

## Quality Assurance and Maintenance Tools

Thomas N. McKee, Aditya Prakash Chandra, Jae Sohn, and  
Sanjukta Nayak

### Introduction

Employers say that IT graduates lack important skills needed in the work place, notable knowledge of current IT and various “soft” skills, including presentation, customer relations, leadership, and team work. [1] The major focus of Pace University’s CS615-616 Software Engineering Seminar is to have student teams 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. [2] During the course, one student team is selected to perform the function of an independent Quality Assurance team. This paper details the work performed by the Software Quality Assurance student team during the 2003-2004 timeframe. The work performed by this year’s team is built upon the work performed by the QA team during the previous year. Previous case studies were reviewed to ensure previous lessons learned were incorporated into the current projects.

The major focus of the 2003-2004 Software Quality Assurance student team was to provide quality assurance oversight to the other groups within the class. In addition, the quality assurance team tracked student project results via case studies and established a data system for tracking, monitoring, and recording user feedback for CS615-616 Software Engineering student projects (QA and maintenance tools project). The project consisted of three parts. The first part of the project was to generate a set of metrics by which student software projects can be monitored and judged for quality. The second portion of the project was to create a web-based database to store project information. The last portion of the project was to use the metrics to create interactive forms that can be used to judge project quality (this portion is scheduled for completion during the Spring 2004 semester).

Selecting a concise set of metrics is a challenge onto itself since the types and topics of each project vary greatly. In addition, each team’s small

size does not allow for designation of a primary QA team member. Therefore, the QA team had to become familiar with each of the other projects to project performance against each metric. Final software quality will be judged on customer satisfaction, project completion (as defined by the project requirement statement), established metric compliance, and timely milestone completion.

### Relevance

The goal of any software quality assurance activity should be to produce high quality software. However, before high quality software can be produced, software quality must be defined. This is not a simple task. Software is a diverse product. In addition, the development teams must often produce software in a condensed time period. Therefore, development teams may view software quality assurance as an added burden. Some software developers continue to believe that software quality is something you begin to worry about after the code has been generated. [3] The first step in ensuring a quality product is to convince the design team that there is an added benefit from quality assurance activities. Once the benefits of such a program have been internalized then the activities can be incorporated into seamlessly to help create an improved product.

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. [4]

### Methods

There are three main methods used by the CS615-616 Quality Assurance Team to perform their task.

Part 1: The first part of the project was to generate a set of metrics by which student software projects can be monitored and judged for quality. The metrics were approved and posted on the project web site. In addition, the project review and testing schedule must be established, accepted by all teams, and posted as a milestone.

Part 2: The second portion of the project entailed creation of a web-based database to store project information.

Part 3: The last portion of the project was to use the metrics to create interactive forms that can be used to judge project quality. Estimated completion date: end of the CS616 semester

### **Part 1 - CS615-616 Software Project Metrics:**

#### **1. General Project Design Criteria:**

1. Remote Database Management Systems such as Oracle, MS SQL Server, MySQL should be used in preference to local databases such as MS Access.
2. PHP or Cold Fusion should be used as a front end scripting language.

#### **2. What is 'good design'?**

'Design' could refer to many things, but often refers to 'functional design' or 'internal design'. Good internal design is indicated by software code whose overall structure is clear, understandable, easily modifiable, and maintainable; is robust with sufficient error handling and status logging capability; and works correctly when implemented. Good functional design is indicated by an application whose functionality can be traced back to customer and end-user requirements. For programs that have a user interface, it's often a good idea to assume that the end user will have little computer knowledge and may not read a user manual or even the on-line help; some common rules-of-thumb include:

- The program should act in a way that least surprises the user.
- The program should satisfy both the novice and the expert user.
- It should always be evident to the user what can be done next and how to exit the program.
- Any user input should be pre-filtered before being sent to the database to prevent errors on the backend.

- Shouldn't let the users do something stupid without warning them.

#### **3. What is 'good code'?**

'Good code' is code that works, is bug free, and is readable and maintainable. The following items are guidelines that may assist in good code development. These may or may not apply to every situation. 'Peer reviews', 'buddy checks' code analysis tools, etc. can be used to check for problems and enforce standards.

- Minimize or eliminate use of global variables.
- Use descriptive function and method names - use both upper and lower case, avoid abbreviations, use as many characters as necessary to be adequately descriptive (use of more than 20 characters is not out of line); be consistent in naming conventions.
- Use descriptive variable names - use both upper and lower case, avoid abbreviations, use as many characters as necessary to be adequately descriptive (use of more than 20 characters is not out of line); be consistent in naming conventions.
- If public classes are created, documentation of the public interfaces should be maintained separately from the internal developers' documentation.
- Function and method sizes should be minimized; less than 100 lines of code is good, less than 50 lines is preferable.
- Function descriptions should be clearly spelled out in comments preceding a function's code – not after.
- Organize code for readability.
- Use white-space generously - vertically and horizontally
- Each line of code should contain 70 characters max.
- One code statement per line.
- Coding style should be consistent throughout a program (e.g., use of brackets, indentations, naming conventions, etc.)
- In adding comments, err on the side of too many rather than too few comments; a common rule of thumb is that there should be at least as many lines of comments (including header blocks) as lines of code.
- No matter how small, an application should include documentation of the overall program function and flow (even a few paragraphs is better than nothing); or if possible a separate flow chart and detailed program documentation.
- Make extensive use of error handling procedures and status and error logging.

- For Java and C++, to minimize complexity and increase maintainability, avoid too many levels of inheritance in class hierarchies (relative to the size and complexity of the application). Minimize use of multiple inheritance, and minimize use of operator overloading (note that the Java programming language eliminates multiple inheritance and operator overloading.)
- For Java and C++, keep class methods small, less than 50 lines of code per method is preferable.
- For Java and C++, make liberal use of exception handlers

#### 4. What are five common solutions to software development problems?

- Solid requirements - clear, complete, detailed, cohesive, attainable, testable requirements that are agreed to by all players. Use prototypes to help nail down requirements.
- Realistic schedules - allow adequate time for planning, design, testing, bug fixing, re-testing, changes, and documentation; personnel should be able to complete the project without burning out.
- Adequate testing - start testing early on, re-test after fixes or changes, plan for adequate time for testing and bug fixing.
- Stick to initial requirements as much as possible - be prepared to defend against changes and additions once development has begun, and be prepared to explain consequences. If changes are necessary, they should be adequately reflected in related schedule changes. If possible, use rapid prototyping during the design phase so that the customers can see what to expect. This will provide them a higher comfort level with their requirements decisions and minimize changes later.
- Communication - require walkthroughs and inspections when appropriate; make extensive use of group communication tools - e-mail, GroupWare, networked bug-tracking tools and change management tools, intranet capabilities, etc.; insure that documentation is available and up-to-date - preferably electronic, not paper; promote teamwork and cooperation; use prototypes early on so that customers' expectations are clarified.

#### 5. Web Design Guidelines:

##### Text

- Background does not interrupt the text
- Text is big enough to read, but not too big
- The hierarchy of information is perfectly clear

- Columns of text are narrower than in a book to make reading easier on the screen

##### Navigation

- Websites that require a 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 buttons and bars are easy to understand and use
- Navigation is consistent throughout web site
- Navigation should not allow the user to get to a dead end page.
- Navigation buttons and bars provide the visitor with a clue as to where they are, what page of the site they are currently on
- Frames, if used, are not obtrusive
- A large site has an index or site map

##### Links

- Link colors coordinate with page colors
- Links are underlined so they are instantly clear to the visitor

##### Graphics

- Buttons are not overly big
- Every graphic has an alt label
- Every graphic link has a matching text link
- Graphics and backgrounds use browser-safe colors
- Animated graphics turn off by themselves

##### General Design

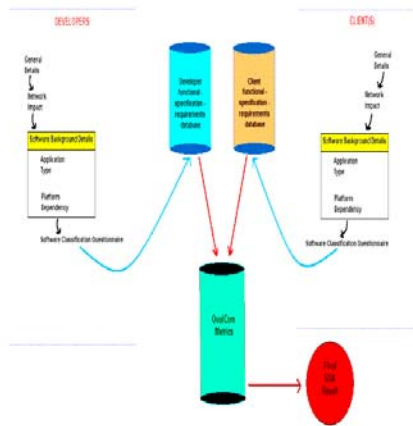
- Applications should be compatible and tested with more than one browser
- Pages download quickly
- First page and home page fit into 640 x 460 pixel space
- All of the other pages have the immediate visual impact within 640 x 460 pixels
- Good use of graphic elements (photos, subheads, pull quotes) to break up large areas of text
- Every web page in the site looks like it belongs to the same site; there are repetitive elements that carry throughout the pages

#### 6. Some general SQA checks:

- project is properly organized,
- development team members have defined tasks and responsibilities,
- documentation plans are implemented,
- documentation contains what it has to contain,
- documentation and coding standards are followed,

- standards and conventions are adhered to,
- metric data is collected and used to improve process and product,
- reviews and audits take place and are properly conducted,
- tests are specified and carried out,
- problems are recorded and tracked,
- projects use appropriate tools, techniques and methods,

### **Part 2 – The project information web based database**



### **Part 3 – Measuring project quality via online metric forms**

Project quality will be judged through the completion of online forms by both the development teams and the QA team. The data collected will be based upon the type of application being developed (i.e. web based software, stand alone application, database, etc.). Once the forms are completed, a quality score will be assigned to the project and stored in the project database. The customer can then complete an online form at predefined intervals. The customer quality score will also be assigned to the project and stored in the project database. The score delta will allow the quality of the metrics to be evaluated. In this way the metric set can be iteratively fine tuned during future project classes.

### **Results**

The QA team was able to accomplish their first two objectives. The team had also hoped to judge the other project’s preliminary project quality by comparing the prototype version presented to each team’s initial requirement statement. However, at the time of this paper preparation, the conformance of each

- software is stored in controlled libraries,
- software is stored safely and securely,
- software from external suppliers meets applicable standards,
- proper records are kept of all activities,
- staff members are properly trained,
- risks to the project are minimized.

project with the posted metrics could not be performed. All of the groups were able to develop early prototypes, but were not able to submit their projects to the QA team for preliminary evaluation. This appears to be a repeat occurrence of the 2002-2003 QA team,

*“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.”*

Three teams submitted copies of their requirements to the QA team for review. The results of the requirement statement review will be presented in the case study section below. Many of the projects have had significant changes to technology, scope, or focus. In addition, many teams have run into problems with getting strong specifications or other details from their customers. This appeared to hamper initial project development. The QA team also noted that many teams did not use the pre-production server to implement their projects. The apparent cause were problems with the available server environments.

### **Case Studies**

To show the effect of Quality Assurance, each 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 1: Team 1 – Pervasive Telemedicine System**

Requirement Statement:

*The telemedicine system for the Nursing Department at Pace University consists of three modules. The first module involves setting up of a video conferencing*

session of the patient at a remote site with a nurse at the Nursing department on Pace campus. The second module is to build a web based interface for the patient at a remote site to interact with the nurse. The third part involves the transmission of vital signs or digital data like Blood Pressure or Pulse, etc of the patient over the web based interface to the nurse in session.

Team 1 was able to complete a prototype at the end of the first semester. This prototype was demonstrated for the customer and a prospective client during the beginning of the second semester. The system did not appear to meet the needs of the customer. This may be due to lack of communication with the customer during the requirements and development phase of the project.

### **Case Study 2: Team 2 – Interactive Flag Recognition System**

The team requirements document was not available for review. The Interactive Flag Recognition team project was completed at the end of the first semester. A system prototype was completed and implemented using MatLab. The system was trained using a sample of flags and flag recognition was successfully demonstrated.

### **Case Study 3: Team 2a – Speech Processing System**

The team requirements document was not available for review. The Speech Processing team project will extend Pace CSIS graduate student's (Naresh Trilok) dissertation work as follows:

- *become familiar with Matlab and rerun the experiments to understand the system*
- *create a grey-scale plot of the 13 frequency bands as a function of time (this plot can be used to display the speech samples before and after segmentation, and also after dividing the segmented samples in the seven speech sounds)*
- *run a 24 feature experiment (12 frequency bands without the first Cepstral component, means and variances)*
- *double the database of speech samples from 10 speakers to 20 speakers, 10 samples each of "My name is (person's name)."*
- *automate the segmentation of the "My name is" portion of the speech samples*
- *use the signal energy (first spectral component) to locate the start of "My name is"*
- *use the ratio of the sum of the spectral components over 2KHz to the sum of those under 2KHz to locate the [z] sound and thus the end of "My name is" (it should also be approximately one second*

*after the start since this phrase is about one second in duration)*

- *automate the segmentation of the "My name is" phrase into its 7 speech sounds*
- *use the elastic matching (dynamic time warping) algorithm to align with a pre-segmented speech sample*
- *rerun the 24 and 84 feature experiments with the increased database of speech samples from 20 speakers*
- *run a 168 feature experiment (both means and variances of the 84 measures)*

### **Case Study 4: Team 3 - Speech Impaired System**

Requirement Statement:

*This project involves the design and construction of a system to facilitate communication with the speech impaired, and in particular with those who can neither hear nor speak. The goal of this system is to enable communication between two people through the combination of current technologies that allow the users to input and output words through common applications such as text documents and voice synthesizer software. The system will use a short hand script to enable the user to input commands faster that will be created by the user Bill Huber.*

Team 3 spent a considerable amount of time deciding which portion of the project they would implement during the first semester. Their presentation detailed the issues for which they selected their text to sound system. In addition, 2 prototypes (Visual Basic and Java software versions) were sent to the QA team for evaluation. Both software versions were able to translate a text passage (or freeform text) to speech sound. However, each of the prototypes appeared to have limitations. The Java application required extra files to be downloaded prior to use, then could only be run once compiled from with a Java environment. The visual basic prototype also functioned to produce speech sound from text. However, it only functioned on Windows 2000 or Windows XP operating systems.

### **Case Study 5: Team 5 – XML Teaching System**

Requirement Statement:

- *To have an online repository for students to learn basics and advanced features of XML.*
- *In addition to above, To have an online xml repository where student and professor can work interactively.*

- To have an end-to-end open source solution.
- To use maximum possible features of an open source IDE for XML.(IBM's Eclipse IDE)

Team 5 was able to complete a prototype at the end of the first semester. However, the initial prototype did not appear to meet the needs of the customer. Team 5 has contacted the customer this semester and supplied a copy of the requirements document, design document, and software.

### Case Study 6: Team 6 – Faculty Info System

Requirement Statement:

*The main objective of this project is to create a system - using Fusebox and Cold Fusion for the web interface and Microsoft SQL Server for backend- that will add new functionalities to the existing web sites of Pace University. This system will facilitate:*

### Case Study 7: Team 7 – Electronic Medical Forms

Requirement Statement:

*The specific aims of the current project are:*

1. To create a new database, using MySQL (or a similarly robust system), for clinical research data.
2. To create a web-based front end through which data can be entered and from which investigators can access the data.
3. To write an application for the transfer of data from the "Medical Forms System" to the new database.
4. To import the data from the old database to the new system.

Team 7 appeared to on the right track for successful completion of their project. Their prototype included a demonstration of a preliminary database using MySQL with a web-based PHP implemented front end. The team stated that they had successfully imported data tables from the legacy database into the new database, although specific test results were not presented. It appears that one of the keys to team 7's success was direct communication with the client (who was also the team leader). This success may be attributed to an informal use of the FAST approach to requirements gathering. Roger S. Prussman states in his text, Software Engineering, A Practitioner's Approach, "Customers and software engineers often have an unconscious "us and them" mindset. Misunderstandings abound, important information is

- Prospective applicants to apply online for the faculty position at Pace University;
- Applicants to scan certain documents (e.g. their Degrees) and upload them onto the database. So that these documents can be viewed and evaluated by the reviewer;
- Certain faculty/admin users to log on, view, and categorize prospective applicants as well as evaluate their applications;
- Current faculty to e-mail the applicants through the system and have all correspondence logged by the system.

Team 6 appeared to on the right track for successful completion of their project. They presented a working version of the system. The demonstrated system appeared to meet the requirements proposed by the customer. In addition, the team adhered to the design requirements established by the Quality Assurance team for a web based database application. It appears that one of the keys to team 6's success was continuous communication with the client.

omitted, and a successful working relationship is never established." FAST – facilitated application specification techniques encourages the development of a joint team of customers and developers who work together to define and solve the problem, thus eliminating the "us versus them" mentality.

Recently team 7 stated that they had found a flaw in their database during testing. This semester, they plan to rebuild the database aiming to meet 90% of the requirement.

### Case Study 8: Team 8 – Weather Station Website

The team requirements document was not available for review. This is a new project for this semester. Team members have problem in gathering requirements for their project because their customers are not sure of what they want. In Software Engineering Book, Gause and Weinberg suggest that the analyst start by asking context-free questions. These questions will lead to a basic understanding of the problem and will be very effective in first encounter between customer and Software engineer. It may also be possible the client knows what they want but are not able to express their requirements.

### Conclusions

The QA team was successful in implementing their project goals. However, the team was not successful in improving the quality of the other team projects. Many teams struggled to complete a working

prototype of their project, prepare a project presentation, and complete their project binder in a timely manner. Many teams did not utilize the QA team as a tool to help improve their quality. One team failed to submit their requirements statement to the QA team for evaluation. The apparent causes appeared to be schedule pressure and lack of communication with the project customers.

Some recommendations for future classes include:

1. Set project milestones to perform a mock Formal Technical Review meeting for each project.
2. Issue the software metrics at the beginning of the project.
3. 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.
4. 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.

## **References**

1. Peter J. Denning and Robert Dunham, *The Core of the Third Wave Professional, The Profession of IT*, ACM, Vol. 44 No. 11, Nov. 2001
2. Charles Tappert, “Students Develop Real-World Computer Information Systems”, *CSIS Technical Report 182*, Pace University, October 2002
3. Sharon Wheeler and Sheryl Duggins, “Improving software quality”, *Proceedings of the 36th annual conference on Southeast regional conference*, ACM Press, 1998
4. Roger Pressman, *Software Engineering*, 5th Ed., McGraw, 2001.
5. Fred Brooks, *The Mythical Man Month*, Addison, 1995.
6. Henry, S., & Kafura, D., *The evaluation of software systems structure using quantitative software metrics*, Software Practice and Experience, 1984.
7. Robin Williams, *Good Design Features*, <http://graphicdesign.about.com/gi/dynamic/offsite.htm?site=http%3A%2F%2Fwww.ratz.com%2Ffeaturesgood.html>
8. R.S. Pressman & Associates, Inc., *Adaptable Process Model Document Templates*, 2001, <http://www.rspa.com/apm/index.html>
9. M. G. Mendonça, V. R. Basili, I. S. Bhandari, and J. Dawson, *An approach to improving existing measurement frameworks*, IBM System Journal, Volume 37, Number 4, 1998, <http://www.research.ibm.com/journal/sj/374/mendonca.html>
10. Edward Kenworthy, *Use Case Modeling, Capturing User Requirements*, 1997, [http://www.zoo.co.uk/~z0001039/PracGuides/pg\\_use\\_cases.htm](http://www.zoo.co.uk/~z0001039/PracGuides/pg_use_cases.htm)
11. Jack W. Reeves, *What is Software Design*, C++ Journal, 1992, <http://www.bleading-edge.com/Publications/C++Journal/Cpjour2.htm>
12. Fagan, M.E., *Design and code inspections to reduce errors in program development*. *IBM Systems Journal*, 15(3), 182-211, 1976.
13. Jim Leonardo, Ritu Mehrotra, Milo Auguste Jr., *Providing a Separate Quality Assurance Team for a Project-Oriented Software Engineering Seminar*, Software Engineering Seminar CS615/CS616, 2002-2003.
14. Oracle Technology Network page, <http://otn.oracle.com/>, accessed April 2004.
15. php Home Page, <http://www.php.net/>, accessed April 2004.
16. OpenSTA Users Home Page, <http://www.OpenSTA.org/>, accessed April 2004.
17. Mozilla Home Page, <http://www.mozilla.org/>, accessed April 2004.
18. MySQL Home Page, <http://www.mysql.com/>, accessed April 2004.