

## Providing a Separate Quality Assurance Team for a Project-Oriented Software Engineering Seminar

Jim Leonardo, Milo Auguste, Ritu Mehrotra

### Abstract

As the major focus of Pace University's CS615-616 Software Engineering Seminar taught by Dr. Charles Tappert, students form project teams to create real world systems for actual customers. One of the project teams was dedicated solely to providing ongoing quality assurance for the other project teams. The goal of this project team was to show that providing quality assurance and testing early in a project will reduce the overall work effort required. This paper describes the methods used and results achieved by this team as well as recommendations for improving the impact of future quality assurance teams in this seminar. Three case studies are used to demonstrate that early quality assurance and testing did have a positive impact even when the projects in question underwent substantial amounts of refactoring, or reengineering, or redesign.

### Introduction

As the major focus of Pace University's CS615-616 Software Engineering Seminar taught by Dr. Charles Tappert, students form project teams to create real world systems for actual customers. This provides an excellent setting for students to learn the "value" skills needed to function in real world team environments [Tappert2002]. As part of this, an independent Quality Assurance team was established. This team has had the responsibility of providing QA to all seminar projects undertaken in the Fall 2002 – Spring 2003 School Year for the White Plains campus. This has helped to resolve the commonly observed human phenomenon that developers have a difficult time honestly evaluating their own code for quality. This team's sole purpose is to provide this assurance through review, testing and establishment of best practices.

The issue of software quality has been debated as long as software has been written for one person to be used by someone else. As software moved away from custom made, machine specific code to off the shelf systems that may run on multiple operating systems with components from multiple hardware

manufacturers while having to coexist with multiple other running programs in multi-tasked environments, the issue of quality has become more and more important while becoming more and more elusive. In the current software world the task of making sure that a given software product is capable of running on the number of platforms that it is likely to encounter seems overwhelming. Especially in Web-enabled applications, where the developer has limited if any control over the environment their application will be running in, it seems impossible to produce a truly defect free system that meets the other desirable aspects of quality. Adding to the burden of producing defect free high quality software is the trend towards using reusable software components. If a developer were to develop an otherwise perfect application utilizing an off the shelf database management system that contained numerous flaws, the overall level of quality of their product would suffer greatly.

So, what constitutes quality? According to the American Heritage Dictionary [American Heritage 2000], quality can be defined as "An inherent or distinguishing characteristic; a property" or "Superiority of kind". When overall software quality is discussed, there are really four main components that are being discussed. These are correctness, reliability, security, and usability. There is a fifth component that is rarely discussed as a portion of software quality, but it is just as important.

Correctness is defined as the degree to which software performs its required function. [Pressman 2000] This is often measured as number of errors and or defects uncovered in the software. This risks oversimplifying correctness as well as reducing the scope of reliability. One has to define what an error is in order to understand what correctness truly is. An error is a failure to conform to requirements in some manner. Correctness is not merely conformance to specification (explicit requirements), but also conformance to expectation. Conformance to specification means that the software conforms to the *explicitly communicated* requirements of the software. Conformance to expectation means that the software conforms to the *implicitly communicated* requirements

as well as the explicit requirements. There is also the problem of requirements that have not been communicated to the development team due to a variety of factors. In order to be correct, software must also meet these “mind reader” requirements.

Reliability is often considered a part of correctness, but for the purposes of this paper, it will be considered a separate characteristic. Reliability is the ability of software to perform correctly over time. In the modern software era, this includes the ability of the software to execute correctly in a vast variety of unpredictable environments. Any software that is designed for multiple users must also be able to reliably execute regardless of the numbers of users accessing the system, or at least gracefully prevent excessive numbers of users from accessing the system. Reliability also measures the ability of the software to handle unpredicted exceptions that may arise from a variety of reasons without compromising data or the system it exists in.

Security comprises two separate concerns. The first of these is that the system must be able to withstand attacks by malicious persons. The attacks may be attempts to access the system without permission for reasons such as theft or system hijacking, or they may be vandalistic in nature. A vandalistic attack is one where the hacker has no purpose other than to destroy or cause chaos. Examples of vandalistic attacks include viruses such as the disk erasing Michelangelo virus from the early 1990’s or the now famous Distributed Denial of Service attacks (which often hijack some systems in order to vandalize others). The second component of security is the need for the system in question to not compromise other systems that may exist in its environment.

Usability is the last commonly discussed component of software quality, but it alone can have a greater impact on the perception of quality than any other component of quality. [Pressman 2000] Usability is generally perceived as the “user friendliness” of the system, but it also should include the degree of efficiency with which the user is able to use the system. A truly usable system must not only be friendly to the novice, but it also must absolutely be friendly to the expert. The need for high usability is highlighted by the fact that time constraints experienced in the workplace limit the amount of formal training that can take place [Butler 1996].

The black sheep of the software quality family is maintainability. In a business environment where being a month ahead of your competition can mean the difference between an 80% share in the

market and a 10% share in the market, it is tempting to focus on short term benefits over long term benefits. The repercussions of this mentality can be very long lasting and impact the ability to release a second version in less time than your competitor. Shortcuts that are often taken include failure to document algorithms and objects, failure to maintain architectural diagrams of various types, failure to split components up into reusable pieces, and a general failure to provide the tools needed to keep the application running into the future. It has often been stated that the majority of the work related to an application lies in maintaining it, but this is the most often overlooked piece in development.

During the course of these two semesters, the QA team has observed the progress of the implementation teams and reviewed their deliverables periodically. Additionally, an online questionnaire has been developed to collect input from the implementation teams regarding their progress. This has helped to reveal how degrees of conformance and correctness fluctuate through development.

### Relevance

Quality Assurance is a necessary step in any software project. Research has shown that organizations with separate Software Quality Assurance groups have a better foundation for producing a superior product in terms of performing the actions associated with high quality software. Unfortunately, with collection of actual quality related metrics being kept to a minimum by most organizations, it is hard to conclusively state that independent SQA groups do promote increased overall quality [Wheeler and Duggins 1998].

Due to the concern that software professionals have that recent graduates do not have sufficient understanding of what is involved in the creation of large, complex systems, some universities have steered their senior capstone courses away from typical coding exercises and more towards total involvement in involvement in all of the other tasks that typically comprise the effort required to produce a real world project [Rising 1989]. This includes all of the design tasks, documentation, and verification and validation. Rising had assigned her students to specific roles and had assigned a dedicated Quality Assurance Manager who was responsible for the overall product quality.

While the Software Engineering Institute has identified a need for a course in Software Verification and Validation geared towards quality assurance [Gibbs 1989], the fact that as late as 2000 reports

describing making SQA a part of large project classes as the only available forum within the a curriculum for doing so would indicate that this is not yet a reality in many universities [Hilburn and Towhidnejad 2000].

## **Methods**

There are three main methods used by the CS615-616 Quality Assurance Team to perform their task. These are establishment of best practices, testing of developed software for conformance to specifications and testing of software for ability to operate under sustained load.

Best practices are established by the QA team based on their own experience and documented best practices from other sources. The practices applied to a given project will depend on its requirements and the technology used. Some practices include:

- Remote Database Management Systems such as Oracle or MS SQL Server should be used in preference to local databases such as MS Access.
- If it is optional, then variable declaration should be forced in source code.
- Comments should be inserted before or during source code creation, not after.
- If public classes are created, documentation of the public interfaces should be maintained separately from the internal developers' documentation.
- Browser based applications should be compatible and tested with more than one browser.
- Any user input should be pre-filtered before being sent to the database to prevent errors on the backend.
- Websites that require login should also provide a logout. Such pages should also automatically direct the user to a login page if they attempt to navigate to that page without being logged in.
- Navigation on web pages should be clear and should be not allow the user to get to a dead end page that they can't navigate away from.
- Fields, buttons, etc. should be labeled consistently. E.g. If one screen has a button labeled "Submit" and another a button labeled "Save", there should be some underlying reason why they have different names. Similarly, if a field is labeled "Category" on one screen, it should always be labeled "Category".

Conformance with best practices has been evaluated by examining all documentation associated with the project, from the requirements through the

user documentation and including the source code. Evaluation for compliance with best practices will be the only area in which all projects will have their source code examined. The majority of actual testing will be black box testing.

Testing for conformance has been done on an ongoing basis as the implementation teams developed the applications. The goal was to discover and correct issues early. It has been shown in many prior reports that correcting errors and omissions early takes far less time and cost than finding and correcting them later. [Brooks 1995, Ambler 2002] The ideal set of tests consisted of two phases. The first phase was to take the use cases, other specifications, and documentation and then follow them through the application to the degree that they have been implemented to date. This is done without taking any overt attempts to cause exceptions in the application. Once the foreseen paths have been traced, the testers then start over and test the applications while attempting to cause errors. This largely depends on the application, but examples include attempting to enter invalid data, attempting to cut and paste URLs into browsers (which depending on the circumstance, proper behavior may be either succeed or fail.), and attempting to enter incomplete data. All of the points of quality are examined during both phases and feedback given to the implementation teams verbally or through email.

Load testing was performed for web applications using a tool that creates automated, scripted simulated users. This tool is OpenSTA ([www.opensta.org](http://www.opensta.org)), which allows a tester to record a session in an application and then modify the resulting script so that each simulated user is executing a different series of instructions and data. The amount of load depends on the application, but the general goal is to ensure that concurrency is maintained and data transactions remain atomic, consistent, isolated and durable (the so-called ACID requirements [Lynch et al 1993]).

When a particular set of tests is run, they are recorded and saved so that they can be run again when the next iteration of the application is ready for testing. This is known as regression testing. The key purpose of this is to ensure that in fixing errors or adding features, previously correct features are still correct. This is now a widely accepted, many would even say required, practice in creating multiple iteration projects. [Ambler 2002, Pressman 2001] The number of errors are collected and categorized. This data, along with more subjective feedback, was then given back to the implementation team. It was hoped that the QA team would be able to resolve this data so that all

of the projects could be compared equally. The major emphasis on quality is on the final product, not the interim iterations, so this data is purely informative. It has been recommended by other authors that metrics such as errors per thousand lines of code and similar metrics only be shared with the people whose code is being tested, not publicized [Pressman 2001].

There were three major types of systems tested. These were web based systems, telephone based systems, and Personal Digital Assistant (PDA) based systems. Despite the physical and conceptual differences present in the three types of systems, the general approach to testing remains the same. The only difference between the three was that only systems with web interfaces were tested for load and concurrency. It was hoped that the QA team would be able to test the telephone based systems for load and concurrency, but only one telephone line was available on the server being tested. The PDA systems were treated as standalone systems due to logistical concerns, so load and concurrency testing was not performed.

## **Results**

The QA team has examined several tools to assist in performing quality assurance, but has been unable to find a key piece for scripted testing of non-browser based applications. OpenSTA has been selected from a wide group of open source test scripting tools for browser based application. This selection was made because OpenSTA allows the tester to record a single path through a web site to generate the initial script. This script can then be randomized or branched so that when run to simulate multiple users, those simulated users will be following different paths. Additionally, metrics can be collected in running the scripts for later evaluation. There were a wide variety of such tools available under various freeware licenses, but OpenSTA provided the greatest degrees of flexibility, usability, and robustness.

The QA team has been unable to find an open source/freeware (or even low cost) tool that will allow a similar function to be performed for non-browser based applications. The chief benefit of such a tool is for regression testing, but it also aids in bridging from the first phase of conformance testing to the second phases. This is because the tester would be able to record a "correct" path and then create an incorrect path by simply modifying a few sections of the script.

Many of the projects have had significant changes to technology, scope, or focus. This has had a serious negative impact on the QA team's ability to

test and collect data regarding its impact on the overall quality of the other projects. From discussions with the implementation teams, many have run into problems with getting strong specifications or other details from their customers. At least one team has also completely changed its programming language and RDBMS since the first semester of the seminar.

Many teams, including the QA team, spent a great deal of time evaluating the technologies they would be using. On average, this was 3.33 weeks per team. There were 11 weeks from the time that initial requirements were to be collected until the end of the first semester. This indicates that many teams lost about 1/3 of the development time available in the first semester while making decisions about their technology platform. One team essentially lost most of the first semester due to the fact that they realized that their chosen web scripting language/engine did not suit their project well and changed everything at the beginning of the second semester. The QA team also has developed an impression that many teams ran into problems with configuring their technologies in the available server environments.

The QA team ran into numerous instances where even though the implementation teams were meeting deliverable requirements, they were not turning their systems over to QA for testing and evaluation. This had a serious negative impact on the ability of the QA team to provide feedback to the implementation teams. This also prevented the QA team from compiling meaningful data reflecting the progress and maturation of the applications. The QA team has been able to evaluate 4 systems (Genealogy, the Absentee system, the Yellow Pages system, and the Help Desk system). The Genealogy system was the only one of these 4 projects that represented a full, two-semester project. One team undertook the Absentee and Yellow Pages systems as smaller projects, and the Help Desk system was undertaken as a one-semester project by another team.

## ***Case Studies***

To show the effect of Quality Assurance, three of the projects will be briefly examined. It should be understood that these case studies only reflect the perspective and understanding of the project of the QA team and not of the implementation team, the customer, the instructor, or any other persons.

### ***Case Study: Team 1-Genealogy System***

Team 1 had a difficult time getting started. They were originally assigned to create a simulator for bidding in the card game Bridge, but the customer for

that system backed out. An attempt to have another professor at Pace serve as the customer did not work out (it is believed by the QA team that this was because nobody fully understood the intricacies of bidding for Bridge). After several other game type problems were considered, it was decided to the team would recreate a system from a prior class that had yielded less than satisfactory results. The goal was to create a web based system for tracking genealogy (a “family tree” system).

This team had several major hurdles to overcome. The first and foremost of these hurdles was to decide what software to use to serve the web pages that would provide the user interface and to decide what RDBMS to use. They were hampered by the fact that none of them had experience in web scripting languages and they had limited database knowledge. They originally tried to use Java servlets and Oracle. Because of the lack of database experience, the QA team carefully examined their proposed database schema. They ran into some learning curve issues with the database, but were generally successful in creating the routines needed to access it. They ran into several problems with the servlets, but managed to create a highly functional first iteration by the end of the first semester. This first iteration was reviewed by the quality assurance team carefully. A number of concerns ranging from functionality to usability were noted and reported back to the implementation team. During the inter-semester break, the team decided they were unable to complete the project in a timely and quality fashion using the servlets. There was also an issue in configuring Oracle on the server they were using. They decided to abandon the existing platforms and to start over from scratch using PHP and MySQL.

The system that was created with the servlets and Oracle was found to have seven major issues and a number of smaller cosmetic issues. Despite starting over, the implementation team did not repeat the issues that were uncovered by the original testing. The restart also enabled them to tackle some of the more fundamental issues such as somewhat confusing navigation from a fresh perspective. Naturally, more issues were discovered in the second and subsequent reviews. These were not fundamental flaws in general, and only one of them (the lack of a delete function) was deemed to be functional. The remainder of the issues were usability issues and were mostly easily resolved.

#### *Case Study: Team 2 – Voice XML Applications (Absentee System)*

Team 2 had several projects to undertake. All of these projects concerned utilizing the VoiceXML

standard to create telephone based systems. The QA team did not concern itself with the underlying technology, but focused on testing these systems once they were sufficiently complete. By the time of this paper’s writing, two of these projects had been worked on. The first one was to refine and port to a different server an existing web and telephone based application for reporting absences for classes.

The Absentee system was substantially refined at the back end and ported to the new server by the end of the first semester. At this time it was reviewed by the QA team. Three errors were discovered and two possible design issues (relating to choice of user name and password) were brought to the implementation team’s attention. The implementation team reported that the design issues were the result of limitations imposed by the hybrid web/phone approach. The other three errors were classified as one functional, one security/navigation, and one usability error. All three of these were addressed by the next iteration. The next iteration also featured a cosmetically redesigned web interface. This was tested to ensure nothing had broken using the same methods as in the previous tests. The QA team was able to verify that the implementation team had successfully redesigned the UI without a negative impact on the quality of the system.

The implementation team was also very careful to ensure that they conformed to the best practices laid out by the QA team during the redesign. It is difficult to say whether the well executed redesign was in part due to adhering to these best practices. What *is* known from discussions with the implementation team is that the best practices did help the implementation team to make the decision to that redesigning the interface was better than just patching it up.

#### *Case Study: Team 2 – Voice XML Applications (“Yellow Pages” System)*

As mentioned previously, a second project was undertaken by Team 2. This project was to create a telephone based “Yellow Pages” for the major departments of Pace University. The team needed to do two things for this project. The first was to design the telephone based user interface. The second task was to create a database accessible via a VoiceXML file. This was accomplished using PHP to create the VoiceXML files. The contents of the database were provided by the customer and consisted of the various departments and their locations.

The system was tested using both standard “corded” telephones and cell phones. The QA team

was unable to test the system for concurrency and load because there was only one phone line provided for testing and development. The system was discovered to have three functional issues and the QA team also raised a few questions about what constituted a department from the customer's standpoint versus the user's standpoint. There were also issues with the voice to text engine being used on the new server that were outside the control of the implementation team that made accessing the phone numbers for some departments difficult. These problems did not seem to be exacerbated by phone quality; they seemed endemic to the system.

At the time of the writing of this paper, the implementation team had not yet had sufficient time to implement the suggested changes. It is hoped that by the true end of the semester (this paper being written about one month prior to the end), these changes will be implemented.

## Conclusions

The QA team has formulated several recommendations that should increase overall product quality for future engineering seminars. These are:

- Technologies should be standardized so that teams do not have to perform their own technology evaluations. This will also help to maintain the applications in the future; the teams responsible for future maintenance will not need to learn a wide variety of languages. Teams would have the option to use other technologies only if they have a good reason to do so (such as customer requirements), but should be VERY strongly discouraged from doing so. The QA team recommends the following:
  - PHP as the main web scripting language. It is free, widely supported, available for all major OS platforms, and supports a very broad range of functions. It also is a C style language that is usually easily learned by those familiar with Java.
  - Java as the main standalone programming language, for the same reasons as well as being the main language taught at Pace.
  - Oracle or MySQL as the main RDBMSs. Oracle is much more robust, but costs a significant amount of money. MySQL is free, but is not nearly so robust.
- If technologies are standardized, then a code library could be built to be utilized by the current and future classes. Development of a repository and a few initial code blocks could form the project for a future team.

- A forum for knowledge sharing should be created and utilized. The reliance on standardized technology will ensure that this forum will remain relevant in the future and have greatest benefit.
- At least one more milestone for code delivery in each semester should be established so that the QA team has more time to evaluate systems. This will also allow the QA team to find problems early and thus reduce the time required to fix them. It should be emphasized to the class that this milestone is a "soft" milestone in which a fully working system is NOT expected. This is merely an opportunity for QA to see the system and hopefully uncover faults early.

The experience with the Genealogy system does support the assertion that starting testing early reduces the cost of fixing errors. When the system was scrapped and started anew, issues uncovered in early testing were generally not repeated. The inference is that when the second system was created, not only was the code more correct, but also that the *design* was more correct. The Absentee system also saw similar results. When the interface was redesigned, errors from the previous version were not repeated, even though some of those errors were specific to the interface and not to the underlying functionality. There were also no new errors introduced by the redesign, this was no small accomplishment for the implementation team!

Unfortunately, the small sample set that we are able to report on does not allow a scientific conclusion to be made regarding the overall impact of a separate QA team on the success of the other teams. In every case where the QA team was able to evaluate systems early, issues were uncovered by the QA team that the implementation team had not uncovered. This indicates that having separate people test did have a positive impact on overall quality. However, this alone does not justify a standalone QA team. This probably could be accomplished simply by requiring each implementation team to test another team's system.

It is felt by this QA team that this experiment in providing a separate QA team should be repeated in the next year's seminar sequence to see if lessons learned this year can allow the future team to provide a greater impact across all of the student projects. Even if the decision is made to not undertake a separate QA team for next year, it is felt that third party testing in the form of allowing teams to cross-test each others' applications is highly desirable. The results of this year clearly indicate to this team that any form of third party testing will yield much higher quality than none at all.

## **References**

*The American Heritage® Dictionary of the English Language, Fourth Edition*  
Houghton Mifflin Company 2000

Ambler, Scott W. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, John Wiley & Sons, 2002

Brooks, Frederick P., Jr. *The Mythical Man-Month Essays on Software Engineering, Anniversary Edition*, Addison Wesley 1995

Butler, Keith A. "Usability Engineering Turns 10", *Interactions*, volume 3, issue 1, ACM Press January 1996

Cockburn, Alistair *Agile Software Development*, Pearson Education, 2002

Gehrke, Matthias, et al. "Reporting about industrial strength software engineering courses for undergraduates", *Proceedings of the 24th international conference on Software engineering*, 2002, ACM Press, pp 395-405

Gibbs, N.E. "The SEI education program: the challenge of teaching future software engineers", *Communications of the ACM* May 1989 Volume 32 Issue 5 Pages: 594 - 605

Hilburn, Thomas B., Towhidnejad, Massood, "Software quality: a curriculum postscript ?", *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, ACM Press, United States, 2000, pp 167-171

Lynch, N. A., Merrit, M., Weihl, W. E., Fekete, A. *Atomic Transactions*, Morgan Kaufmann, 1993.

Pressman, Roger S. *Software Engineering: a Practitioner's Approach, 5<sup>th</sup> ed.*, The McGraw-Hill Companies, 2001

Rising, Linda "Removing the emphasis on coding in a course on software engineering", *Proceedings of the twentieth SIGCSE technical symposium on Computer science education*, 1989

Tappert, Charles "Students Develop Real-World Computer Information Systems", *CSIS Technical Report 182*, Pace University, October 2002

Wheeler, Sharon, Duggins, Sheryl, "Improving software quality", *Proceedings of the 36th annual*

*conference on Southeast regional conference*, ACM Press, 1998

## **Software of Note:**

*OpenSTA (Open System Testing Architecture)*:  
<http://www.opensta.org>

## **Contacting the Authors:**

The authors may be contacted regarding this paper at the following email addresses:

Jim Leonardo - [jim@jimleo.com](mailto:jim@jimleo.com)

Milo Auguste - [milo@theone.homedns.org](mailto:milo@theone.homedns.org)

Ritu Mehrotra - [rmehrot@gcr.com](mailto:rmehrot@gcr.com)