

Integrating Quality Assurance into the Software Development Life Cycle

Leslie Tierstein, STR LLC and Hilary Benoit, W R Systems, Ltd.

Introduction

The necessity of producing a quality software product in a repeatable and consistent way is indisputable. But how do you integrate the Quality Assurance (QA) function into the Software Development Life Cycle (SDLC) in a way that is effective and efficient, acceptable to both customers and developers, and that doesn't adversely affect the project budget and schedule? Appropriate checks and balances need to be built into the development life cycle. Sometimes, these are such obvious activities as QA audits and reviews. At other times, they are so discretely built into the life cycle that they are accepted as a normal development activity by the entire project team.

This paper documents the experience of integrating quality assurance activities into two full life cycle custom development projects. It describes the software tools, procedures, activities, and methodologies used on Oracle Designer and Developer projects in a web deployment environment but which are applicable, with minor modifications, to other development tools and environments. Real-life examples and lessons learned illustrate what worked and what didn't.

The SEI Capability Maturity Model (CMM), CMM-Integration (CMMI) and QA

At the request of the federal government, and with the assistance of the MITRE Corporation, the Carnegie Mellon Software Engineering Institute (SEI) developed a model and questionnaire to gauge the maturity of the software development process. The SEI's Capability Maturity Model (CMM) provides guidelines against which an organization is able to measure its own software process capability. The CMM is a software process framework for developing software through Key Practices Areas (KPAs) that enable organizations to improve their ability to meet goals of cost, schedule, and quality.

As the CMM was gaining exposure and acceptance in the US, an alternate set of QA-oriented standards, the ISO-9001, was being promulgated by the International Organization for Standardization (ISO). ISO-9001 is a set of standards for quality management systems for projects of all types, incorporating but not tailored for software engineering. Possibly as a result of influence by ISO, SEI recently developed Capability Maturity Model Integration (CMMI). The CMMI is a framework intended to integrate the many models currently in existence worldwide, including ISO-9001, not limited to the disciplines of systems engineering and software engineering. An organization is able to migrate easily from the CMM to CMMI. Comprised of five "maturity levels", the CMM is organized into KPAs as follows. (The CMM KPAs are cross-referenced to the CMMI model in order to show their relevance to CMMI.):

CMM		CMMI	
Maturity Level/ Process Description	KPAs	Maturity Level/ Process Description	KPAs
1. Initial Ad hoc ("Chaotic")	None	1. Initial Ad hoc ("Chaotic")	<ul style="list-style-type: none"> None
2. Repeatable Disciplined	<ul style="list-style-type: none"> Software Configuration Management Software Quality Assurance Software Subcontractor Management Software Project Tracking and Oversight Software Project Planning Requirements Management 	2. Managed Planned, performed, managed, controlled	<ul style="list-style-type: none"> Configuration Management Process and Product Quality Assurance Supplier Agreement Management Project Monitoring and Control Project Planning Requirements Management
3. Defined Standard Consistent	<ul style="list-style-type: none"> Peer Reviews Inter-group Coordination Software Product Engineering Integrated Software Management Training Program Organization Process Definition Organization Process Focus 	3. Defined Consistent across organization	<ul style="list-style-type: none"> Verification Integrated Project Management Requirements Development Technical Solution Product Integration Validation Verification Organizational Training Organizational Process Definition Organizational Process Focus
4. Managed Predictable	<ul style="list-style-type: none"> Software Quality Management Quantitative Process Management 	4. Quantitatively Managed Predictable	<ul style="list-style-type: none"> Quantitative Project Management Organizational Process Performance
5. Optimizing Continuously Improving	<ul style="list-style-type: none"> Process Change Management Technology Change Management Defect Prevention 	5. Optimizing Continuously Improving	<ul style="list-style-type: none"> Causal Analysis and Resolution Organizational Innovation and Deployment

Achieving at least a CMM level 2 is becoming more critical for both Government and commercial contracts. To have at least attained a level 2 indicates that your organization has in place some basic, and essential, processes for undertaking a software development project and that you can assure your customer that the potential for delivering a quality product or service, within budget and schedule, is good.

Start at the Beginning - Define and Document the Software Development Process

As a foundation to the QA process, you need to start with a well-defined Software Development process with clear life cycle phases, and entry and exit criteria, in order to establish QA procedures appropriate to each phase. It doesn't have to be complicated, but it does have to be clear, workable, and logical. If you want or need the added bonus of CMM compliance, you should ensure that your software process includes all of the KPAs important to the CMM.

CMM level 2 compliance was a requirement of the contract to develop the initial project. The contractor (W R Systems) defined and documented a comprehensive software development process that covered the SDLC. Well-defined life-cycle phases with documented entry and exit criteria, and quality checkpoints and reviews meant that QA would be essentially built into the process and that the QA function would have a good starting point for its activities. The defined life cycle phases were:

Strategy

Analysis

Design

Build and Test

Deployment

Maintenance

Persuasion and The Cost of Quality

Unless upper management already supports the idea of Software Quality Assurance (SQA) and a QA function, it may be necessary to prepare the ground through presentations and training. For example, likely topics might include:

- Cost of Quality (COQ) categories and their associated SQA activities
- Overview of the SEI CMM

A requirement of the SEI CMM – level 2 is that there exists an individual or group responsible for SQA on a project.

Set up the QA Function

Once the need for a QA function has been established, the question is: how do you staff it? One solution is to take developers and give them a temporary stint in Quality Assurance. This, apparently, is often cited as an ideal method of ensuring that QA personnel have a technical background and also of ensuring that eventually all your development staff understand and carry the torch for all those QA ideals. However, this method rarely works in real life. It's a rare developer who will give up a chance to be "creative" in order to endure a period in the dreaded QA group!

For our QA group, we found it easier to take extremely smart people who displayed excellent analytical skills, paid attention to detail, were persistent, and able to concentrate for long periods of time on the kinds of things that would drive lesser mortals crazy. In addition, they needed to be computer-literate (of course), preferably with a background in Software Development, fast learners, self-starters, and have - or develop - thick skins! Communication skills, written and verbal, were essential, as was an excellent command of the English language since one of the problems of working in QA is that everyone expects your work to be perfect, and that means no spelling or grammatical errors in your documentation! The QA manager on our project came with 15-year background in software engineering and saw moving into Quality Assurance as both a challenge and the opportunity to learn a new area of software engineering. It's also important to remember that tools and training for QA should ideally mirror those of the development team - development tools, workstations etc.

Select the Pilot Project

Our next step was to select a suitable pilot project – in this case, a full-life-cycle development project that would use the Oracle database and the Designer/Developer tool set, and which was planned to last five years - plenty of time in which to establish an SQA function from the ground up. The project was to design and develop a logistics system that would support the Coast Guard's supply replenishment and financial functions.

Integrate the QA function into the Life Cycle Phases

It was important to define the perceived purpose and goals of the QA function. In other words, what did we want QA to accomplish? The software engineering process was already defined with checkpoints such as end-of-phase reviews, so wouldn't the QA function be, in a sense, redundant? Not really. Our goal was that QA would help to ensure that the software development process was being complied with through ongoing QA audits, reviews, and management notification. Having a specialized function to provide management with visibility of project processes meant that the development team members were freed up to concentrate on what they did best. In addition, project management personnel had a dedicated extra set of eyes and ears ready to alert them to problems and deficiencies early on in the project life cycle - or at a level where they wouldn't normally be noticed until the impact was much greater.

Using a documented software engineering process that required specific inputs and outputs (entry and exit criteria) for each life cycle phase, QA was able to develop a checklist of deliverables to review prior to the next phase being started. Through consistent QA auditing of processes and products, it was expected that a high level of Customer confidence would be maintained.

Strategy Phase

The QA manager was assigned at the very beginning of the project - even before most of the development personnel came on board. She had an extensive software development background that enabled her to be particularly effective and to work closely with the Technical Manager throughout the project.

Develop the QA Plan and Procedures

The first assignment for the QA manager was to write the project's QA Plan. This had to be integrated with the Project Plan, comply with W R Systems' software development methodology, and specifically address the development platform and tools to be used. The section on SQA Procedures, Tools, and Records was by far the largest section of the plan, with 49 procedures addressing more than 70 project deliverables.

The QA Plan had to be specific and to the point. Risk areas were identified and a real-life schedule of QA audits and reviews was created. It was important to identify how testing would be accomplished, and to describe specific QA procedures that would be applicable to the many stages and activities of the project. QA developed a cross-reference that mapped each procedure to the Military Standard (MIL-STD)-498 - *Software Development and Documentation* and to the ISO 9000 series of standards for quality systems. The MIL-STD series provides a set of standards to be used on Government contracts. At the time this project began, compliance to MIL-STD-498 was a requirement of the contract. The ISO 9000 series of standards was published in 1987 by the International Organization for Standardization and has since been adopted by many countries. It is rapidly replacing prior national and industry-based standards. The customer was delighted with the cross-reference. The main activities of the QA function would be to:

- Perform QA audits and reviews of all project processes and deliverables
- Create QA records of every audit and review
- Notify project management of non-compliance with standards and procedures, or of any major problem noticed by QA
- Verify that corrective action has been performed and that Problem Reports have been resolved
- Manage the Requirements Traceability process
- Attend project meetings
- Provide necessary training in standards, testing, or other QA-related topics
- Provide final sign-off for testing

Maintain Quality Records

Documenting the results of QA audits and reviews is extremely important in establishing an audit trail of QA activity, compliance with SEI CMM, and follow-up on corrective action. Every activity that QA performed was documented on a pre-printed form specific to the particular procedure. The form was completed on-line as a Word document and then printed for storage in the Project Notebook, and for distribution to Project and Corporate Management. The QA records were maintained both as hard copies and electronic files, and were available for review by the Customer at any time. They became a valuable tool in maintaining an audit trail of deliverable reviews and were referenced many times in order to settle any confusion over what was reviewed, by whom, and when. A QA Review Form is given as Figure 1 in the Appendix to this paper.

Keeping Metrics on QA Activities

The QA records satisfy the requirement to document QA activity. However, we decided to take this a step further and find some method of easily producing reports and queries on the metrics we were keeping; for example, the results of audits and reviews per subsystem or deliverable, defect types, density of defects, frequency of certain types of QA activities, etc. Using Microsoft Access, QA decided to develop a simple database application with data entry forms and reports. The application was called *Quality Assurance Tracking System (QATS)*. Data from QA reviews and audits were entered on a regular basis, providing a good basis for metrics reporting throughout the project, such as the type and number of problems or defects encountered in QA audits and reviews. This information was particularly effective in identifying areas for improvement or corrective action. For example, during the early phases of the project, we found that the majority of defects overall appeared to be in the areas of documentation and non-conformance to standards. Monthly reports also provided listings of all the QA activities for the project - something that's always useful when trying to get a picture of how QA is providing value-added project support.

Reviewing the Requirements

One of the most critical elements of a successful project is the quality of the requirements. Time spent in reviewing the requirements at the beginning of the project will pay dividends as the project progresses. During the strategy phase of the project, the requirements were carefully reviewed by the QA manager, who defined subgroups and unique requirement IDs that identified the subsystems and functional subgroups to which each requirement belonged. The emphasis for QA was on reviewing and organizing the functional requirements (170 of them) into the functional categories that would be associated with the various application subsystems such as Supply and Finance. Once the requirements were organized, QA reviewed them for clarity, redundancy, completeness, and testability. This was the first step in the creation of a Requirements Traceability Matrix (RTM) that would become one of the project's primary QA tools for helping project management to keep the development process on track.

Specific features were associated with the requirements:

- Each requirement had to be specific enough to be testable and had to specify what needed to be done, not how to do it.
- Each requirement had to be uniquely identified so that it could later be traced to functions, modules, code, and tests.
- QA would ultimately verify that each requirement was implemented in the finished application, and that every feature of the finished application corresponded to a requirement.
- Defects in an application would be categorized as missing requirements, wrong functionality, and extra features (i.e., not corresponding to a requirement). A requirement not implemented is a failure to provide the agreed-upon system.
- Functionality with no corresponding requirement is an error in scope ("creeping featurism" or "gold-plating") that may result in an increase of schedule and budget.

At this stage, we were able to start to draft the test plan and design basic functional tests. The requirement ID codes were used in generating reports and in tracking test case coverage.

Develop the Requirements Traceability Matrix (RTM)

The RTM is a vital part of QA's toolkit. It allows QA to monitor the progress of requirement coverage throughout the project and ensure compliance to customer requirements of the software and its deliverables. Since it's been found that a large percentage of software defects and failures (almost 60%) can be traced to lack of traceability to requirements (ref. "National Software Quality Experiment 1992 – 1998"), we decided that the development and maintenance of the RTM, a key requirement of the company's software development process, would be formalized, and that the QA manager would take overall responsibility for its development. Finding a repository for the requirements was the first step. It was necessary to maintain the customer requirements electronically, both to more easily maintain the inevitable changes over the project life cycle, and to be able to print out reports and trace a history of changes to requirements. We chose to implement the RTM through the Oracle database and by extending the Oracle Repository, but it can also be implemented, as we have done on subsequent projects, in other ways, such as with an Access database, Excel spreadsheet, or Word document. The important thing to remember is that a formal and consistent method of tracing the development products to the requirements is essential.

The RTM report, and its variants, was so successful that updated versions became a regular deliverable. From the RTM, QA was able to produce reports for determining where links were missing – for example, if a requirement was not mapped to a function, or a function had no corresponding requirement. This provided visibility into the percentage of coverage. At this stage, one of the RTM reports, the Requirements-Functions Cross-Reference Report, allowed QA to monitor the completeness of requirements coverage from the earliest stages of analysis. The RTM was pivotal to the testing process at every stage of the life cycle from Analysis through Build and Test. This was probably the single most important tool for the QA Manager to determine coverage of user requirements during testing.

Oracle Designer's Application Design Transformer (ADT) automatically generates modules based on function definitions. The repository maintains the association between the module and the source function. If a module was not generated based on a function, the developer had to manually enter the corresponding function name, and, if required, define the function as well. QA would routinely check on the completeness of the information, by running the report and would notify the Project and Technical Managers of any missing data in the RTM that could be attributed to data entry falling behind. Traceability of functional requirements was thus achieved.

The QA manager became the custodian of the Functional Requirements List (FRL). All requests to change the list – initiated either by users or by the developers - had to be approved through a formal Configuration Control Board (CCB). The requirements change management procedure was developed by QA to ensure that there was the essential control while retaining the flexibility for the customer to make modifications as required. The request for a change to a requirement was submitted via a pre-printed form to QA, who then reviewed it for completeness and clarity before submitting it to the Configuration Manager (CM) who called the CCB meeting. Approved changes to requirements were entered into the RTM database by QA who reprinted the Functional Requirements List (FRL) with the most current requirements. This was distributed to team members. The tight control over the requirements proved invaluable over the three-year project cycle. A history of changes could be easily printed as a report from the RTM database at any time, as well as an up-to-date FRL.

A sample of the FRL is given in Figure 2 in the Appendix.

Establish the Deliverable Review Process

Independent review of a deliverable product is a Quality Control activity that can catch defects that the producer has missed. Every deliverable on this project was to be put through a formal review procedure including a Technical Review followed by a QA Review. Once it was clearly established that all deliverables would be put through a review process, various review route sheets were developed that would accompany each deliverable on its journey from the developer to the technical review, to the QA review, and finally to the customer. The route sheets were checked off and signed by each reviewer once the review was considered complete and the product satisfactory. The Technical Review, generally performed by the Technical Manager, would ensure that the product was technically correct and effective. QA would then review for compliance with standards, procedures, completeness, and overall professionalism. Once all the project

members were instructed in the review process and underwent it a couple of times, it became second nature. This was one of the more successful QA procedures and was of course, particularly necessary during the analysis and design phases where the greater portion of the deliverables consisted of design documents and plans. A sample Project Deliverable Route Sheet is given in Figure 3 of the Appendix.

Create the Project Standards and Procedures

Project standards need to be set up during the early stages of the project. The Technical Manager wrote an extensive and comprehensive manual of Oracle development standards that covered everything from naming standards to SQL coding standards. The standards were developed after consulting published works including Steven Feuerstein's PL/SQL books Dorsey and Koletzke *Oracle Designer Handbook*, and *Designer Standards*, by Kramm, et al. In addition, non-Oracle specific standards governing user interface design were also consulted. However, many standards, particularly those specifying the user interface, needed to be project-specific and developed in-house.

QA reviewed and contributed to the standards, and the project team was trained in the use of the standards document, which became the “bible” of Oracle development projects that followed. Looking back, the creation of such extensive and detailed standards was one of the most useful activities that was undertaken to ensure consistent, high quality development. As new people joined the team, QA was able to give them this Oracle Standards document and train them in the requirements of the project. Having the standards also made the QA reviews and subsequent Peer Reviews more effective and objective. QA should always review and audit against a standard and not use subjective opinion. The project standards manual specified approved entries for every object type used by Oracle Designer and Developer. It contained a complete set of standards for:

- Object names
- The format for labels, hints, and help text
- Coding standards, including not only syntax, style, and naming conventions, but standard library calls, and use of server-side vs. client-side code
- Screen and report look and feel

Analysis Phase

Begin the QA Reviews and Audits

The analysis phase was the start of a period of ongoing auditing and reviewing by QA for verification of compliance with requirements, standards, and procedures. The effort spent in these activities at this early stage was extremely worthwhile since defects at the early stages of the life cycle are much cheaper to find and fix than those detected during the later phases of development and testing.

The deliverables from this phase were:

- Final Requirements Document
- Function Hierarchy
- Traceability Matrix
- Entity Relationship Diagram(s) (ERD)
- Process Model

Although QA found that automated Oracle Quality Control reports were useful during this period, nothing was a substitute for old-fashioned detailed desk checking against the standards. Developers used the Designer tools, e.g., ER Diagrammer and the Function Hierarchy Diagrammer to produce the deliverables. QA performed detailed reviews of hard copy deliverables against the actual soft copies in Designer, and ensured that standards were followed in the appearance and content of diagrams.

Design Phase

The design phase proved to be the busiest phase for QA on this project. There were many deliverables developed during this phase and Oracle Designer provided substantial reports that were considered customer deliverables:

- Physical Data Base Design
- Data Dictionary (Metadata repository)
- Module Network (Menu) Hierarchy
- Updated RTM
- Prototype Model(s)/Demonstrations

Quite often, we found ourselves writing custom reports on the contents on the Metadata repository. Our project DBA learned the Designer views and wrote these reports in a format that was both “tree-friendly” and user-friendly. During this phase, QA actually performed what are strictly speaking quality “control” activities such as detailed reviewing of the report output to ensure that table development standards had been followed, such as the ordering of columns where Primary and Foreign keys were required to be listed first. Project Naming standards were strict and QA developed several SQL queries to help detect non-conforming names. While it often seemed that Quality Control activities run by the QA function were predominant during this phase of the project, as the project progressed, more Quality Assurance activities run by developers were gradually introduced into the development process, such as Peer Reviews and Code Walkthroughs.

Keep the Customer in the Picture

Frequent contact with the customer was essential in providing visibility on the project and in ensuring that the software was developed in tune with the customer’s evolving needs. As the project developed, and as more and more analysis was performed with regard to the functionality of the legacy system and of the proposed system, customer needs became more defined and were often changed extensively, in accordance with the project’s Requirements’ Change Procedure.

Design Reviews

Design Reviews with the customer are quality checkpoints in the SDLC. It’s an opportunity to elicit formal customer feedback on the application design. The QA manager attended all design reviews and took detailed notes as a form of verification that all action items were documented and subsequently resolved. Action Items from these meetings were entered into the Problem Report/Action Item Tracking Database - an Oracle database that was developed by the CM manager as a repository for problem reports and action items. The QA manager periodically reviewed the database to ensure that corrective action was taking place and that all action items were addressed. Supporting documentation and deliverables for the design reviews were put through technical and QA reviews prior to being delivered to the customer in time for the meetings.

Build and Test Phase

Re-address Configuration Management (CM)

Until version 6i, Oracle Designer’s version control was virtually non-existent. Even in the 6i release, versioning has been subject to some difficulties. Consequently a strategy to use an alternate configuration management approach was developed. As the Build and Test phase progressed, we decided to use Merant’s PVCS tools to manage the version control of all source code and documentation.

Peer Reviews and Code Walkthroughs

The Build phase is an appropriate stage to introduce Peer Reviews and Code Walkthroughs as team activities. Both are extremely effective as QA processes and help to remind the developers that quality is their responsibility. The QA manager attended these activities on an ad-hoc basis, essentially to verify that they were being performed correctly and that they were documented. Participants in these activities were trained to perform their assigned roles as reviewers, facilitators, recorders, and producers. Items for review were distributed to the participants in sufficient time prior to the

activity to allow at least an hour of review time. The peer review or code walkthrough would generally last about an hour; any longer proved ineffective.

A Peer Review Form is given in Figure 4 in the Appendix.

Develop Prototypes

As the development process progresses, it is important to keep the customer informed on the progress of the application. We found that prototyping specific functional areas at regular intervals served to both assure the customer and to verify that we were still “in synch” with customer requirements. We held frequent meetings with the customer, on a bi-weekly basis, to demonstrate the completed modules in what were called “Prototype Demos”. These meetings were extremely successful in maintaining a high level of visibility between customer and developer and helped to iron out most remaining problems concerning functionality and design. The initial stage of the Build phase actually overlapped with the final phase of Design since this was considered an effective method of showing the customer what the system would look like in real life. Each prototype was packaged with the software modules, supporting user documentation, which was written in parallel with the software development, and various Oracle-generated reports; and associated Requirement Traceability reports. QA designed a route sheet that itemized all the required objects that belonged to a specific prototype, as shown in Figure 5.

Every object was put through the Review Process with final sign-off by the QA Manager. Action Items that came out of the Prototype meetings were entered into the project’s Problem Reporting/Action Item database. QA would periodically verify that action items from specific prototypes had been resolved and closed out. If not, a Management Notification Report would be written up and distributed.

Unit Testing

Unit Testing is traditionally the domain of the developer. After all, every developer has a vested interest in making sure that his/her code is high quality and does what it’s supposed to. At first, QA resisted participating in unit testing other than to help to develop the process, procedures, standards, and artifacts. As a result, unit testing was very informal. After the first prototype demos, however, it became apparent that informal unit testing was not being performed in the most consistent manner. Tight schedules often squeezed out the unit test stages and developers weren’t trained in testing and test writing. The developers would often test their modules to prove that they worked and spent little or no time on negative test cases that would find the bugs. With a large number of modules to be developed and tested, and with integration testing coming up, it became imperative to get tighter control over the test process. We addressed this by having the QA manager develop and teach a basic Test Writing course for the developers, many of whom were inexperienced in testing. The emphasis of the course was on building effective test cases that would be designed to find bugs. The test quality improved noticeably after this training. Having identified unit testing and later, bug fix testing as areas of weakness, the QA manager initiated a more formal method of unit testing using documented unit tests, peer reviews, and with the CM manager and QA manager more tightly integrated with the process. QA performed independent testing on an ad-hoc basis to ensure that the tests were fully exercising the module’s functionality and that test results were correct. A test form, Figure 5 in the Appendix, was developed by QA to serve double duty as the test script, the test report, and signoff form.

The new-style unit test required signoff by both a peer reviewer and QA. Only then could the module be considered to have been successfully unit tested. The completed test forms were kept in the Software Development Folders (SDFs) by the developers. QA kept copies and used them to develop statistics such as the number of bugs found during testing, and the percentage of tests that were failed by QA or by the Peer Reviews.

Integration Testing

Decision to Automate

Compuware's test management tool *QA/Director* was selected as a tool that would allow tests to be maintained in a test repository - essential for regression testing - and which would track bugs/problem reports, and execute tests and test reports. QA configured and installed the test tool and trained developers on how to write unit tests and enter them into the test repository. We found that developers were far more likely to develop good tests when using this tool. Test scripts could be printed out to supplement test plans and other test documentation; Test cycles could be set up that included entire test suites within subsystems of the application under test; Bugs could be entered into the Bug Tracking repository and assigned to specific developers for action. QA was able to go into the tool, run tests, review test results, and monitor bug fix progress. Compuware's *QA/Director* enabled us to provide structure to our test process although testing remained essentially a manual process since the testers still ran the tests step-by-step, rather than by running automated scripts.

Developing the Tests

Test development was accomplished in two phases.

1. **Writing the high-level test descriptions.** Extreme Programming (XP) methodology includes collecting "User stories" from the customer. One or more testers then assists the customer in defining and automating the tests. We used essentially the same methodology: user business scenarios, collected from the customer, were used as a starting point for determining interaction and integration between modules. QA, with input from the customer, worked the stories, or "scenarios" into basic test descriptions that encompassed the integration of the associated modules. With over 700 modules and a complex functionality among six subsystems, this was possibly one of the more difficult stages of testing and required a great deal of interaction between functional users, developer team leaders, and QA. The need to integrate the software with several external systems also needed to be addressed.
2. **Writing the test scripts.** Using *QA/Director*, the developers then wrote the test scripts from the test descriptions. We monitored the test writing progress and coverage through the various graphing and reporting features of *QA/Director*. QA continued to monitor test coverage through the RTM.

Managing the integration test effort

A test team was formed consisting of selected developers from each subsystem, and a senior developer became the Test Manager. In order to maintain independence and be able to monitor the test process, the QA manager did not participate in actual test execution. From time to time, QA would perform random independent testing or would monitor actual test execution by the testers. This proved to be a successful method, particularly at times when a pressing schedule required speedy execution of tests. Test meetings were conducted daily to discuss test schedule, progress, and problems. QA was the final sign-off on the successful completion of test cycles and all defects found during testing were logged into the bug-tracking database in *QA/Director* for resolution by the developers.

System Testing

The project used "requirements-based testing" which meant that we needed to verify that all the requirements previously specified were actually supported. Tests to determine compliance with the requirements were therefore written with the requirements in mind. All test scripts and test scenarios had to specify the requirement(s) being tested. If any additional test scripts were written, or existing scripts updated, requirement references had to be included. In order to facilitate the development of the tests, QA designed a Test Scenario form that enabled the test writers to include pertinent data for their tests. A sample of this form is given as Figure 7 in the Appendix.

We used Compuware's *LoadRunner* to assist with Performance and Load Testing. This allowed us to perform volume and peak load testing before the production hardware was available.

Deployment Phase

In order to facilitate deployment and cause as little disruption as possible to the users, we undertook a phased deployment, releasing individual subsystems several months apart. Our main challenge was to maintain two environments running in parallel – the development/test environment and the production environment. Strict CM control of source code in both environments was vital at this stage, as was ensuring that any modified code was kept in synch. QA review and test procedures continued as normal in the development environment. In the production environment, however, a faster turnaround was required. To facilitate this, problem reports and change requests from the users were called in or faxed, and the QA and CM functions worked closely together to expedite the test and delivery of modified code. User training took place prior to implementation, and QA worked closely with the trainers to develop and test all training materials.

Maintenance Phase

Once the system was deployed, the process of ongoing “tweaking” continued as the users became more familiar with the new application. The QA and CM procedures that were put in place during development were kept in use once the system was deployed. The source code remained under configuration management and any change requests from the user community were processed according to the same CCB review process as during development.

Process Improvement

As the project developed and SQA became more accepted as an integral part of the software process, there became more opportunity for process improvement activities where project team members could meet, take an existing process and analyze it, step by step, to identify weaknesses in the process, and brainstorm for ways of improving it. Often, these sessions were a welcome relief from the daily stresses of team members and many of them participated enthusiastically. Successful areas of process improvement included:

- CM - the CM process became critical during the testing and maintenance phases. QA and CM initiated a meeting of the project’s Integrated Project Team (IPT) - developers and users - where the complete Testing/QA/CM process was analyzed and diagrammed prior to a brainstorming session that identified loopholes in the current process and areas to improve. The process was successfully improved to everyone’s satisfaction, and we introduced the PVCS CM tool for stricter source code and documentation version control.
- Formalized Unit Testing with increased participation by QA
- Technical and QA review of all customer delivered items became fully integrated accepted into the development process
- Training - The QA function provided training in several of the areas such as test writing, processes, how to set up and maintain Software Development Folders (SDF), and how to run peer reviews, code walkthroughs, or Centers of Excellence (COE) sessions. The technical manager and team leaders held training sessions in a variety of technical areas such as using EXPLAIN PLAN, optimizing SQL queries, and using best practices for coding and reporting.

Results/Metrics

Metrics collected and used on the project included:

- Number of, and results of, QA records/reviews
- Areas of greatest problems/defects
- Customer acceptance statistics
- Requirements Coverage
- Test Coverage and Test Results
- Problem Reports/Defects found - per module, per subsystem, classification and type, time taken to resolve
- Development progress - Estimated vs. Actual

Lessons Learned

Acceptance of SQA - both the function and the person

It's vital that the QA function is perceived as both collaborative and "value-added". Someone who is confrontational with the developers, overly and personally critical, and who "points the finger" and apportions blame, is going to damage the reputation of the QA function - possibly for good. Perceptions are often incorrect, but unfortunately, often impossible to eradicate. However, if the QA manager is able to guide the development team through the maze of standards and procedures that support the project; continually help to improve, streamline, and facilitate the processes; and assist in training the project team, while providing continual oversight of the project to the project manager through reports and reviews, and all the while retain a sense of humor and requisite "thick skin", then the QA function in general, and the QA manager in particular, will be viewed as an integral component of any software project. You know you must be doing something right when developers who have left the company call you up months later to tell you of their horror at finding themselves in a company where "they don't have any QA!" On our pilot project, QA was always made to feel part of the team and was given responsibility for such tasks as the RTM and Requirements Change Process - activities that were seen as "concrete" and "added value" rather than just adding to a perception of "QA as police" checking up on everyone. Avoiding the "them and us" syndrome is critical in making QA a success. It is equally important that there be a structure to the QA activities rather than a sense of free-floating around the project looking for problems. This is particularly important when the QA person is billable, and accountability is an issue. A well-designed QA Plan, detailed and specific QA procedures, and a well-thought out QA schedule are tools that are invaluable in providing structure, guidance, and credibility to the QA function. Specific tasks related to the QA function were clearly defined for each life cycle phase.

What Worked?

- Technical and QA review process for all deliverables (internal and external)
- QA involved in requirements management from the very beginning
- Close collaboration between QA, CM and the Technical Manager
- QA involvement in Design Reviews
- QA review forms and QA records; QA established as final sign-off on deliverables
- Centers of Excellence (COEs), Code Walkthroughs, Peer Reviews
- Formalized and documented Unit Testing
- Using a database to facilitate the creation and maintenance of the RTM and requirements' coverage reporting.
- Using a COTS test management and bug tracking tool during integration test
- Entering all problem reports and action items into a database

What did NOT work?

- Anything involving excessive paperwork for the developers
- Anything that caused lengthy turnaround time for deliverables
- Expecting developers to read and digest lengthy standards documents
- Assuming that developers would enter all the required RTM information

Implementing QA on all projects

Once QA had been successfully integrated into a large and lengthy project, it is much easier to "clone" the process on other software projects. Much of the core work has already been done. You have your "Lessons Learned". You know which SQA activities were successful and which weren't, and where training is required. You have a stash of tried and tested artifacts - audit and reviews forms, standards checklists, procedures, test documentation, plans and guidelines, training and presentation materials; records; repositories; and tools. Some of the more successful documents on our project turned out to be test forms, peer review forms, and QA review and audit forms. Much more than this, you have documented results and have confidence in your processes. Not everything works. Not everyone will "come on board". But little by little, you will see the results of your efforts as people in the organization take for granted the QA function as

a part of their project. And their creativity is not affected one bit!

Expanding the QA Group

When it became time to hire more people for the QA group, we used a mix of part-time and full-time people. Some full-timers we hired had some experience of QA or testing already and were familiar with the software development life cycle. Some part-timers were already employees of our company and were talented, smart people who welcomed the opportunity to learn something new and perhaps even break into the software engineering arena. Some have enjoyed SQA so much that they eventually became full-time QA specialists or testers. Others moved into software development. Everyone received training - a 10-module internal training course in QA that provides a solid foundation for performing their tasks.

Extreme QA (XQA)

We were fortunate not to have to practice “Extreme QA” since we had the time and ability to build a range of full QA activities into the life cycle. However, our emphasis on requirements management, full test coverage to requirements, and a minimizing of complex and unnecessary paperwork and activities, meant that we already had an established QA process that could be streamlined to fit future projects where more aggressive timelines, smaller teams, and a rapid prototyping environment were the norm.

Conclusion

Our pilot project was successfully deployed and became an example of a full life cycle software development project with integrated quality assurance. Subsequent projects have gone on to utilize the same QA processes and procedures. By the end of the project, we had been externally assessed as being SEI CMM - Level 3 compliant.

About the Authors

Leslie Tierstein is a Technical Project Manager for STR L.L.C, a Fairfax Virginia-based provider of Oracle custom development and training. Leslie can be reached at ltierstein@earthlink.net or ltierstein@strllc.com.

Hilary Benoit is the Director of Quality Assurance for W R Systems, Ltd., Fairfax, VA, an Information Technology Services company, and is a certified quality professional with the American Society for Quality (ASQ) and the Quality Assurance Institute (QAI). Hilary can be reached at hbenoit@wrsystems.com.

Appendix: Sample Forms

Sample forms referenced in this document are given on the following pages. These are:

Figure 1: QA Review Form

Figure 2: Sample Functional Requirements List (FRL)

Figure 3: Project Deliverable Route Sheet


Figure 4: Peer Review Form

Figure 5: Prototype Verification Form

Figure 6: Unit Test Form

Figure 7: Sample Test Scenario

Figure 1. QA Review Form

QA QUALITY REVIEW QAF-02 Page 1 of 2	PROJECT:	
		TASK#:
	π INITIAL	DATE:
	π FOLLOW-UP	REVIEWED BY

1. TASK DESCRIPTION

2. CHECKLIST

DESCRIPTION	CONFORM TO STDS			COMMENTS
	YES	NO	NA	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
	π	π	π	
DESCRIPTION	YES	NO	NA	COMMENTS

3. WORK PRODUCT

Is the end result polished, professional & intelligible?	π	π	π	
Are features traceable to the requirements?	π	π	π	
Does the end result satisfy all of the requirements?	π	π	π	

4. METHODOLOGY

Were procedures adopted and used?	π	π	π	
Were baselines established and archived?	π	π	π	
Were change control mechanisms adopted & used?	π	π	π	
Were QA mechanisms adopted & used?	π	π	π	
Were sub-contract mgt mechanisms adopted and used?	π	π	π	

5. OVERSIGHT

Were reviews/tests performed per standards?	π	π	π	
Was a final comprehensive review/test performed?	π	π	π	
Were all QA activities performed per the QA Plan?	π	π	π	
Are there any open PR(s), ECP(s), ECO(s)?	π	π	π	
Is a follow-up review required?	π	π	π	

Figure 2: Sample Functional Requirements List (FRL)

Functional Requirements List

FLS Main

Requirement Identifier				Function
FLS	01	XXX	002	Verify and activate DODAAC data.
FLS	01	XXX	003	Maintain and print DODAAC data.
FLS	01	XXX	005	Create Navy Unit Identification Code (UIC) reports. (Deferred at CCB of mm/dd/yy)
FLS	01	XXX	006	Automatically update Master Address file based on DODAAC inputs.
FLS	01	XXX	007	Maintain, print, and view Master Address file. (Deferred at CCB of mm/dd/yy)
FLS	01	XXX	012	Provide the capability to import and export transactions via DAAS. These transactions include: MILSTRIP, MILSTRAP, MILSBILLS, DODAAC, DLSC, DLSS, SSR, WSF, and KSS.
FLS	01	XXX	013	Unload mailing and shipping addresses to TANDATA and FEDEX.
FLS	01	XXX	024	Provide capability to correct DAAS transactions that do not pass edit/error checks.
FLS	01	XXX	025	Maintain and print organization information. (Deferred at CCB of mm/dd/yy)
FLS	01	XXX	026	Maintain and print person information. (Deferred at CCB of mm/dd/yy)
FLS	01	XXX	027	Maintain and print domain information. (Deferred at CCB of mm/dd/yy)
FLS	01	XXX	028	Provide capability to query DODAAC information.
FLS	01	XXX	029	Maintain and print OPFAC data. (Deferred at CCB of mm/dd/yy)
FLS	01	XXX	030	Maintain and print carrier information. (Deferred at CCB of mm/dd/yy)
FLS	01	XXX	031	Provide capability to send email.
FLS	01	XXX	033	Provide the Address Comparison Report. (Approved as ECP#3)
FLS	01	XXX	034	Provide the Address Label Report. (Approved as ECP#4, but deferred at CCB of mm/dd/yy)

Figure 3. Project Deliverable Route Sheet

Project: _____ Files of _____

Task #: _____ Deliverable: _____

Date: _____ Return To: _____

Configuration Management - Document Check-Out		Complete <input type="checkbox"/>
Configuration Manager: Deliverable name for check-out, including version: Date of Check-Out: Original Document Name(s): New Document Name(s):		

Technical Review		Complete <input type="checkbox"/>
Initial Review		
Date Submitted: Reviewed By: Date Completed: Follow-up Review Required?:	Comments:	

Follow-up Review (if needed)		
Date Submitted: Reviewed By: Date Completed: Follow-up Review Required?:	Comments:	

QA Review		Complete <input type="checkbox"/>
Initial Review		
Date Submitted: Reviewed By: Date Completed: Follow-up Review Required?:	Comments:	

Follow-up Review (if needed)		
Date Submitted: Reviewed By: Date Completed: Follow-up Review Required?:	Comments:	

Production		Complete <input type="checkbox"/>
Date Submitted: Production Coordinator: Date Completed: Date Delivered:		

Post-Production Check	NA <input type="checkbox"/>	Complete <input type="checkbox"/>
------------------------------	-----------------------------	--

Configuration Management		Complete <input type="checkbox"/>
Date Submitted: Configuration Manager: Date Completed: Notes/Comments:		

Figure 4. Peer Review - Report

Peer Review - Report

Date/Time:	Date:	Start Time:	End Time:
Work Product:			
Peer Review Leader:			
Author:			
Reviewers:			
Notes taken by:			
Action Items:	<i>Note all action items resulting from this peer review session. Continue on a separate sheet, if necessary.</i>		

Figure 5. Prototype Verification Form

Deliverable Route Sheet Technical Review								
Deliverable (ELIN/Description): <u>Initial Prototype PDR</u>								
Subsystem Name: _____								
Submitted By: _____								
Return To: _____								
Report Name		INITIAL REVIEW				FOLLOW UP REVIEW		
		Date Submitted	Date Completed	Reviewer	Follow-up required?	Date Submitted	Date Completed	Reviewer
User Guide P:\PROJ\...Task2E\ \DRAFT\ UG .doc								
Comments:								
MODULE DEFINITION form report		INITIAL REVIEW				FOLLOW UP REVIEW		
		Date Submitted	Date Completed	Reviewer	Follow-up required?	Date Submitted	Date Completed	Reviewer
Comments:								

Figure 6: Unit Test Form

Project:		S/W Version:	
Module ID & Short Name:			Date:
SPR #(s) associated with this test:		<i>If there is no bug/SPR associated with this module, please check below, as appropriate.</i>	
SPR(s) attached: YES <input type="checkbox"/> NO <input type="checkbox"/> SPR #(s) received by phone: <input type="checkbox"/>		This is an Enhancement <input type="checkbox"/> This is New Development <input type="checkbox"/>	
Associated LCCB #(s) - if applicable:			
Test Site: WRS <input type="checkbox"/> Customer <input type="checkbox"/> Test Plan: Software Test Plan (ELIN 1010)	Developer Tester(s):		TEST database: HP755 <input type="checkbox"/> T600 <input type="checkbox"/> PRODUCTION database: <input type="checkbox"/> <i>Check at least one for QA to test in</i>
Brief Description/Purpose of Test: <i>Attach appropriate Bug Reports, if possible.</i>			
Other <u>affected</u> modules <i>(for unit integration/impact analysis) These include other screens or reports that this change will affect and that therefore will need to be tested.</i>			
Other associated packages, functions, procedures and views: <i>Specify ALL that need to accompany this test.</i>			
Test Complete? Yes <input type="checkbox"/> No <input type="checkbox"/>		Test Result: Pass <input type="checkbox"/> Fail <input type="checkbox"/>	
Results/Comments:			
Item/Action to be Tested <i>(complete on separate sheet, if necessary)</i>		Result	
Developer Tester Signature(s) & Date:		Peer Review Signature & Date:	
Does the User Documentation require updating as a result of this bug fix/enhancement? YES <input type="checkbox"/> NO <input type="checkbox"/>			
QA Signature & Date:			
CM Information	Date moved to TEST:	Date moved to PRODUCTION:	
Version of module:			

Figure 7. Sample Test Scenario

Test Scenario Form (example)					
Subsystem: FINANCE	Functional Area: ACCOUNTS RECEIVABLE				
<p>Description: (Please be brief but clear) Accrue any applicable late charge interest and generate dunning notices for all eligible receivables.</p> <ol style="list-style-type: none"> 1. Perform query (and print results) to determine if : <ol style="list-style-type: none"> a. For bill x, late charge interest has accrued in the amount of \$y 2. Print and review dunning notices to determine if: <ol style="list-style-type: none"> a. The customer name and address are correct b. The date is correct c. The principal, late charge penalty, late charge administrative fee, and late charge interest are listed correctly. 					
<p>Input: Bill #000007 - Late charge interest \$1.64 Bill #000010 - Late charge interest \$16.42</p>					
<p>Output: Entry posted to general ledger Dunning notice printed</p>					
<p>Required Results: Query successful. Dunning letter printed showing correct data.</p> <p>The following entry shall be posted to the general ledger:</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 70%;">Accounts Receivable - Billed</td> <td style="text-align: right;">\$pp.pp</td> </tr> <tr> <td>Interest Earned</td> <td style="text-align: right;">\$qq.qq</td> </tr> </table>		Accounts Receivable - Billed	\$pp.pp	Interest Earned	\$qq.qq
Accounts Receivable - Billed	\$pp.pp				
Interest Earned	\$qq.qq				
<p>Other subsystems interfaced with: Supply</p>					