands in terms of the coverage numbers. By using these tools we continuously ensured that we didn't fall below the set code coverage goal. Bugs found using these unit tests included but not limited to garbage collection bugs (GC), bugs related to exception handling. Having achieved this success, other projects have started using this as a best practice in their projects due to its generic nature and coverage reporting. Code coverage graph below Fig 1

Fig 1



Performance Tests

As stated at the start of this paper, business goal was to achieve 4X improvement in the performance. With that stated objective, team built a performance test framework using Microsoft Windows Powershell. Microsoft Windows Powershell enabled the team develop a performance test framework. Windows Powershell can be used to test Software Development Kit (SDK) or platform/technology frameworks (developed on Microsoft platform) that enable sharing data directly with Client applications. It is based on programming standards such as XML (eXtensible Markup Language), COM (Component Object Model), and HTTPS (for communication over the Internet). With the help of Windows Powershell, all types of consistent test frameworks such as Unit, Integration, Functional, and Performance can be easily & quickly developed as it is based on .NET and hence provides direct access to the entire CLR, plus the ability to script existing COM (ActiveX) and WMI objects. The SDK functionality is tested at multiple testing stages- Unit, Functional, System and Performance. The requirement was to develop a common test framework which will perform all the above through a command-line interface as SDKs' tend to be non-UI based. Currently the Unit, Functional & Performance tests are automated using this framework with all the inputs and outputs using XML technology. This framework got integrated into the Testing Center of Excellence (TCoE) framework. Developers and the Test Engineers had the allowance of executing automated tests at any point in time.

Branching Strategy

There was a set of criterion defined for promoting the code form a child branch to a parent branch. A brief overview of the promotion strategy is presented here.

Every feature development team owns a branch at the feature level. Depending on the number of features in the product there would be as many branches. Each child or feature branch would ultimately get merged into the parent branch. These tests include but not limited to all major features of the product. It is a step above FSTs in the sense that this set includes more tests covering more features than normal FSTs. The team executed all the tests successfully. This promotion criterion helps built quality into the product from within.

System Testing

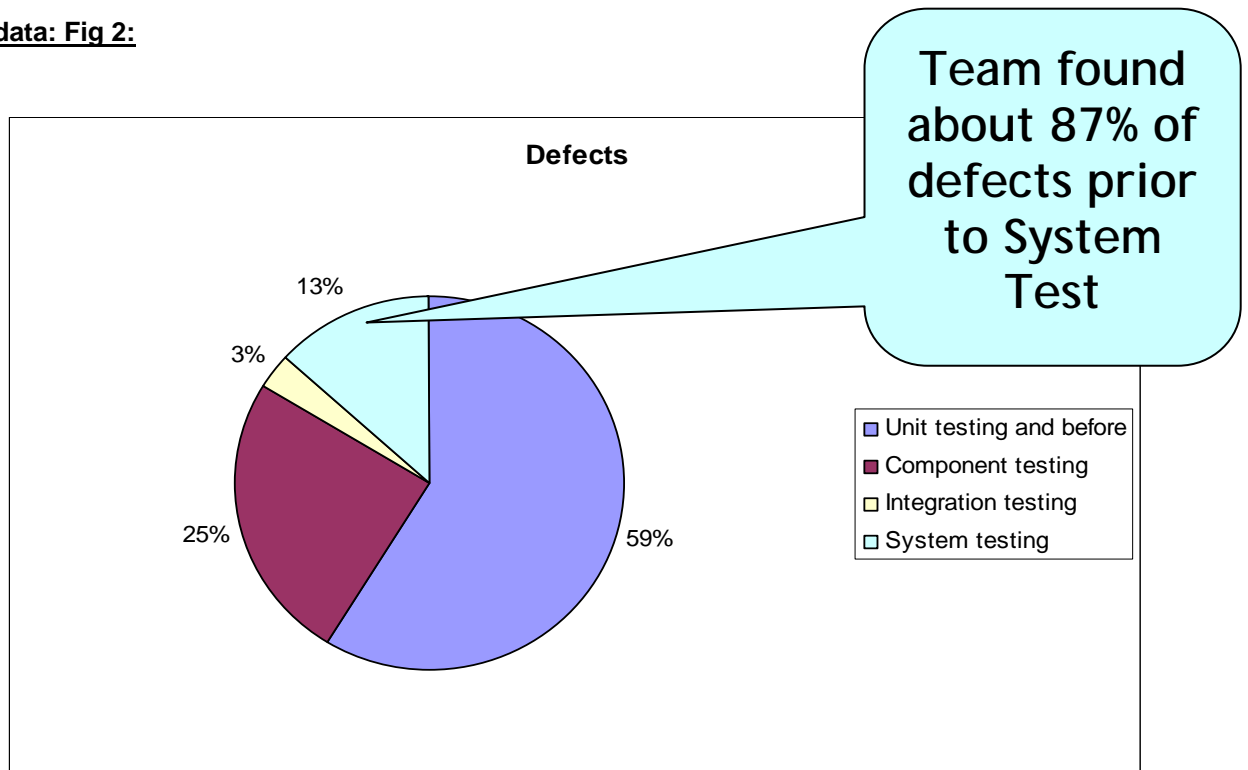
Since the product was subjected to a rigorous series of tests, the amount of effort the team had to spend during the system testing phase was minimal. The quality of the product was apparent as **87%** of the defects were caught well before the System Testing phase. During the system testing phase, a stable build was identified and all the tests were executed on that. These tests included DBTs, BATs, FSTs, Unit tests and performance tests.

Regression Testing

During the regression testing, unit tests and performance tests where executed on the build just before the Market release. The team can be proud that after one month into the market, there are no bugs reported regarding the performance of the infrastructure queries.

Final results:

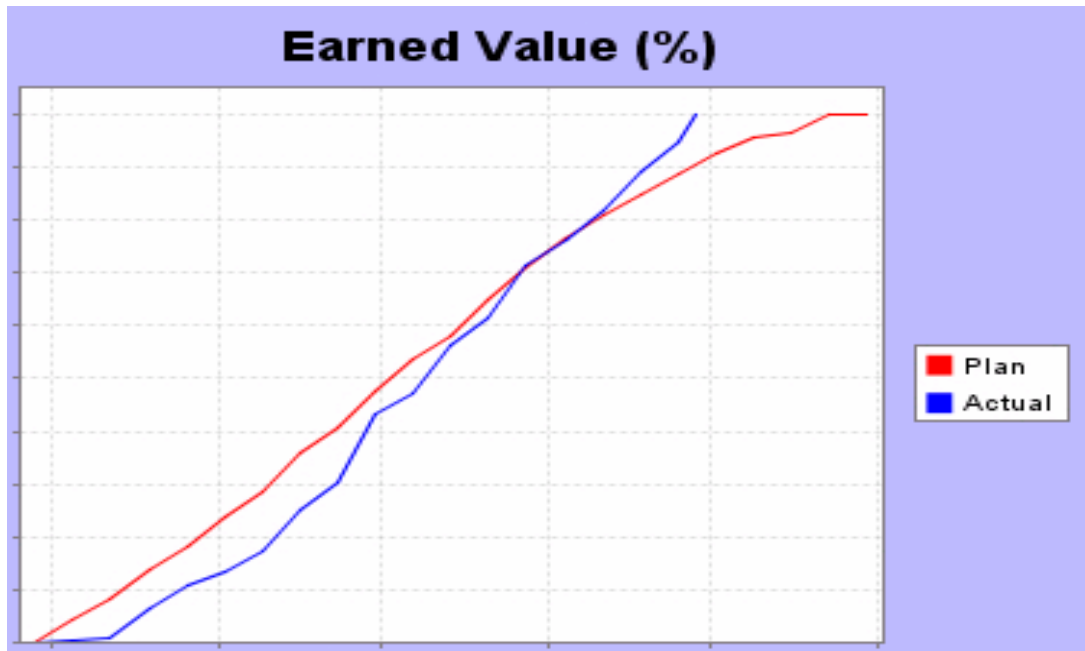
Defect data: Fig 2:



Quality results: Fig 3:

Quality Parameters	Results
System Test Defect Density	70% of Plan
Build success on group branch	100% of plan
Code review	90% of plan
Design review for feature changes	90% of plan
Unit Test Code coverage	99% of plan

Earned value through TSP: Fig 4:



Effort hours: Fig 5:



Effort Summary: Fig 6:

Effort Summary	(% Plan Vs Actual)
Weeks until Code Complete	0
Effort (task hours)	-10
Developer Productivity (LOC/Hr)	25
Size (LOC)	1

Conclusion

Following a “bottom-up” approach of building quality into the product immensely helps team in delivering better products on time and with quality. The processes mentioned would be very helpful in achieving this goal. Above mentioned process can be followed as a best practice in the Organizations so that all the products developed across silos can follow this best practice to be able to – not only improving the product quality but also being able to quantitatively measure them as well.

Biography of Sudhir Patnaik

Sudhir has 12+ years of experience in academic research, software development and testing. Prior to joining Intuit, he was working as Group Test Manager at Infosys Technologies Ltd., Bangalore and at Accelrys - Bangalore R&D Center of Excellence as Director of Product Testing & Technical Services team responsible for testing of products in the Life Sciences domain. Other companies where he worked earlier include Misys Healthcare Systems Inc., Tucson, AZ and Siri Technologies, Bangalore. Sudhir has worked in different industry domain areas such as Aerospace, Logistics, Healthcare and Life Sciences involving client/server, legacy and Internet technologies. His area of experience in testing includes establishing test teams and test labs, developing testing strategies, test metrics, identifying automation opportunities, and management of the test life cycle. He has utilized various automated testing tools such as WinRunner, SilkTest and worked on varied and diverse platforms. He has published & presented many articles in QAI, SEPG and STeP-IN Conferences in India. Sudhir is a member of Steering Committee in the STeP-IN Forum, Bangalore SPIN and Association For Software Testing (Cem Kaner). He has a Bachelors degree in Electrical Engineering and Master degree in Electronics & Communication from REC, Rourkela.

Sudhir is currently Group Manager – QA at Intuit India Development Center managing testing of products developed in financial domain.

Biography of Balasubrahmanyam Pillalamarri

Balasubrahmanyam Pillalamarri has 5+ yrs of professional experience in QA & Testing. He worked with reputed organizations like Symphony Services and Intel. His professional experience includes work on various domains that include instant messaging, Wireless Networking, Finance and Telecom Cost Management.

Balasubrahmanyam is currently working with Intuit India Development Center as Senior Software QA Engineer.