11-2013

A Study of Quality Assurance and Testing in Software Development Life Cycle

Pranathi Nandhyala
St. Cloud State University

This starred paper submitted by Pranathi Nandhyala in partial fulfillment of the requirements for the Degree of Master of Engineering Management at St. Cloud State University is hereby approved by the final evaluation committee.

Ben Baliga
Chairperson

Kbmyanicus

Patricia Hughes
Dean
School of Graduate Studies

A STUDY OF QUALITY ASSURANCE AND TESTING IN SOFTWARE DEVELOPMENT LIFE CYCLE

by

Pranathi Nandhyala

B.Tech., Jawaharlal Nehru Technological University, 2006

A Starred Paper

Submitted to the Graduate Faculty

of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Engineering Management

St. Cloud, Minnesota

November, 2013

ACKNOWLEDGEMENT

I would like to express my sincere thanks to the committee members for their involvement in my starred paper. I believe that their timely advice and inputs will help me in the successful completion of the paper.

I would also like to thank the Department of Engineering Management, Mechanical and Manufacturing Engineering and also library of St. Cloud State University for all the valuable resources I was given access to.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

ABSTRACT

OBJECTIVES OF THIS DOCUMENT

The objective of this document is to specify a Software Quality Life Cycle (SQLC) that will be used in the development of high quality software. The goal is to create streamlined usable process that supports the SQLC so that the activities related to software quality can be integrated into the existing software development processes. In addition, it is important that we create these processes so they will:

- Not inhibit the flow of work
- Not inhibit the creativity of the people
- Not fail immediately because of the time or resources required
- Not fail in the long run because the process or life cycle is unsupportable or inflexible.

This document will:

- Outline the Software Quality Life Cycle (SQLC) and the steps in that life cycle.

- Focus on a framework and guidelines, not step by step instructions.
- Define software quality and testing terms that may be unfamiliar or used inconsistently.

Benefits for the users of this document:

- Provide a repeatable process where the user don't have to reinvent the wheel.
- Decrease the learning curve of those new to software quality assurance.
- Aid communication and eliminate confusion with the use of consistent terminology.
- Contribute to a higher degree of accuracy for project estimates.

SCOPE OF THIS DOCUMENT

The Software Quality Life Cycle testing involves continuous testing of the system during the development process. At predetermined points, the results of the development process are inspected to determine the correctness of the implementation. These inspections identify defects at the earliest possible point. This document will explain the Software Quality Life Cycle (SQLC) process and how it relates to the Software Development Life Cycle (SDLC).

This document will:

- Encompass the full life cycle of quality assurance and testing an application.
- Include main levels of testing (unit, system, user acceptance, and installation).
- Provide an overview of all types of testing.
- Focus on web and client/server applications.
- Address maintenance testing.

Chapter 2

SOFTWARE DEVELOPMENT LIFE CYCLE

Software Development Life Cycle is the process of developing information systems through analysis, planning, design, implementation, integration, maintenance and testing of software applications. SDLC is also identified as information systems development and also application development. The usage of a structured approach towards the design and development of the end product is very essential for the development of quality software. In short, the following could be stated as the key measures for the success of the SDLC process for building a successful software system:

- Restricting the scope,
- Progressive improvement,
- Pre-defined and well Organized Structure,
- Incremental and Advanced Planning at each of the stages.

Software Development Life Cycle (SDLC) can also be stated as a methodology that is generally used for the development, maintenance and replacement of the information systems for the purpose of improving the quality and reliability of the design and development phase of a software.

Most of the risks that arise in the software development life cycle can be alleviated if each of these steps can be followed in its entirety.

The general phases are:

- Analysis;
- Estimation;
- Design;
- Development;
- Integration;
- Testing;
- Implementation.

It usually involves designing the application from the very basic level, documentation of everything, making improvements and fixing all the defects and bugs that are detected in SDLC as a part of testing. It is the lifecycle of Software from scratch to completion.

Reliable and quality software is conceptualized based on the following principles:

- Execution of the software development process (process),
- Modeling of the measurement of product (product),
- Formation of the management and human interactions (people).

SDLC STEPS

The software development life cycle can be divided into various steps as below:

1. Requests of the product from customers/original idea;
2. Creation of attribute lists based on item 1;
3. Technical design of attributes based on item 2;
4. Design of the software and time assessment based on item 3;
5. Implementation of the code based on item 4;
6. Software Testing;
7. Fixing the detected bugs;
8. Final release of the software.

Discussion of each step in brief:

1. Requests of the product from customers/original idea.

Any kind of information from the client or the customer which is relevant to the system, its attributes and scenarios is gathered in this phase. This mostly includes all the artifacts that are intended to be produced, suggestions of any tools to be used etc. Basically these are the raw specifics and details or materials collected from the customer.

2. Creation of feature lists based on the specifications.

This step deals with the consideration of refining the raw specifics and details gathered in the previous phase. Depending on the processes and standardization levels they adopt, this refinery process generally varies from one organization to other. Proper organization of all the details of the proposed system is done in this stage.

3. Technical design of attributes based on the lists.

This phase deals with all the technical aspects to be used in the process. All the technical aspects come into the scene and the designing of those features in terms of computer terminology takes place at this stage. It involves designing of the architecture and the causes that influence and affect them.

4. Design of the software and time assessment based on technical design.

In the software development life cycle, the software design phase is produced from the results of the requirements phase. It includes detailed design of the software in terms of the interfaces that it will present and also the applicable functionalities that have to be implemented. It also involves the scheming of the database. In this phase it is also very important to estimate the timeline required based on the number of resources available.

5. Implementation of the code based on the software design.

Writing the source code based on the required functionality by sticking to the coding standards, code optimization, etc. is done in the coding phase.

6. Software Testing

It is important to make sure that the software functions properly after it is developed. Software testing deals with various mechanisms that are used to report the bug and unwanted results that might occur in the Software Development Life Cycle.

7. Fixing the detected bugs.

This is sometimes called Beta release. To undergo the tests in order to check issues like the performance and behavior under different circumstances, the newly developed software will be released. Any other bugs that occurred earlier in the cycle are rectified here.

8. Final release of the software.

The software is ready to be released to the customer once all the bugs identified in the previous stages had been rectified and the fix has been implemented.

SDLC CATEGORIES

Broadly, the SDLC steps discussed earlier can be categorized into the following:

- Specification of Requirements;
- Analysis and Design of Requirements;
- Coding and Testing;
- Deployment and Support.

Figure 1 shows the software quality of life cycle.

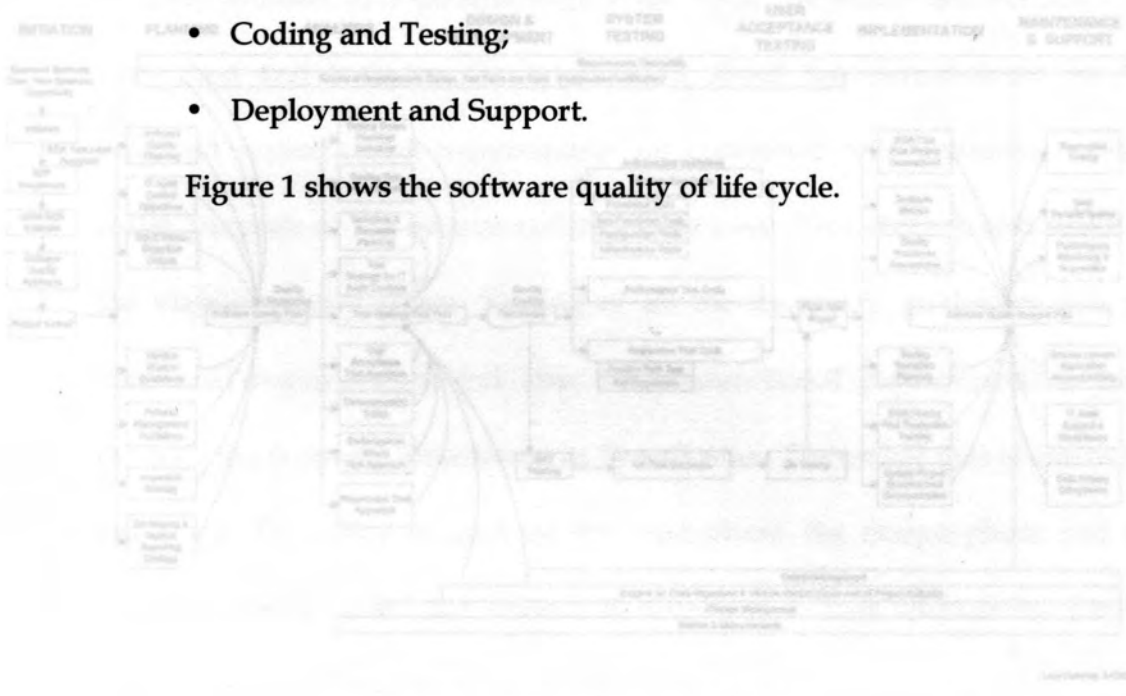

Figure 1. Software Quality Life Cycle

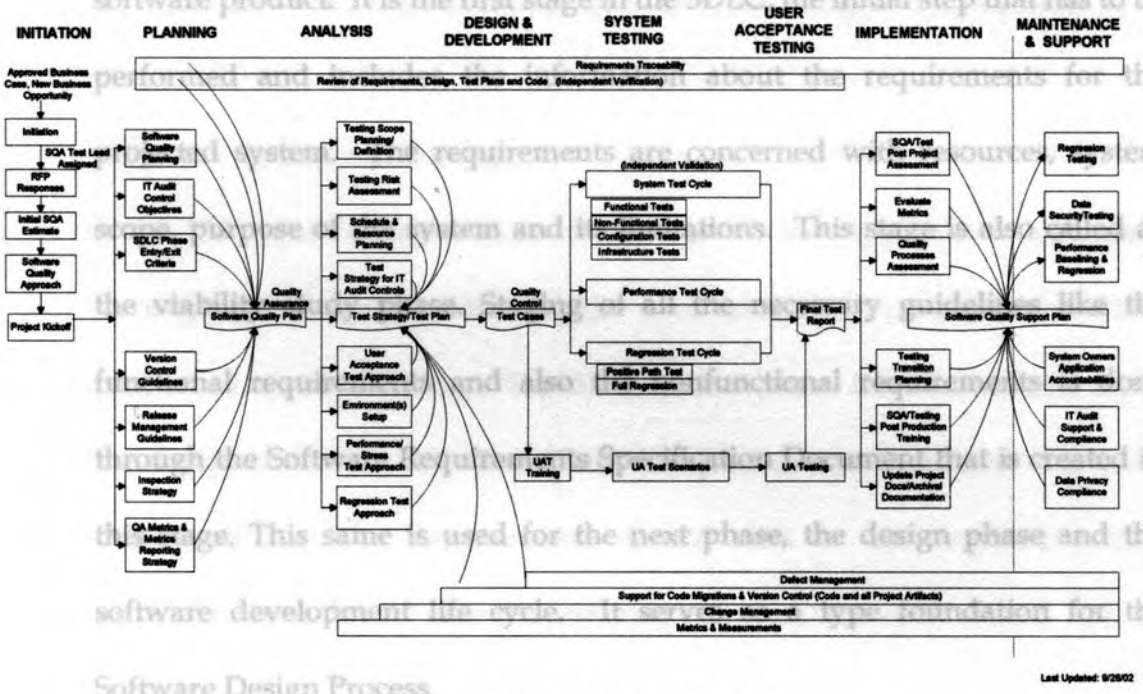

Figure 1. Software Quality Life Cycle

Detailed discussion of the categories:

1. Specification of Requirements.

Software Requirements Specification (SRS) provides an absolute explanation of all the functions and conditions of the software to be designed. It extracts the functional and the nonfunctional requirements of the preferred software product. It is the first stage in the SDLC, the initial step that has to be performed and includes the information about the requirements for the projected system. The requirements are concerned with resources, system scope, purpose of the system and its limitations. This stage is also called as the viability study phase. Staging of all the necessary guidelines like the functional requirements and also the nonfunctional requirements is done through the Software Requirements Specification Document that is created in this stage. This same is used for the next phase, the design phase and the software development life cycle. It serves as a type foundation for the Software Design Process.

2. Analysis and Design of Requirements.

As soon as the requirements specification is done, it is examined for its accuracy and competency. Issues such as if that specific proposed system will serve the purpose or not, what are the essential constraints that have to be taken into consideration and observing the management of the activities in an

approach are all carried out in a hierarchical approach which is mentioned in the Business Requirement Specification document.

The design stage takes the requirements identified in the accepted requirements document as its opening input. The Analysis and Design phases are very important in the software development life cycle. This phase also considers the work assignments, project schedule, budgets and deadlines to be met.

3. Coding and Testing.

The next step involves formation of the application logic and developing the functionalities for the user interfaces on a particular action performed which is coding. The development phase takes as its chief input the design rudiments described in the accepted design document. It is to be considered that the Programmers should stick to the required coding principles. In order to save the valuable resources the code should also be optimized.

Now comes the Testing phase of the Software Development Life Cycle. After the source code is created, the testing phase begins to divulge the bugs and defects that were committed during the previous phases. This testing can be manual or it can be automated using the testing tools and methodologies that are in use.

4. Deployment and Support.

The software that has been developed and deployed should provide sufficient support for maintenance so as to keep up with the bugs that can exist even after deployment of the software is completed. This could sometimes take far more time than the original development process of the software.

The maintenance phase of the software development life should be capable to support the changes that can occur during the post implementation stage. Unexpected input values into the system can result in a change in the working system. Figure 2 illustrates the interaction between the SDLC, the SQLC, and the RMM.

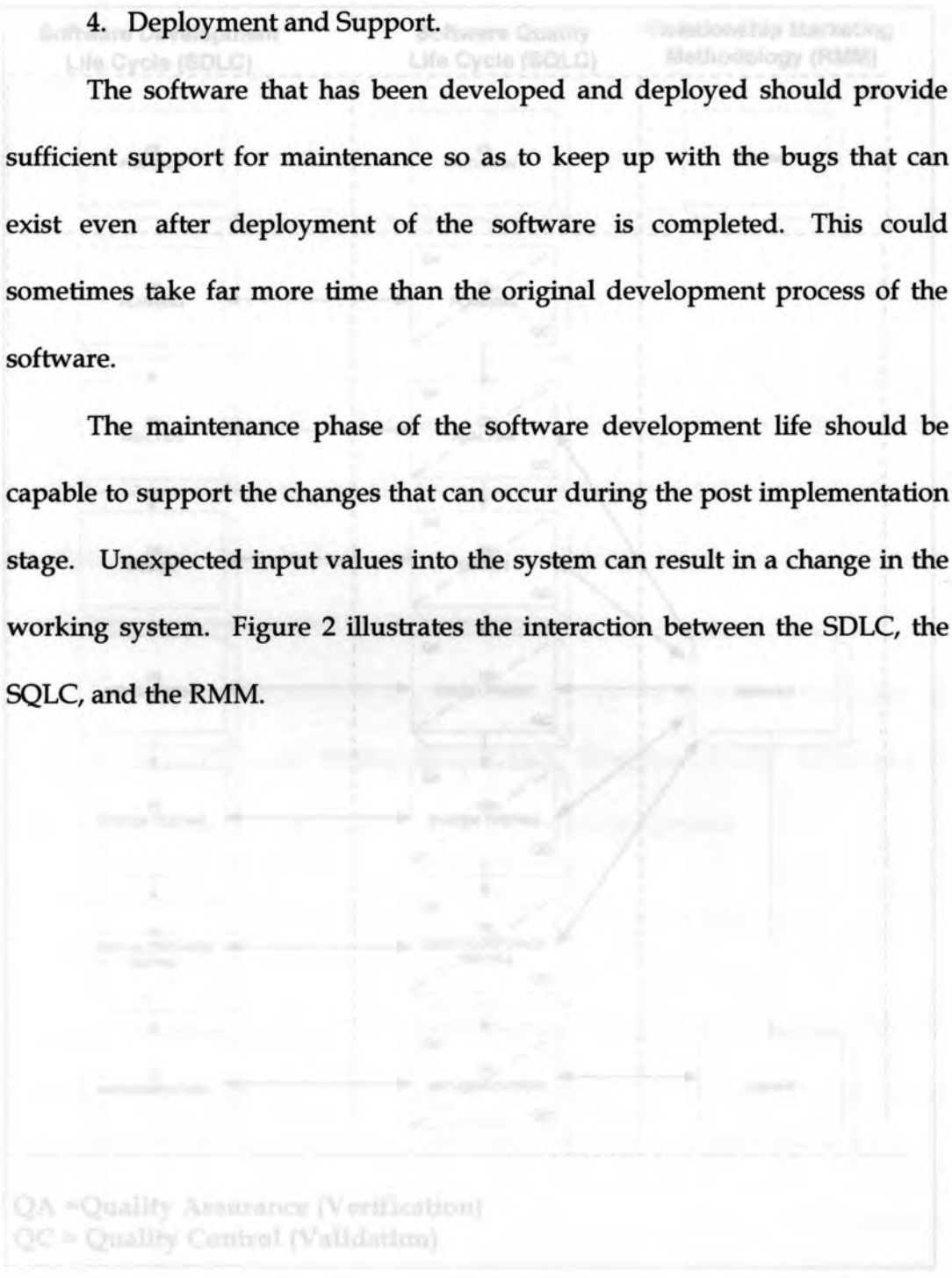

Figure 2. Interaction between the SDLC, the SQLC, and the RMM

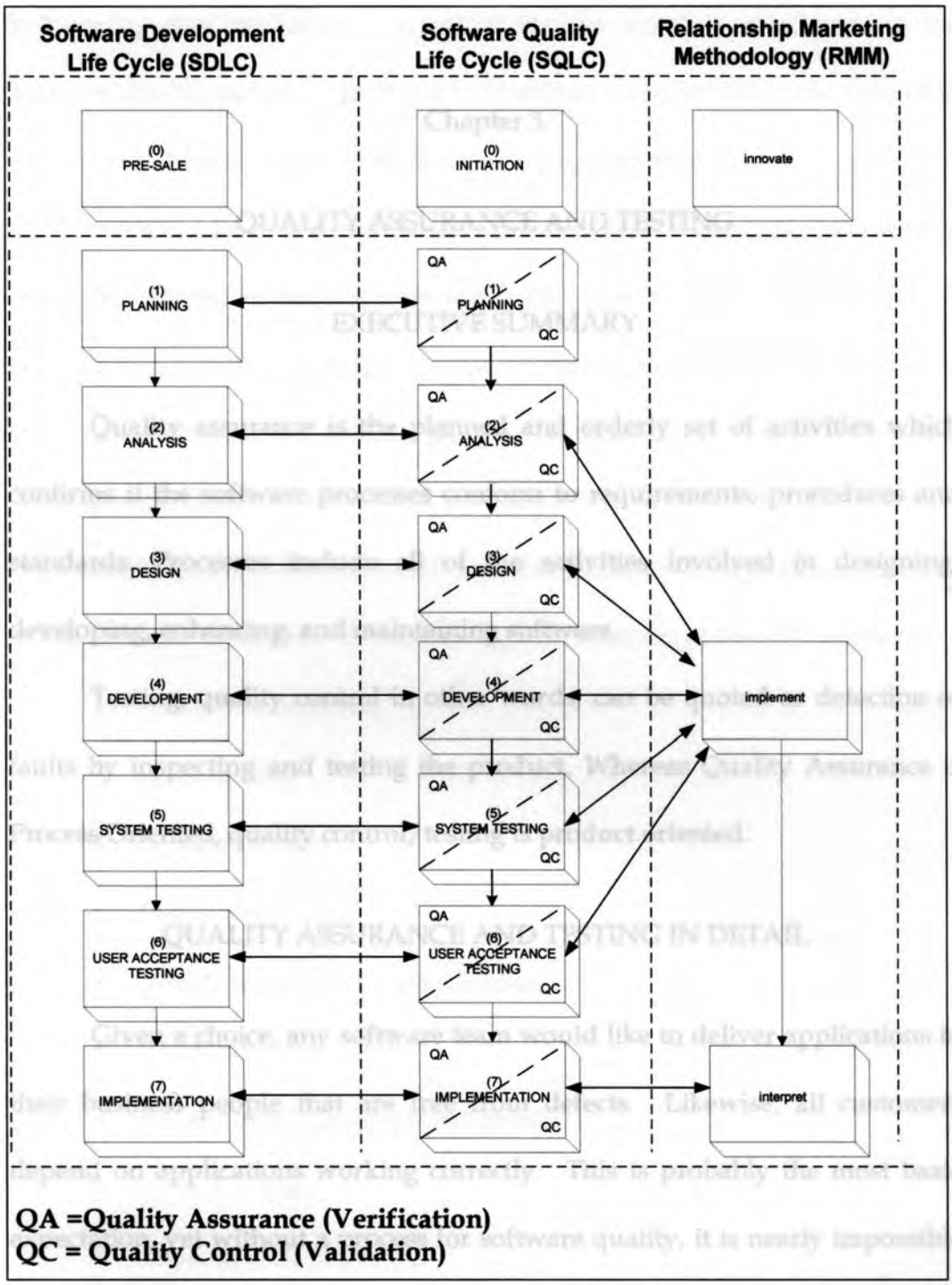

Figure 2. Interaction between the SDLC, the SQLC, and the RMM

Chapter 3

QUALITY ASSURANCE AND TESTING

EXECUTIVE SUMMARY

Quality assurance is the planned and orderly set of activities which confirms if the software processes conform to requirements, procedures and standards. Processes include all of the activities involved in designing, developing, enhancing, and maintaining software.

Testing, quality control in other words, can be quoted as detection of faults by inspecting and testing the product. Whereas Quality Assurance is Process Oriented, quality control/testing is product oriented.

QUALITY ASSURANCE AND TESTING IN DETAIL

Given a choice, any software team would like to deliver applications to their business people that are free from defects. Likewise, all customers depend on applications working correctly. This is probably the most basic expectation, yet without a process for software quality, it is nearly impossible to attain. It is a most common assumption that testing will solve all of the

software quality problems. In reality, testing is only a small part of the software quality picture. Having a solid software quality life cycle process in place is vital to the goal of delivering quality applications.

There are several characteristics of good quality assurance and testing practices as noted by leading software quality practitioners. For the SQLC, it is idealistic to employ both QA and QC methods. Quality Assurance (QA) is the process of building quality into the development process to prevent defects, while Quality Control (QC) is the process of validating that software any remaining defects are detected and fixed before the application is made available to the customer. The following are examples of QA/QC processes:

- Systematic methods,
- Early testing,
- Repeatable tests,
- Test Automation,
- Quality metrics & measurements,
- Test Strategies,
- Written test cases/expected results,
- Version control,
- Inspections & Reviews,
- Defect prevention and tracking,

- Software quality and testing risk assessments,
- Integration of testing and IT Audit controls,
- Levels and types of testing considered,
- Development and test release/migration process integration.

Defining a Software Quality Process that encompasses many of the thoughts and principles mentioned above is further illustrated. It is intended to be a framework that can be adapted to specific development environments and projects.

The matrix below (Table 1) explains the differences between SDLC and SQLC and details and also the various steps involved in both the cycles.

Although both work in a similar fashion, the focus to detail on quality assurance in software quality life cycle is what sets it apart from software developmental life cycle.

Table 1. Deliverables of the SDLC and SQLC

Life Cycle Phase	SDLC	SQLC
Initiation	• Project Kickoff	• Project Kickoff
Planning	• Service Request	• Software Quality Plan
Analysis	• BRD - Business Requirements Document • Project Schedule • CSS/ESP Documents - Updated • IT Estimate (Updated) • IT Costs • CSS/ESP Costs • Risk Assessment • MOT SOW • Issues Log •	• Test Strategy/Test Plan •
Design	• Development Standards - "New" • Best Practices - "New" • Systems Design Document	• Test Cases
Development	• Unit Test Plan • Source Code	• Test Cases
System Testing	• System Test Plan • System Test Cases	• Final Test Report
User Acceptance Testing	• End User Documentation • UAT Plan • UAT Cases	• Final Test Report
Implementation	• System Documentation • Production Implementation Plan - checklist • CCI Audit Checklist - signoff • CSS/ESP Documents • Status Report - Final	• Software Quality Support Plan
Maintenance and Support	• IT Support Plan	• Software Quality Support Plan

Chapter 4

KEY SOFTWARE QUALITY PRINCIPLES

The purpose of this section is to describe the activities performed during the Software Quality Life Cycle (SQLC). The activities, although numerous, should be considered for the inclusion of a software project. The amount of software quality is governed by the complexity, size, risk, schedule and budget of the project.

The Software Quality Life Cycle (SQLC) supports and is supported by industry-wide software quality principles, some of which are described in the following subsections.

SOFTWARE ENGINEERING INSTITUTE'S CAPABILITY MATURITY MODEL

The Capability Maturity Model (CMM) for Software identifies and explains the ideology and practices underlying the maturity of the software process and is intended to help software organizations improve the maturity of their software processes in terms of an evolutionary path from ad hoc,

disordered processes to mature, orderly software processes. The maturity model can be organized into five maturity levels described below:

1) **Initial**. The software process is indicated as ad hoc, and seldom chaotic. Few processes however are defined, and success mostly depends on individual work and heroics.

2) **Repeatable**. Cost, functionality, and schedule are tracked by establishing basic project management processes. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3) **Defined**. The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization.

   All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

4) **Managed**. Detailed measures of the software process and product quality are collected. Both the software products and process are quantitatively understood and controlled.

5) **Optimized**. Quantitative feedback from the process and from piloting innovative ideas and technologies is used for continuous process improvement.

The touch points of the software Quality Life Cycle as they relate to the Capability Maturity Model are indicated in Table 2.

Table 2. Touch Points of the Software Quality Life Cycle as It Relates to the Capability Maturity Model

• Requirements Management	• Integrated Software Management
• Software Project Planning	• Peer Reviews
• Software Project Tracking	• Quantitative Process Management
• Software Subcontract Management	• Software Quality Management.
• Software Configuration Management	• Defect Prevention
• Organization Process Focus	• Technology Change Management
• Organization Process Definition	• Process Change Management
• Training Program	

SIX SIGMA

In the six sigma methodology, quality is defined in relation to understanding and meeting customer requirements. Quality is a measurable outcome and that the customer is the ultimate judge of quality.

As a process improvement methodology, six sigma began in the mid-80s at Motorola purely as a statistical problem solving tool. Other companies soon began to adopt the same methodology. With Allied Signal in 1993, six sigma developed from a statistical problem solving tool into a robust methodology of leadership and support systems. Six sigma really came to the forefront when General Electric committed six sigma. The huge impact in dollars saved carried the six sigma message forward to other companies. Now, companies all over are implementing six sigma methodologies.

Six sigma uses the DMAIC model to help fix established processes. The DMAIC model is:

1. **Define**: Set project goals and boundaries based on the organization's business goals, customer needs, and the process that needs to be improved to get the user to a higher sigma level.
2. **Measure**: Pinpoint the location or source of problems as precisely as possible by building a factual understanding of existing process conditions and problems. Establish a baseline capability level.
3. **Analyze**: Develop theories of root causes, confirm the theories with data, and finally identify the root cause(s) of the problem.

4. **Improve**: Develop, implement, and evaluate solutions targeted at the verified cause. Demonstrate, with data, that the solutions solve the problem and lead to improvement.
5. **Control**: Helps make sure the problem stays fixed and that the new methods can be further improved over time.

Six sigma focuses on:

- Consistently improving customer satisfaction.
- Anticipating the ever-changing needs of the customer.
- Making fact-based decisions.
- Understanding process variation.
- Supporting the business by making bottom line numbers.

Table 3 shows the touch points of the Software "Quality Life Cycle as they relate to six sigma.

Table 3. Touch Points of the Software Quality Life Cycle as It Relates to Six Sigma

• Document Control • Metrics • DMAIC model for Existing Product or Service • Financial Analysis / Cost of Quality • Management	• Process Management • Simulation

V-MODEL SOFTWARE DEVELOPMENT LIFE CYCLE

The left side of the V-Model diagram represents software and test development (verification), and the right side represents test execution (validation). The V-Model is an improvement over other models, since test development occurs in advance of test execution.

The V-Model has well defined phases and milestones (see Figure 3). It has dedicated test phases, and test findings will generally only cause modifications to deliverables of the current phase. The V-Model contains feedback and validation mechanisms to ensure production of a deliverable that meets future requirements.

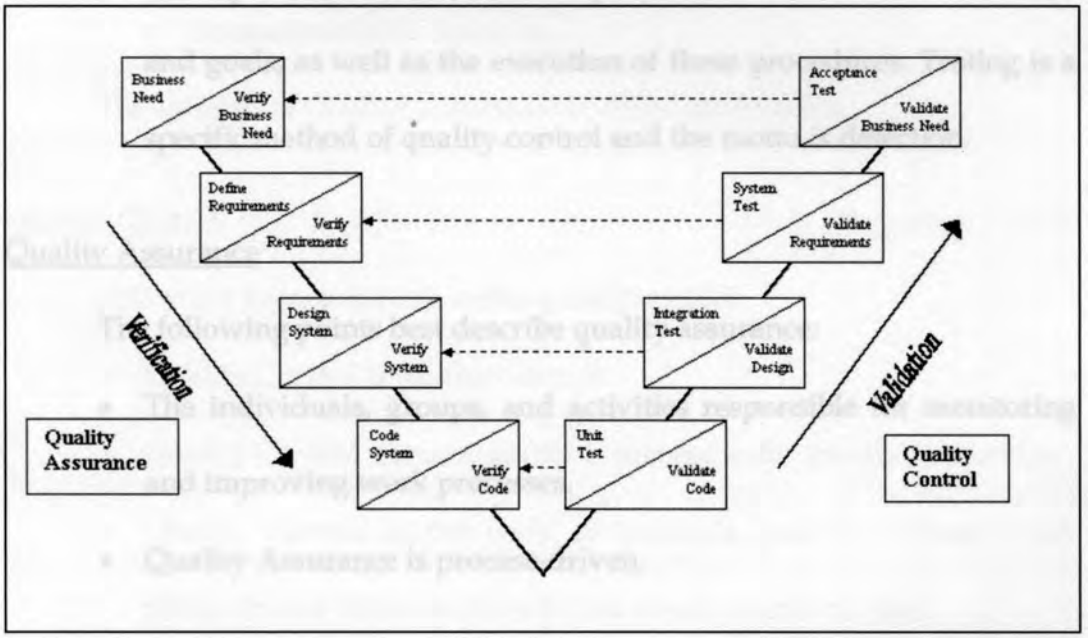

Figure 3. V-Model Software Development Life Cycle

QUALITY ASSURANCE VS. QUALITY CONTROL (QA VS. QC)

These two terms have many interpretations because of the different definitions for the words **assurance** and **control**; in turn, "quality assurance" and "quality control" are sometimes erroneously thought of as synonyms when referring to the actions performed to ensure the quality of a product, service, or process. Below are the descriptions of these terms as they relate to the SQLC:

- Quality Assurance establishes standards and goals to ensure the work products meet customer needs or expectations. It is the process and is preventive in nature.
- Quality Controls are established procedures to meet those standards and goals, as well as the execution of these procedures. Testing is a specific method of quality control and the motto is detection.

Quality Assurance

The following points best describe quality assurance:

- The individuals, groups, and activities responsible for monitoring and improving work processes.
- Quality Assurance is process-driven.

- Quality Assurance ensures quality is built into the product or service.
- Quality Assurance ensures that processes are established and continuously improved.
- Quality Assurance is the set of support activities needed to provide adequate confidence that processes are established and continuously improved in order to produce products that meet specifications and are fit for use.

Set of support activities.

- Facilitation,
- Training,
- Measurements & Analysis.

The mission of Quality Assurance is **defect prevention**.

Quality Control

Following aspects best describe quality control:

- Quality Control is product-driven.
- Quality Control ensures quality is present in the product or service.
- Quality Control is the body of methods used to validate work products and demonstrate whether requirements are met.

- Activities and individuals with responsibility for checking product quality with applicable standards.

Set of control methods.

- Software Testing,
- Reviews,
- Walk-throughs,
- Inspections.

The mission of Quality Control is **defect detection**.

Work Products

The remainder of this section describes the Software Quality Life Cycle umbrella process and their QA and QC roles. These processes are shown at the top and bottom of the SQLC Diagram because they are key supporting practices that span multiple phases of both the Software Development Life Cycle and the SQLC, and are the foundation for a successful quality implementation.

Independent validation/verification. The creator of the code/document is not the same person who reviews it. Design Reviews, Code Reviews and Inspections are performed to ensure requirements are met; this is a QC function. A Carlson Marketing Group Quality recommendation states that

System testing should be performed independent of the developer(s); this is also a QC function.

Version control. Version Control is the process of archiving versions of software and logging changes to the documents, software and defects. Version control is a critical first step toward more efficient development and asset management practices. By organizing revisions, enabling concurrent and parallel development with less risk, and reducing common team coordination errors, version control enables developers to produce higher quality software, faster, and with less rework. Lost changes, overwrites and content errors are risks associated with not practicing version control; these are all QC functions.

A Carlson Marketing Group Quality recommendation states that projects should be under configuration/version management; this is a QA function.

Release management and code migrations. Release Management is the coordination and discipline of controlling and auditing the changes that are incorporated in a build or release of a software product, while Code Migrations are the act of promoting new or enhanced code into either a testing or production environment; both are a combination of QA and QC functions.

Metrics and measurements. Defect Tracking is one way to gather and measure statistics in order to determine the quality and success of a project; the actual tracking of defects is a QC function, while the reporting of them is a QA function. Another metric is the measurement of number of requirements to number of test cases.

- **Measurement:** A single attribute of a project or process (e.g., number of defects).

A measure is a quantitative description of a single attribute of a project, service, or process. For example, hours would be a measure associated with a process, defects is a measure associated with a product; and customer satisfaction could be a measure associated with a service. If these measures are standardized, then every time they are used the measure will have the same meaning. It can be used effectively to manage by fact.

The measurement of an hour as a time to complete a work task is subject to interpretation unless it is well defined. For example, an hour could be a paid hour or a worked hour. We might pay someone eight hours during a day, but they may actually work ten hours. We need to know whether the hours mean hours worked or hours paid for. Likewise, a defect can mean many things to many people. If a user forgets to provide a requirement, is that a defect? The number of defects will be impacted by the precise

definition of a defect. Therefore, the definition of a measure is called a standard unit of measure. Without standard unit of measure, collected quantitative data is not comparable.

A measure should meet two characteristics:

a) **Reliability**: consistency of a measure

- If the measure were taken again, would the result be the same?
- Can different people apply the same measure with the same results?

b) **Validity**: degree to which a measure actually measure that it was intended to

- A measure can have reliability, but lack validity.
- If a measure is not reliable, it cannot be valid.
  - **Metric:** Two or more measures combined to evaluate a project or process (e.g., number of defects per 1000 function points).

    Examples of metrics are:

    $$\text{Quality} = \frac{\text{Number of Defects}}{\text{KLOC / Function Points}}$$

    $$\text{Rework Cost} = \frac{\text{Cost to Correct Defects in Phase}}{\text{Total Phase Cost}}$$

$$\text{Computer Availability} = \frac{\text{Down Time}}{\text{Available Time}}$$

Requirements traceability. Test coverage best practices dictate that business requirements coverage requires 100% testing coverage. This level of coverage may be exhaustive in many cases, but it is important that testing coverage is adequate to ensure that all the requirements are met.

Business requirements coverage verifies that the system meets all the stated business objectives as outlined in the system requirements documentation; system tester usually performs this testing during the planning and test authoring stages of a project. The system tester produces a test requirement to test case matrix.

Requirements traceability is a QA function.

Defect tracking. Defect Tracking is used to trace defects throughout the project, including the following:

- Assign Defects,
- Communicate Defects,
- Prioritize Defects,
- Manage and Report Defects.

A defect is defined as a discrepancy between expected and actual results. Defects occur at every opportunity in the project: Between the customer and the high level description, between the high level description and detail design, between design and source code, between source code and as delivered executable. Also, defects can appear at every architecture or network layer, tier, and interface. Implementation of defect tracking is imperative in order to drive delivery of the product or service.

The actual tracking of defects is a QC function, while the reporting of them is a QA function.

Change management. Change management on a project exists for a number of reasons:

- To make sure that changes are documented according to the documentation guidelines in the SDLC.
- To make sure the changes are numbered for tracking purposes.
- All changes have an audit trail to determine the following
  - Who requested the change,
  - Who approved the change,
  - When the change was made,
  - What was changed,
  - Who made the change.

- To ensure changes are documented in the Business Requirements Document(s), Design Documents, System Documentation, Test Cases, etc.

Change management is a QA function.

Chapter 5

SOFTWARE QUALITY LIFE CYCLE

PROJECT INITIATION PHASE

The objective of this phase of the Software Quality Life Cycle is to establish and define Software Quality Assurance role in the project. An output of this phase is the Project Kickoff.

The remainder of this subsection describes the activities that contribute to this phase of the SQLC.

Initial SQA Estimate

In the initiation phase of a project the New Business team is preparing a new business case and following up on a new business opportunity. As the RFP responses are being gathered and an estimate is being prepared an SQA Team Lead should be assigned to prepare the Software Quality estimate.

The types and scope of testing required will be often fairly well known based on the type of system that is being developed and prior experience testing systems for clients with similar applications. A number of factors influence what percentage of the overall IT project estimate the Software

Quality hours may be, such as number of browsers and operating systems the system testing must be executed in, if performance testing should be executed, etc.

Software Quality Approach

This document specifies for the client how the software will be tested and what standards will be employed for testing purposes. This is not always a deliverable required by a project.

Project Kickoff

The purpose of a Project Kickoff meeting is to formally notify all stakeholders that the project has begun, and make sure everyone has an understanding of their roles and responsibilities. The kickoff meeting is a time to get all the team members, clients and stakeholders together and formally set the stage for the start of the project.

PLANNING

The main objective of this phase of the Software Quality Life Cycle is to collect and verify the information from the Planning phase and to create the Software Quality Plan, which is the output of this phase.

The remainder of this subsection describes the activities that contribute to this phase of the SQLC.

Software Quality Planning

Planning how to build quality into the process instead of validating that quality exists after the application has been constructed; this includes planning, methods, reuse, build release guidelines, Unit Testing strategies to make sure that the software is robust and defect-free before inspections and testing.

IT Audit Control Objectives

SQA should complete the 11 Carlson Companies Audit approved IT Audit control objective worksheets to ensure requirements, design and test documentation are planned for accommodating all of the objectives outlined in these worksheets (this is usually in a non-functional requirements document).

Life Cycle Phase Entry/Exit Criteria

The purpose of having entry and exit criteria is to establish a standard of when deliverables are complete and dependent activities can begin. It also enables a smooth transition between activities of both the SDLC and SQLC.

Hypothetical examples of entry and exit criteria are described in sections below:

- **Initiate Testing**

  Begin testing when the following criteria are met:

  a) After review and approval of the written test plans by the Project Manager(s).

  b) Upon receipt of full functional 'test level' code.

  c) Upon creation of a testing environment, including the needed hardware and software.

  Testing should not begin until the developers have signed off on their unit testing. Code may be delivered in pieces; so initial testing may be done using pre-determined test cases.

- **System Testing**

  Testing is an attempt to extricate all of the errors from a computer system. No system can be tested with all of the possible test cases. Even a relatively simple system will have an infinite number of valid inputs and an equal or larger number of valid outputs. Given that all test cases cannot possibly be performed, it is not possible to be certain that a system is completely error-free. The job of the testing staff is to make certain that as many errors as possible are

discovered and fixed. In some cases the cost of the testing could exceed the value of the system being developed. Some variables which determine when testing is completed are:

- All included test procedures have executed successfully once.
- There is mutually satisfactory resolution to all known defects.

Version Control Guidelines

These are approved standards that have been established for the process of storing versions of software and logging changes to documents, software (code) and defects. This process is used for new development and maintenance of existing applications, to control risk mitigation and is a requirement for release management.

When proper version control processes are in place, it allows for the flexibility to revert to a previous version of a piece of code if/when necessary.

Release Management Guidelines

These are approved standards that have been established to document, schedule and control the changes that are incorporated into a new build or release for Production deployment.

An audit trail exists when proper release management processes are in place.

Inspection/Review Strategies

Partnering with developers to build a robust unit test process that eliminates defects before release to system testing prevents defects from occurring by having rigorous development standards.

- **Inspections**

  Software inspections provide an immediate and concrete step that every organization can take to improve its process maturity. They provide a powerful mechanism for improving software product quality by detecting and correcting defects early and preventing their reoccurrence.

- **Peer Reviews**

  Peer Reviews are considered an industry best practice for detecting software defects early and learning about software artifacts. Peer Reviews are composed of these software walkthroughs and are essential to software product engineering activities.

QA Metrics and Reporting Strategy

The following are examples of QA Metrics that will be defined during the Planning phase, and measured and reported to Project Management throughout the SQLC/SDLC:

- Requirements metrics,
- Defect metrics,
- Test coverage metrics,
- Test risks metric,
- IT Audit Controls Coverage metric,
- Final test report.

Software Quality Plan

The Software Quality Plan is the deliverable that specified how the application will "build quality in" to the planning, analysis and development activities preceding system and user acceptance testing. Important components of the Software Quality Plan are:

- IT Audit and Control Objectives – since these objectives are well known before requirements are defined it is important that they are reviewed and integrated into the requirements definition, design and development phases. Part of the Software Quality Plan will specify whose role it is to integrate the Audit objectives into each phase and how these Audit objectives will be reviewed.
- SDLC Phase Entry/Exit Criteria – Each phase will have specify entry and exit criteria to fully document the handoff process between each phase in the Software Development Lifecycle (SDLC).

- Version Control Guidelines—Early in the project code control and version control will be implemented using a supported version control tool. The Software Quality Plan will indicate project assets that will versioned and naming conventions.
- Release Management Guidelines—The Software Quality plan will document the release strategy and build schedule that will include a specification of handoff and migration documents.
- Inspection and Review Strategies—The Software Quality plan will document which project assets will have formal or informal reviews for the purpose of defect prevention.

ANALYSIS

The objective of this phase of the Software Quality Life Cycle is to create the test strategy/test plan for the project, which are the outputs of this phase.

Testing Scope Planning/Definition

The Test Planning phase consists of the following sequential steps, which are performed iteratively in a RAD or spiral development life cycle:

- Organize requirements,
- Identify Test Scope,

- Identify test runs,
- Determine test data sources,
- Create test work plan,
- Design test plan,
- Prepare requirements checklist,
- Prepare work plan for Test Execution phase.

Testing Risk Assessment

Although ideally 100% of every type of test requirement would be tested, it is often difficult to accomplish this due to time or resource limitations. Therefore, some method of identifying high-risk or error-prone modules needs to be incorporated so that testing efforts are focused on those areas first. This technique is known as risk analysis.

There are four methods for performing risk analysis.

- Judgment and instinct,
- Dollar estimation,
- Identifying and weighting risk attributes,
- Software risk assessment packages.

This section will focus only on judgment and instinct since this is the most commonly used method.

Judgment and instinct based risk analysis is the process of relying on past experience and knowledge of the team members to identify potential trouble spots in a testing project.

The test manager along with assistance from the project manager and various business and testing staff review the entire system requirements with regard to its purpose and complexity to identify those areas that may pose the largest threat to testing and quality assurance. The project is generally compared with past projects of similar magnitude or complexity to help identify error-prone modules.

The objective is to identify how to best utilize the resources to stabilize the most complex or mission critical functions of the system first. This will ensure that if time runs out before all testing is performed, the team can feel comfortable that the system is functional although it may not be perfect.

Once these areas have been identified, they should be documented and a strategy should be devised that focuses the resources on these areas first. Any requirements that did not get tested completely should be tested after the initial release and maintenance releases should be distributed as necessary.

Schedule and Resource Planning

The personnel assigned to a project contribute to the cost, depending on manpower levels. Most projects are resource limited, in that the number of

people with a given skill who are available to the project is limited. The level of manpower available at any stage in a project will affect the time scales, and hence the cost. Resources are assigned to different tasks within the Project Plan. Each of these assigned tasks have a given deadline that needs to be met to ensure the testing portion of the project is on time and within budget.

Test Strategy for IT Audit Controls

In designing a test strategy for IT Audit Controls, the risk factors become the basis or objective of testing. While the test factors themselves are not risks, they are attributes of the software that, if they are wanted and not present, pose a risk to the success of the software, and thus constitute a business risk. There are several IT Audit Controls that have been defined for software development. SQA will include these IT Audit Controls in their testing to ensure the software is compliant with Company policy and standards.

Examples of test factors:

- Authorization,
- File Integrity,
- Audit Trails,
- Service Levels,

- Compliance,
- Performance.

User Acceptance Test Approach

Since each build has been thoroughly system tested throughout the development process, user acceptance testing is focused on verification of business scenarios. These tests should first focus on common business scenarios than more complex scenarios and finally verify the robustness of the system by trying to break it with destructive testing.

Environment(s) Set Up

The test environment preparation form is created during the Analysis phase. The purpose of this form is to help determine what resources will be required for the test environment, when they are needed and who is responsible for setting them up.

This includes any data setup, parameter preparation, hardware, software, special circumstances such as developing stubs, coordination with vendors, etc. The requirement section should also include detailed setup instructions such as time frames for completion, arrival of equipment, etc.

Performance/Stress Test Approach

Performance test approach. The objective of performance testing is to determine whether the system can meet the specific performance criteria:

Determine the performance of the system structure.

- Verify the optimum use of hardware and software.
- Determine response time to on-line use requests.
- Determine transaction processing turnaround time.

Stress test approach. The objective of stress testing is to simulate a production environment for the purpose of determining:

- Normal or above-normal volumes of transactions can be processed through the transaction within the expected time frame.
- The application system is structurally able to process large volumes of data.
- System capacity, including communication lines, has sufficient resources available to meet expected turnaround times.
- People can perform their assigned tasks and maintain the desired turnaround time.

Regression test approach. Regression testing involves assurance that all aspects of an application system remain functional after testing. The introduction of change is the cause of problems in previously tested segments.

- Determine whether systems documentation remains current.
- Determine that system test data and test conditions remain current.
- Determine that previously tested system functions perform properly after changes are introduced into the application system.

Test Strategy and Test Plan

- **Test Strategy.** The objective of testing is to reduce the risks inherent in the system. The test strategy must address the risks and present a process that can rescue those risks.
- **Test Plan.** The objective of the test plan is to describe when and how testing will occur. This test plan will provide background information on the software being tested, the test objectives and risks, as well as on the business functions to be tested and the specific test to be performed.

DESIGN AND DEVELOPMENT

The objective of this phase of the Software Quality Life Cycle is to create the test cases for the project, which are the outputs of this phase.

Test Cases

- **Specifications**—Creation of test design requirements, including purpose, preparation and usage.
- **Cases**—Development of test objectives, including techniques and approaches for validation of the product. Determination of the expected result for each test case.
- **Scripts**—Documentation of the steps to be performed in testing, focusing on the purpose and preparation of procedures; emphasizing entrance and exit criteria.
- **Data**—Development of test inputs, use of data generation tools. Determination of the data set or sub-set needed to ensure a comprehensive test of the system. The ability to determine data that suits boundary value analysis and stress testing requirements.

UAT Training

The UAT Training task is primarily one of coordination in that it must ensure that everything needed for training has been prepared. The coordination normally involves the following tasks:

- Scheduling training dates.
- Notifying the people who should attend the training.
- Obtaining training facilities.

- Reproducing the material in sufficient quantity for all those requiring the material.
- Assisting with the creation of test cases.

Many times training will be provided through manuals or special material delivered to the involved parties. SQA will assist with all of the business training needs.

SYSTEM TESTING

The objective of this phase of the Software Quality Life Cycle is to execute the required different system tests for the project. The main output of this phase is to begin the Final Test Report.

The remainder of this subsection describes the activities that contribute to this phase of the SQLC.

A System Test verifies all business requirements and internal workings of the system to ensure that the system is functionally correct and delivered to the spec/requirements. It is primarily performed through the creation of test conditions and functional checklists. Test conditions are generalized during requirements, and become more specific as the SDLC progresses, leading to the creation of test data for use in evaluating the implemented application system.

- Create a test matrix to prove that the system requirements as documented are the requirements desired by the user (see Table 4).
- Use a checklist prepared specifically for the application to verify that the application's compliance to organizational policies and governmental regulations.
- Determine that the system meets the auditability requirements established by the organization's department of internal auditors.

Table 4. Testing Type Matrix

Testing Type	Testing Level	Responsibility
Informal unit	Unit	Developer
Formal unit	Unit	Test analyst or developer
Environment	System	Developer
Integration	System	Test analyst or developer
System	System	Test analyst or developer
Backup/Recovery	System Installation	Developer or technical and application support
Performance	System	Test analyst or developer
Stress	System	Test analyst or developer
Volume	System	Test analyst or developer
Regression	System	Test analyst or developer
User acceptance	User acceptance	User
Documentation/Procedure	User acceptance	Test analyst or User
Usability	User acceptance	User
Installation	Installation	Test analyst or developer
Parallel	Installation User acceptance	User
Pilot	Installation User acceptance	User
Configuration	Installation	Test analyst

System Test Cycle

System testing should begin in the requirements phase, and continue through every phase of the life cycle into Maintenance and Support.

- Functional Tests.
- Non-Functional Tests.
- Configuration Tests.
- Infrastructure Tests.

Performance Test Cycle

In the most basic terms, the final goal for Performance testing is for business users to consistently have the positive experience they expect. Applications require a performance test strategy that can deliver a plan for non-disruptive growth, continuous availability, and consistent response times—even during peak usage timeframes. In order to provide for these criteria the following three activities in the performance test strategy are employed: Performance Benchmarking, Performance Regression Testing and Performance Tuning/Acceptance.

- ***Performance Benchmarking***—Performance benchmarking is the process of carefully considering and defining the types of performance test scenarios the application will need and then creating, running and analyzing these performance tests against

early versions of the application. Executing these tests provides a set of performance metrics commonly referred to as a performance benchmark. This benchmark will provide the Test Team with points of reference against which prior releases and future performance tests can be compared. This comparison helps us quantify the scalability of the application, and to understand the application's true performance improvements or degradations.

- ***Performance Regression Testing***—The benefit of regression testing is *repeatability*. Performance regression testing is the process by which the application is retested and measured against the earlier benchmark tests to ensure that the application performance hasn't been degraded because of application changes that have been made. A performance regression test is planned/scheduled each time new features are added or expanded, or if new application interfaces or additional web servers are added to the system.
- ***Load Testing, Performance Tuning and Acceptance***—This is the final load-testing phase prior to the official migration of the application to production. In this step, all of the different pieces of the Web application (including the hardware and all load-balancing and software components) are integrated, and application performance

is validated. Different transaction scenarios of real-life usage volumes are emulated, and the scalability of the final configuration is validated. The validation is demonstrated by executing multiple, varying test scenarios against the QA and Production hardware configurations at predicted user and transaction volumes.

Regression Test Cycle

Regression testing requires that a regression test bed (a comprehensive set of reusable system test cases) be available throughout the useful life of the delivered system. A regression test set can be viewed as a miniature model of the production system.

An important consideration when creating a regression test set is keeping the time to run the regression test to a minimum. Having a small database for the test data, and automating the execution and comparison of test results as much as possible can accomplish this.

Maintaining a test bed will not be a trivial effort. As changes are made to the application, some test cases may need to be deleted from the regression test set and other test cases will have to be added. This means that the test data will also have to be updated accordingly. Furthermore, when changing the regression test set, additional considerations will have to be made regarding the remaining test cases. Review those test cases to verify they can

still be run with the new test cases or without the test cases that were eliminated. The regression test bed must be maintained to keep it aligned with the application as the application itself evolves.

Components of a Regression Test Set

- **Test Cases:** A set of test cases, designed to test the application. Normally functional test cases form the core of the regression test bed. These test cases are updated where necessary with each new version of the system.
- **Test Bed:** A test bed that normally contains only the minimum amount of data required satisfying the test cases. A small amount of test data is desirable to keep run times to a minimum and to keep the data at a manageable size for evaluation of results. The test bed may also need to be updated with each new version of the system.
- **Test Results:** A set of test results (outputs) from the last baseline that will be compared to the outputs from running the current version of the system. This means that there are always two sets of test outputs retained: the current version and the previous version.
- **UA Test Scenarios:** The objective of UA Test Scenarios is to determine whether the acceptance criteria have been met in a delivered product. This can be accomplished through reviews and

test execution. User Test Scenarios are created to validate the system.

USER ACCEPTANCE TESTING

The objective of this phase of the Software Quality Life Cycle is to execute the final testing phase for the project. The main output of this phase is to contribute to the Final Test Report.

A **User Acceptance Test** verifies that the system is "fit for use" in production. The User Acceptance Test validates the business process that uses the system - not the system itself. UAT verifies that the interfaces, data migrations, manual processes, and other applications work well with the system so that there is an excellent migration to production and user acceptance of the system.

In practice, there is very little actual testing that occurs in UAT. It is mostly a verification step with a detailed review of requirements and system testing. The bulk of the UAT work is to prepare data migration/conversions and documentation that specifies/validates how the system will work in the production environment with existing forms, invoice/check stock, user roles, existing business processes, etc.

Below are some excellent UAT activities:

- Review System Test Cases and Results.
- Review Final BRD or Use Cases.
- Document User Acceptance Test Cases that test how the system will be used. These are usually organized by user role or business process (include manual process, processes with other systems).
- Set up production security roles.
- Perform data conversions/uploads for production.
- Verify printing of reports/invoices/forms/checks on production equipment.
- Verify that all user apps can co-exist on users PC (installation).
- Perform usability tests to determine if it is acceptable for end users/the business.

Final Test Report

A Final Test Report is designed to define the scope of testing, present the results of testing and draw conclusions and recommendations from those test results.

IMPLEMENTATION

The objective of this phase of the Software Quality Life Cycle is to execute the required different system tests for the project. The main output of this phase is to begin the Software Quality Support Plan.

The implementation phase of the software quality life cycle reviews the testing that was performed throughout the lifecycle of the project. It's a time to assess, evaluate and define the Software Quality Support Plan.

SQA/Test Post Project Assessment

Once the project has gone into production a Post Project Assessment meeting is held. Test Metrics must be analyzed to draw conclusions about the effectiveness of system testing. From this analysis, the appropriate party can take action. The summarized results must be output into a form for presentation that provides an assessment of testing.

The judgmental approach normally expresses the assessment in terms of an opinion of the assessor. The user reaction provides the same type of assessment and normally includes examples that illustrate good or poor testing performance. The problems and compliance to standards approaches normally express the assessment in terms of what has or has not happened; for example, there are a known number of problems, or X standards have been

violated in a test process. The metrics approach provides the assessment in quantitative terms that show quantitatively the goodness or badness of the test process.

Evaluate Metrics

Testing metrics are relationships that show a high positive correlation to that which is being measured. Metrics are used in almost all disciplines as a basis of performing an assessment of the effectiveness of some process. Some of the common testing metrics used to evaluate are:

- Number of Tests Run.
- Testing Costs.
- Defects Logged.
- Test Automation.

Metrics evaluation is performed to make the application system testing more effective. This is done by a careful analysis of the results of testing, and then taking action to correct identified weaknesses.

Quality Processes Assessment

We review existing software testing processes and their impact on risk and quality. We then summarize our findings and present recommendations. This is the first step in Software Quality process improvement.

Testing Transition Planning

This is the process of preparing an ongoing test plan for the application after it has moved into production. Usually, the defect tracking software will need to be updated to specify enhancement requests and developers and testers that supported the application will move on leaving a much smaller team behind to support testing for the application. All testing work provided by other team members now need to be transitioned to the support team.

SQA/Testing Post Production Training

This is the process of continuing SQA support for the application in production.

- When to call SQA for supplemental testing help.
- How to train BAs to test and meet IT Audit Control Objectives.
- How to train BAs in technical testing for the application.

Update Project Docs/Archival of Documentation

Documentation of the test process records both the tests to be performed and the results of those tests. Documentation is an integral part of the formalization of testing. Test documentation is important for conducting the test and for the reuse of the test program during maintenance. Test documentation should be continually updated.

The test documentation is a part of the systems documentation. Therefore, it should be stored with the system documentation. It should be clearly identified as test documentation. It is recommended that documentation be stored by type. The test plan should be stored in one location and the test results in another. Each should include a table of contents outlining each piece of information in the documentation and where that information is located.

MAINTENANCE AND SUPPORT

The objective of this phase of the Software Quality Life Cycle is to execute the required different system tests for the project. The main output of this phase is to contribute to the Software Quality Plan.

The remainder of this subsection describes the activities that contribute to this phase of the SQLC.

Modifications to a production system are generated from a number of different sources such as regulation changes, defect corrections, enhancements, etc. The cost to maintain a system over a span of eight to ten years has been estimated to be two to three times the cost of its initial development effort. The effort to test these changes is referred to as maintenance testing and is generally one of the most overlooked and

underestimated types of testing. Maintenance testing may also be referred to as regression testing since regression testing can make up a large portion of the effort.

Many modifications may appear to be minor from a development standpoint. Therefore, maintenance releases are generally not treated with the same discipline as a new development project. Without following structured disciplines such as analysis, design, and testing, many maintenance releases actually cause new problems in the system. Because patches tend to erode the stability of the system, the ratio of testing to programming in a maintenance environment is usually higher than in new development. It is important that all modifications be thoroughly tested prior to maintenance release.

Maintenance testing often requires the test analyst to have a good understanding of the system as a whole and how all the functions interact with one another. This helps the test analyst develop a comprehensive test plan that includes all possible areas that may be affected. Since this test plan may become very large, automated tools are often used during test execution.

A recurring theme in software maintenance is the importance of documentation and the general lack of good documentation. Documentation is important in a maintenance environment for three reasons and falls into two classes.

Reasons for documentation:

- Acts as a communication medium among members of the development team.
- Acts as an information repository for the maintenance team.
- Tells the users how to use and administer the system.

Classes of documentation:

- Process documentation.
- Product documentation.

Process documentation records the development and maintenance of a system. It includes plans; schedules and process standards and is used to manage the development process.

Product documentation includes system documentation from the perspective of developers, maintainers and users. It is used for system development, maintenance and user operation.

Regression Testing

Regression testing should be used when there is a high risk that new changes may affect unchanged areas of the application system. In maintenance, regression testing should be conducted if the potential loss that could occur due to affecting an unchanged portion is very high. The determination as to whether to conduct regression testing should be based

upon the significance of the loss that could occur due to improperly tested applications.

Data Security Testing

Security is a protection system that is needed for both secure confidential information and for competitive purposes to assure third parties their data will be protected. The amount of security provided will be dependent upon the risks associated with compromise or loss of information. Protecting the confidentiality of the information is designed to protect the resources of the organization. Security testing is designed to evaluate the adequacy of the protective procedures and countermeasures.

System Owners Application Accountabilities

System Owners are responsible for implementing and supervising processes related to the operation and use of automation systems.

- System Owners responsibilities include:
  - Security Management
  - Data Privacy Promise
  - Data Integrity
  - Uncontrolled Applications/Databases
  - Disaster Recovery and Contingency Plans

- System Backup and Recovery

- The SQA Department will provide training to all identified System Owners.
- The IT Personnel will support the Operations accountabilities with IT processes and mechanisms

IT Audit Support & Compliance

The SQA department works closely with the Clients Corporate IT Audit department to ensure that there's an awareness of Audit Control Objectives during new application development and existing application maintenance.

Data Privacy Compliance

The security and data privacy department handles this.

Software Quality Support Plan

Ongoing plan for how the application in maintenance will meet software quality goals established in the original Software Quality Plan and Test Plan.

Chapter 6

AGILE SCRUM

AN OVERVIEW

Many projects extend to much more time than projected and cost might increase than planned to companies. Organizations targeting to improve their software development processes are now finding that Agile method can help their Enterprise more reliably deliver software quickly, iteratively and with a feature set that hits that mark.

While agile development method has different "flavors", Scrum is one process of them for implementing Agile technology. This section will discuss the Agile Scrum process along with variants of Scrum that can be used to help in improving organizations software releases.

DEFINITION OF SCRUM

Scrum is a procedure of implementing agile software development, in which functionalities are delivered in 30 day sprints. Scrum is a name that is derived from Rugby, in which sprint is the process of stopping play, then

dynamically playing until the sprint ends and a new one begins. The same idea applies here, where the team defines the requirements for a 30 day sprint and work on them continuously for 30 days without being diverted by other things or things have been re-prioritized. A definite functionality is not known as it is being completed until it is analyzed, designed, coded, tested, re-factored and documented. Once the 30 day sprint is completed, most functionality defined in the 30-day sprint has to be completed. If for some reason, part of it is not completed (because of being underestimated), the uncompleted features will be moved to next sprint. A sprint is taken as passed if all the completed features have high quality and can be moved into production (or beta) after completing the sprint cycle.

UNDERSTANDING SCRUM RULES

In Agile development to make sure scrum is successful, teams must have commitment from the highest stake holders to the individuals doing the work and every individual will have to go after the rules. Some underlined rules for the Scrum to be successful:

- Get Number of Hours Commitment straightforward – Before starting an Agile development 30-day sprint, every team member has to commit to a certain number of hours for the 30 day sprint.

- Gather Requirements / Estimates straightforward—The Product Manager will specify the sprint target and the team will gather the requirements and provide a calculation up front. Once the estimates are done, the requirements are prioritized and only the ones that will fit into the sprint are worked on (based on estimated hours of all tasks vs. hours committed to by team members).
- Daily Time Card Entry—Every person in the team accepts to enter their actual hours worked, estimated hours to complete the task and hours remaining to complete the task every day.
- Build the Code Daily—Every programmer will have to check code in daily or more frequently if possible. The checked in code must be compliable. An automated process scripts will create daily builds to prevent manual merging of code and allowing the Quality Assurance Engineer to test tasks of the sprint.
- New Requirements should not be added for a Sprint—Any new Requirements should not be entered into the sprint unless all tasks of the sprint are finished. Management and other parties that are not directly involved in completing the features for the sprint will not be allowed to give actions to add requirements on tasks not included in the sprint. When an emergency task is required by

management, the sprint has to be abandon and a new sprint has to start with the new task set.

- Daily Scrum Meetings should be kept short—Every day scrum meetings will be conducted to decide what tasks have been done since the last Scrum meeting, what tasks will have to be done in the next Scrum meeting and what hinders can come in the way of any person on the team. The Daily Scrum meeting should be designed to be completed in 15 minutes. If at all it takes 30 minutes, it is also ok, but it should not extend longer than that without a solid business reason for it. Team members that attend to Daily Scrum meetings in late are required to pay the Scrum Master a $1 fine that goes towards the team development activity.
- Code Inspections are Paramount—When completing a particular task, the programmer should demonstrate the feature to the team and show the code that delivers the feature. The team should have to inspect the code for re-usability, cleanness, and devotion to established coding standards.

SCRUM EXPLAINED

Organizing a Scrum development needs a key change in how teams work together. In general Waterfall development, teams generally will have a project sponsor, a project manager, analysts, designers, programmers, testers, and documentation specialists. Every team member will have definite duties which generally do no overlap and they will have a definite reporting arrangement (most of the team members report will to the project manager).

Scrum starts with an 8 hour Scrum Kickoff Meeting. The Scrum Kickoff meeting is divided into two 4-hour parts, in which first team members will specify what tasks, will be needed for the 30 day sprint. Then the last 4 hours will be used to provide some rough estimation for the items identified for the sprint. Then if the estimations exceed the available resources, the tasks are prioritized and less important tasks are removed from the sprint. The Scrum will use a time-box approach which is an important component, in which meetings and events will have a definite time period (e.g., no more than 8 hours for the kickoff meeting) and it is strictly enforced by time-box. When the tasks are entered in for the 30-day sprint, no changes will be allowed (new tasks will not be entered until the next sprint). When calculating tasks for a sprint, the calculations have to include time for analysis, design, coding,

testing, re-factoring, and documentation. A task is not said to be completed until all those activities are completed.

Every day, a Daily Scrum Meeting will be conducted to specify how the tasks are progressing. The meeting is no longer than 15 minutes, and each team member is asked three questions:

- What did the team member accomplish since the last Daily Scrum Meeting?
- What will the team member do before the next Daily Scrum Meeting?
- Is there anything that is stopping member's progress (and remedies are discussed)?

From a programmer's point of view, Scrum development is a new pattern which is very empowering but it does need them to follow specific rules:

- The source code is only checked out for the time needed to complete a task. No exceptions. Most of the source code will be checked in daily, as most tasks are separated into small task sets.
- Time for the task must be entered daily. For every task, members will have calculated hours, actual hours and hours remaining to complete the task. This information has to be updated at the end of

every day so that the Scrum Master able to verify if the release status is moving as needed.

- Programmers will not be allowed to be dragged off on different projects; they have to stick to the tasks they have been assigned to, for the sprint.
- All the team members have to attend the Daily Scrum Meeting and have to be on time.
- Every day the Source Code will be compiled and deployed to a test server. Teams can use automated build scripts/tools to speed up the whole process. Automated test cases must be run against the daily releases to find any defects introduced by the release.

When a Scrum 30-day sprint is completed, all the tasks that were completed can be then moved to a QA, beta or production environment. Following the sprint, there is a retrospective meeting in which team members will discuss and document the things that went well and things that can be improved upon in the next sprint.

Team Composition

- Management of the Scrum development needs a major coordination on how teams work together. Whereas In traditional Waterfall software development method, teams will normally have a project

sponsor, a project manager, analysts, designers, programmers, testers, and documentation specialists to develop a project and every team member will have some specific jobs which are generally do not overlap and they will have a definite reporting method (most of the team members generally report to the project manager).

- Using Scrum, teams have just three roles in a team and it may be normally limited to seven or less individuals (Though, there can be multiple Scrum teams in sets of 7 or less).
- Product Owner – Product owner is the person who is responsible to identify and prioritize the tasks that will appear in a 30-day sprint. This person may be normally the Product Manager, CTO, in some cases the CEO, or some other high level stakeholder that eventually is responsible for planning the roadmap of their product. But, before a sprint begins, the Product Owner communicates the target of the sprint to the team and what tasks will have to be analyzed for the release. But, this does not mean that all the desired tasks will be completed in the sprint, the team calculates and organizes the items priority wise for the sprint (during the Sprint Planning sessions), and only the tasks that can fit in the sprint will be done.

- Scrum Master—Scrum Master is similar like Project Manager in Waterfall development environments, but he does not manage the team deliverables at micro level. But, this person is responsible for ensuring that the 30 day sprint is not exceeding the time frame, no new tasks will be added to the sprint, code verifications happen, and checking everyone plays by the rules. The Scrum Master coordinates and organizes the daily sprint meetings. The Scrum Master is not a task master; He is a leader that he will empower the team members to complete the assigned tasks and to help eliminate obstacles that slow down the team.
- The Team—In general Waterfall development, a team consists of analysts, designers, testers and documentation specialists. But in Scrum, every team member is empowered and expected to self-manage themselves and to participate in all jobs will be needed to deliver a task. This will also include analysis, design, coding, testing and documentation. The Team is responsible for staying focused on assigned tasks, taking help from other members as they encounter obstacles, completely testing their code, refactoring code, logging their time daily (including estimated time remaining on

each task), and for checking in their code every day or more often if possible. Repeated.

It is impractical to guess that The Team will handle quality assurance and documentation well. Team composition has to be improved to include two additional roles:

- Software Quality Engineer—is responsible for the quality of the sprint. In general programmers do not test the code with the same approach as a Software Quality Engineer (SQE) does. Once definite requirements are defined, the SQE develops a set of test cases (manual or automated) to test each requirement completely. So before coding starts, the test cases will be available to the programmers on the version control tool. The programmers will be expected to run each test case before marking development as being complete. When a requirement is marked as being complete, the SQE is responsible for executing the test cases again to ensure they all will get pass. The SQE also runs weekly backward test cases to ensure that the legacy functionalities are not being missed by the release. But, the SQE has been developed automated test cases for reverse tests, and they should run daily or more frequently, as required. The SQE will not wait until the end of the sprint to start

testing the product; they will start testing once a task is completed. At the end of the sprint, all testing has been completed and reverse engineering has been run frequently against the finished tasks.

- Documentation Specialist—He is the one, who is responsible for creating User Guides, Administrator Guides and other training materials in terms of documentation. In general programmers do not always have the written communication skills to write documentation in a way that a layman can read and understand it; that is reason it is important to have a separate resource for the documentation purpose. When a requirement has been completed testing by the SQE, the DS starts the documentation of that requirement. The DS will not wait until the end of the sprint to start documentation, but the end of the sprint will include all completed documentation.

The 30-Day Sprints

Agile development method differs from standard Waterfall development method in which the development will have a smaller time limit with a smaller task set. Agile Scrum will put releases into QA, Beta or Production for every 30 days (called 30 day sprints). Once completing a sprint, the software product can be moved to QA, Beta or production (if

production-ready) or can move into another 30 day sprint to develop additional tasks.

Agile Scrum and Pragmatic Agile Development (PAD)

A much intense version of Agile scrum is discussed further below which has been analyzed to be more efficient methodology for most organization structures. This model differs from a traditional version of scrum as some changes have to be made to Agile Scrum to work well for the development environment. This version of Scrum is called as Pragmatic Agile Development (PAD) and varies from a more traditional Scrum development in the below ways:

1. Scrum Planning—The planning for a forthcoming sprint will be accomplished in 1 day in general Scrum, But in PAD the planning limits a week. The reason for this is that, more detailed requirements are written in this than a traditional Scrum
2. User Descriptions vs. Specifications—The requirements will be written on index cards (called User Stories) and does not contain any samples or thorough explanations of the task set in general Scrum. Whereas in PAD, more time is spent writing the

requirement specifications with samples to ensure that the time is well spent on the task, to reduce the rework.

3. 30-Day Sprints – The development is completed in 30 calendar days in general Scrum. Whereas in PAD, development is completed in 30 working days and this excludes holidays. So this gives the teams more evenly distributed sprints.

4. Team Composition – The developers will be expected to perform all jobs (analysis, design, coding, test case development, execution, and documentation) in general Scrum. Whereas in PAD developers will help with analysis and design and will do all the coding. Teams have specialized team members (Software Quality Engineers) for test case development and specialized team members for documentation. It's done this way because general experiences of some people have shown that specialists are needed for these specific areas.

The target of software development has to deliver software rapidly and with good quality, so tuning a methodology to meet organizations needs makes sense.

Product Backlog

Product Backlog includes items like, existing clients requesting new features or team coming up with new features that make the product more marketable. Whereas in the PAD Scrum Planning week, team sets the goal for the sprint and assign priorities to product backlog items to specify which ones will fit within the sprint.

Scrum Kickoff and Planning Week

According to the traditional Scrum development, teams have only one planning day for the sprint and requirements are noted on index cards (called User Stories). So it is believed that this will not be enough time or details to deliver quality tasks, as we will have to show that taking the time to completing the detail for the task saves time, when the client (or internal team) receives the task, it takes less re-work. So it is advised that a week is dedicated to planning.

Every delivery (or sprint) will start with a PAD Planning Week. The very first day of the PAD Planning week will start with defining the goal for the sprint and identifying tasks team wishes to have in the delivery (from the product backlog), in prioritized way.

The releases are done in 30-day sprints in traditional Scrum. The 30 days are calendar days, including holidays and weekends, this will equal to 19

to 23 working days. Sprints are best implemented in 30 working day sprints (excluding holidays), as this gives more evenly distributed sprints for the development. Generally the 30-day sprint starts once the PAD Planning week finished.

The first day of the PAD Planning Week, every team member will identify the number of hours they can give to the sprint, allowing team to decide the maximum quickness for the sprint to be completed in terms of hours (maximum hours). Once the maximum hours are known, what tasks fit in the sprint can be decided. So when the high level features are recognized in the first day of PAD Planning, specific work order numbers are assigned to each high level task and assign a set of work orders to each team member.

The team members will spend the week defining the complete requirements for their assigned work orders.

Note: Work Orders are simply a hard copy document of each functional task.

- **User Interface Design Guidelines**

  It will be critical for the team to define its user interface styles in a style guide and ensure all team members follow those standards, as there will be multiple team members creating requirements.

Decomposing Work Orders

Team is forced to make hard decisions about the features that can fit into the sprint, since the sprints are time framed to 30 working development days. Once the work order for a task of the sprint is assigned to team members, they should divide the task into multiple work orders so that they can be arranged in precedence of specific pieces of the tasks.

For example, let us assume that the team is redesigning a company's user interface for a more pleasing look and feel, and the visuals are the most important issue for the sprint. It includes features like the screens should be changed to be more user-friendly, and there are some issues that need to be addressed (like prompting to save changes when changes are made and they switch between tabs on the screen). In this case, the requirement should be divided into two separate work orders (one for the visuals and another for the tab order). By doing this, it allows team to prioritize the tab switching lower than the aesthetics.

The 30-Day Sprint and the Daily Scrum Meeting

- **Sprints**

Agile software development varies from standard Waterfall development in which development has a less time limit with a smaller task

set. Agile Scrum develops and delivers releases every 30 days (called 30 day sprints). The 30 days is 30 calendar days in the traditional development of Scrum. It has been analyzed that 30 working days will work better.

The application can be moved to production if it is ready for production on completing a sprint, or extend to another 30-day sprint to develop additional tasks. In the ending of the PAD Planning Week, team should have a group of detailed requirements and calculations and the Scrum Master will create a project plan that includes each task which is represented by a work order and the individual assignments.

The task items that appear in the project plan for the sprint are referred to the Sprint Backlog.

In the 30-day sprint every task should have a priority order and should be worked in priority order so that if the any team falls behind, team can ensure that the highest priority task items make it into the sprint. So tasks those are not completed at the end of the sprint will be reprioritized for possible inclusion in the next sprint.

Best Practices for 30-Day Sprints

- **Creating the Test Cases before Coding Begins**

It is significant to create a set of test cases before coding begins, as each task is documented. Developers should always execute all the prepared test

cases before making the task available to the Software Quality Engineer(s) for testing purpose. This method will hugely reduce flaws count and significantly pace up the quality assurance testing.

- **Daily Code Builds**

Each developer should check in their code into their source control system if their code is compiled successfully each day. And if any of the related SQL scripts changed, they also have to be checked in to the source control system. For all this, it is better to use an automated build tool that will run at the end of the day and will do a GET on all source code and SQL Scripts changed code. It will compile the code into DLLs and execute the SQL scripts to upgrade the database. It assures each team member has a new build on QA server daily so that the teams Software Quality Engineer(s) will test new features and run reverse tests when required. There are so many tools available for automatic builds; one of them is Automated Build Studio by Automated QA

- **Code Inspection**

The code will not be taken as complete until the coding is completed, unit tested completely, all prepared test cases have been executed successfully, code has been re-factored if needed, and technical documentation will be written when required. The team has to do a code inspection by reviewing the

code in the source control system related to the task once indicated that code is complete for a task. The review of the code should look to follow the standards, finding of logic errors or performance problems, and reusability. If the code inspection finds failures, defects there should be created and assigned—allowing the developer to fix the issues before the code is tested by the Software Quality Engineer(s).

- **Daily Hours Entry**

Every team member has to enter the hours they spent on each task during every day. It is important to enter the percentage complete of task or estimated remaining hours to finish the task (this is the preferred method) when entering the time. Sprint team will know how many hours remain for all tasks and can determine if members are progressing on a track to finish the sprint with all the desired features, by entering the estimated remaining hours for each task worked on every day. From a metrics perspective, inspect burn down charts daily.

A burn down chart is simply a chart that shows day-by-day the number of estimated hours, actual hours and estimated hours remaining. Teams will see the estimated hours move downwards and it will be on pace so all the committed work is achieved in the sprint as the sprint progresses.

- **Daily Scrum Meeting**

Each team will have to conduct a Daily Scrum Meeting every day. This daily meeting is restricted to 15 minutes and each team member is asked three questions, in the traditional version of Scrum:

- "What did the team member do yesterday?"
- "What will the team member do today?"
- "Are there any hindrances or anything stopping the team member's progress?"

But in general 15 minutes is not always enough time to have a good dialog and a meaningful meeting. Most of the time, everyone will complete this in 15 minutes, but most regularly it will take about 30 to 45 minutes. To speed this up, each team member is asked to post a summary of what they did yesterday and a summary of what they plan to do tomorrow in a daily discussion forum that is automatically distributed to all team members. It will allow the team members to spend the Daily Scrum meeting talking about obstacles, design decisions, and dependencies to progress.

Reporting and Metrics

Teams will want to collect metrics that answer 2 questions from metrics point of view in Agile:

1. Are the sprint requirements going to be completed by the end of the sprint? Teams can use burn down charts that show the number of hours remaining for each day of the sprint. As the sprint moves on, the chart should incrementally move downwards, showing whether the team will be done with all requirements at the end of the sprint.
2. Do the requirements completed in the sprint have high quality? Test cases and failure statistics can be used for this. Test Case statistics will indicate if team has thoroughly tested the software and failure statistics will alert as to the quality of the software.

Spreadsheets or an Application Lifecycle Management (ALM) tool can be used to generate the information required to answer these critical questions while collecting statistics and metrics. While a spreadsheet will cost less (no purchase of a tool required) but it will require that someone has to key the data into the spreadsheet daily and keep it updated. ALM tools will prevent ambiguous data entry and can give a more inclusive statistics and a better view of status.

Companies, who are short of budget for an ALM tool, can collect statistics and enter them into a spreadsheet. The trickiest problem with this approach is that the teams have to keep the spreadsheet up to date and teams need to have a way of collecting the statistics. For example, every day each

member has to enter how many hours they have remaining on each task on which they are working on. Teams also have to manually keep tracking the number of test cases, the status of them, the number of defects and the status of them.

Capture metrics by an application management tool. Team members need not be asked each day for their remaining hours by having an ALM tool to provide statistics. They will enter their time in the ALM tool daily and it will automatically update the statistics. The ALM tool can also be used by QA members to manage requirements, test cases and laws, and statistics will always be easy to check.

There are so many ALM tools available in the market; some screen shots that show how Software Planner creates statistics that give the metrics required for Agile sprints are shown below.

1. Burn Down Charts.

Burn down charts will be available without reentry of data. The data is pulled from the timesheets entered every day by team members. A major advantage is that teams can easily catch between sprints, comparing with one another.

The below example explains a tool namely version one that describes a burn down chart of each requirement and it records how

many hours should be remaining day-by-day and the actual hours remaining day by day. By reviewing the graph daily, teams determine if they are moving towards the plan or not.

Below is an example of how user stories for a sprint are tracked using version one tool. Each requirement is divided into the lowest modular level possible and is considered to be one story. Development and testing efforts are assigned to each story and the required resources are allocated to the stories. Coding testing and UAT review of all stories on the Taskboard fulfills the completion of one sprint cycle. Stories might in turn be deferred to next sprint or cancelled due to various reasons like change in requirements. Completion of all stories by all teams in a sprint calls for regression which and end of a regression run with all the identified defects fixed marks the end of one sprint cycle.

The next sprint cycle starts here and follows and repeats the same cycle as the previous sprint cycle.

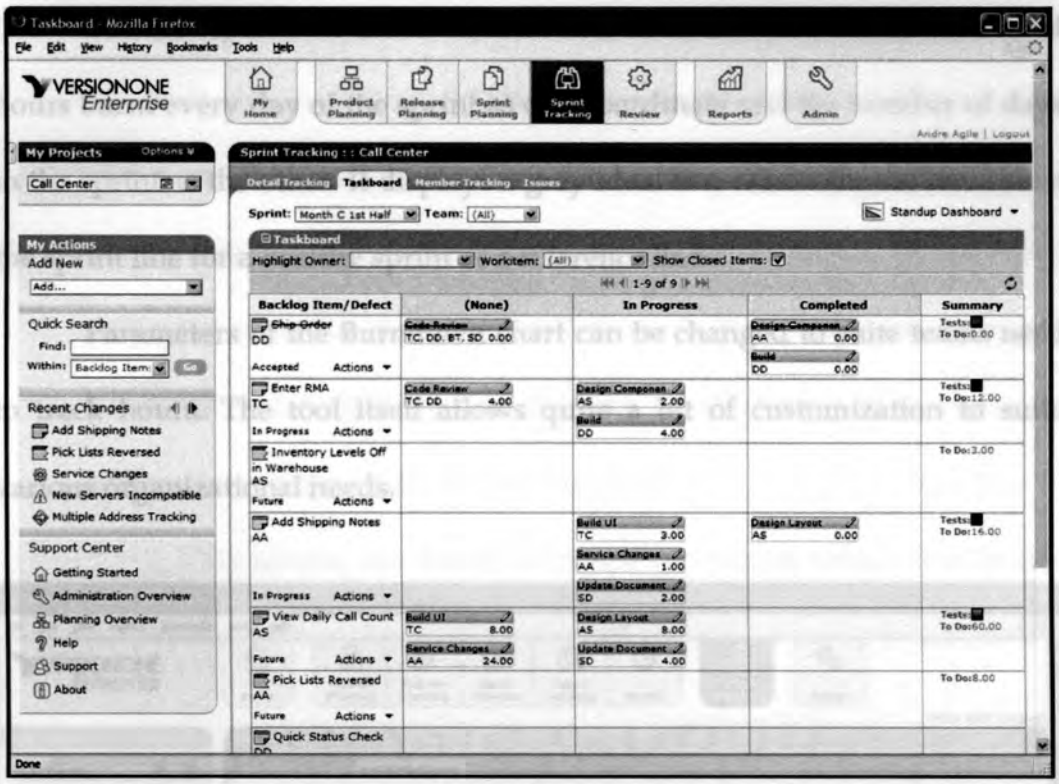

Figure 4. Version One Sprint Taskboard Screenshot

Version one allows the ability to track the overall progress of each sprint from the perspective of the entire project and also each individual team involved in the project.

This requires the team members to burn (track) hours at the end of each work day to depict the accurate progress of the teams work and that of overall sprint. This type of dashboard called Burndown chart is very useful in tracking the overall health of the sprint when there are more than one teams working across the same project.

Burndown chart is a two dimensional chart that takes total number of hours burnt every day of the sprint as one coordinate and the number of days in the sprint as the other. It displays a grey ideal line which should have been the sprint line for a healthy sprint as a reference line.

Parameters of the Burndown chart can be changed to suite teams need to track hours. The tool itself allows quite a bit of customization to suite various organizational needs.

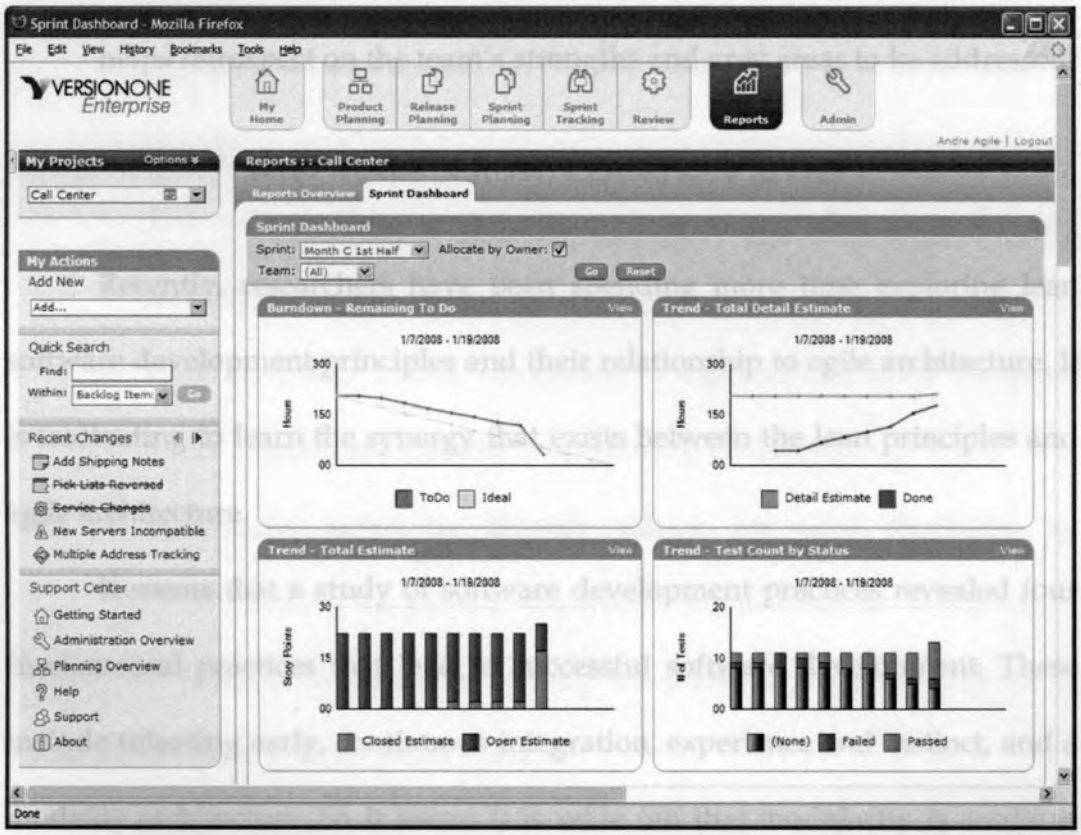

Figure 5. Version One Burndown Charts Screens

2. Quality Assurance Analysis

Teams can always see quickly how many test cases are running for each sprint, and how many passed and failed, every day. There is always an option also to trend defects every day to ensure that as the sprint is ending, the quality of the product release is high. And one more key benefit is that it allows quick checking on the testing status between sprints, comparing with one another.

This allows the teams to check on overall team's health and helps retrospect on the team's strengths and grey areas to be addressed.

WHY AGILE SCRUM/LEAN PRINCIPLES

Recently, researchers have been spending more time exploring lean software development principles and their relationship to agile architecture. It is fascinating to learn the synergy that exists between the lean principles and agile architecture.

It seems that a study of software development practices revealed four fundamental practices that lead to successful software development. These include releasing early, continuous integration, experience and instinct, and a modular architecture. So it seems it is wide felt that modularity is a critical

component of agile architecture. But the thrust of the discussion comes later on in Chapter Two, when speaking of deferring commitment.

Agile architecture focuses on exploring how modularity helps increase architectural agility. Modularity is a required (and to this point, missing) aspect of agile architecture. The basis for this claim follows:

- Architecture is design, but not all design is architecture.
- Design is architecture if the design is architecturally significant. That is, if it's hard to change.
- The goal of architecture is to eliminate the impact and cost of change.
- The way to eliminate the impact and cost of change is through flexibility.
- With flexibility comes complexity. Organizations must therefore strive to increase flexibility while taming complexity.
- Modularity helps us identify where we need flexibility by understanding the joints of the system where flexibility is necessary.

Without modularity, it is hard to identify the joints so it's more difficult to understand where flexibility is needed.

Another reason Agile stresses on modularity is that this kind of breakdown of tasks reduces any dependencies of resources on each other. It

makes it easier for the resources to move onto a different task in case of a deferment of one which in turn cuts the slack time and incorporating lean behavior.

Deferring commitment is another factor that adds lean behavior to this type of methodologies in general. Deferring commitment focuses on two fundamental factors - reversibility and irreversibility. In general, reversible decisions are those that can be changed while irreversible decisions are those that cannot be changed. Teams should strive to make irreversible decisions at the last responsible moment. As it is at this moment teams possess the most knowledge which allows them to choose the most viable option. But teams are also advised and specially quoted that:

"First and foremost, teams should try to make most decisions reversible, so they can be made and then easily changed." This in turn also accommodates changing requirements which is not very uncommon in most business models.

For most, this captures the essence of eliminating architecture. If companies are able to take a seemingly architecturally significant challenge and make it reversible, then they have effectively minimized the impact and cost of change to a point where change is no longer architecturally significant.

Chapter 7

QUALITY CENTER

Mercury Quality Center is a web-based test management tool. It gives users a centralized control over the entire testing life cycle. It gives an easy interface to manage and organize activities like Requirements coverage, Test Case Management, Test Execution Reporting, Defect Management, and Test Automation. All these activities are provided from a single tool, which is web-based and can be accessed from anywhere hence, making the task of the testers and managers easy.

QUALITY CENTER COMPONENTS

Mercury Quality Center can be divided into two parts:

- Site Administrator Bin.
- Quality Center Bin.

Site Administration Bin: It is the starting point for the usage of Mercury Quality Center. This part is used for all the administrative activities. Password for site admin is defined during the installation so make sure that

the user remembers the password during installation. From this part of Mercury Quality Center, we generally do the following activities:

- Creating the projects.
- Assigning users to the projects.
- Creating specific roles.
- Configuring QTP or Winrunner scripts to use from Mercury Quality Center.
- Configuring the mail servers.
- Verifying licensing information.
- Information about database.

Quality Center Bin: This part of Mercury Quality Center gives functionality of almost everything that as a tester or test manager the user need to do in user's day to day activity apart from execution. This is the most common interface used by the customers or users. In this part, we generally do the following activities:

- Creating test plans.
- Defining requirements.
- Creating test cases.
- Creating test labs.
- Associating requirements with defects in essence.

Mercury Quality Center is installed as a service in Microsoft windows environment. Before start working on it, make sure that Mercury Quality Center service is running.

As soon as the user accesses the application, the first screen is a login screen where the user needs to provide administrator credentials which were used during the installation of Mercury Quality Center. Once the user is logged on to the SABin, the user can perform all the administrative tasks mentioned above.

Define user's projects in SABin. Quality Center provides the role based accessed to the Projects. For example, A Test Manager can create projects and Test Lead can prepare test plans and tester can write the test cases. This role based access makes it very easy to control access to various artifacts of the project and also distribution of responsibility among team members.

TABS IN QUALITY CENTER

Following four things can be managed in Mercury Quality Center:

- Requirements,
- Test Plan,
- Test Lab,
- Defects.

Once the user have created a project in SABin, Now log on to QCBin by providing users credentials and access the project that the user have created. Here, the user will notice different tabs for Requirements, Test Plan, Test Lab and Defects.

Requirements Tab: under this tab the user can organize the project requirements. The user can also create folder hierarchy to represent various features in user's project. This can be accomplished by just right-clicking and choosing appropriate options.

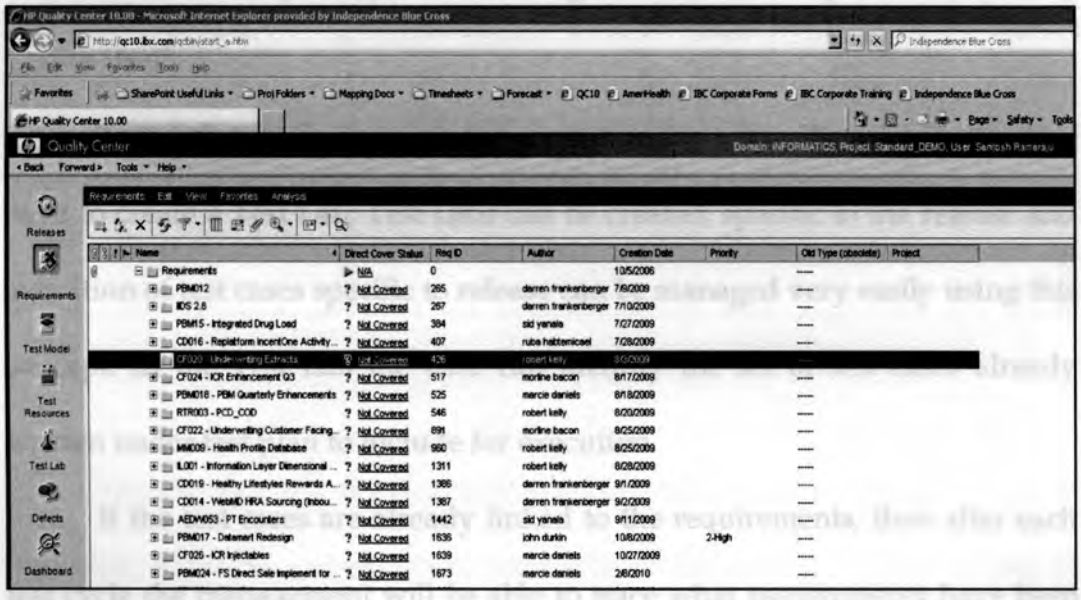

Figure 6. Sample Use of Requirement Tab

Test Plan Tab: will have information about the test cases. These test cases can also be mapped to requirements created in the earlier steps, thus

makes foundation for the traceability metrics. Each requirement can be mapped to one or more than one test cases.

After creating new test case the user will see in the left hand pane. The right hand pane will have tabs for writing the steps, mapping to requirements, description, expected result etc. Every test case will have steps and for every step the user can specify the expected behavior.

The test cases written here can also be linked to the QTP or Winrunner Scripts. This way, it is providing the user better management for the automation and capability of executing automation scripts from Quality Center itself.

Test Lab tab: To manage test execution for a specific release, the user have to create a Test Lab. Test Labs can be created, specific to the release and execution of test cases specific to release can be managed very easily using this concept. In the Test Lab the user can identify the set of test cases already written under test plan to include for execution.

If the test cases are already linked to the requirements, then after each test cycle the management will be able to trace what requirements have been tested.

When the user chooses the option of manual test execution, a window will open up containing the steps to execute. These steps are executed and

after every step the user can specify whether it is passed or not. Quality Center also allows parameterized manual test execution, where some of the default parameters like username; password etc. can automatically be read during the manual execution.

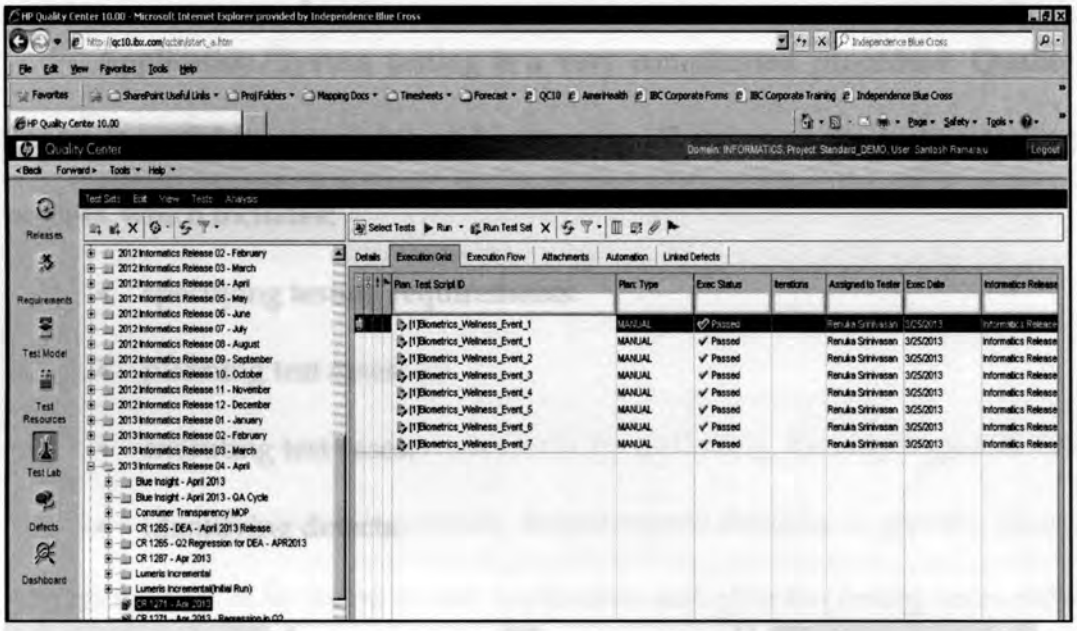

Figure 7. Text Lab Tab

Defects Tab: If any defects are encountered during the failure of any of the steps, it will be automatically logged in to the defect tracking system of Quality Center. Defects can also be submitted by directly browsing to the Defects tab and creating a new defect which can further be linked to the test case execution from test lab.

Report generation is one of the most important parts of the test management process. Once the user is done with planning and execution, reporting comes into play. Mercury Quality Center provides a very good reporting feature by providing certain pre-defined reports and also capability to create users own reports.

Application/System testing is a very complicated procedure. Quality Center is useful in organizing and managing all stages of the system testing process, which includes:

- specifying testing requirements.
- planning test cases.
- executing test cases.
- identifying defects.

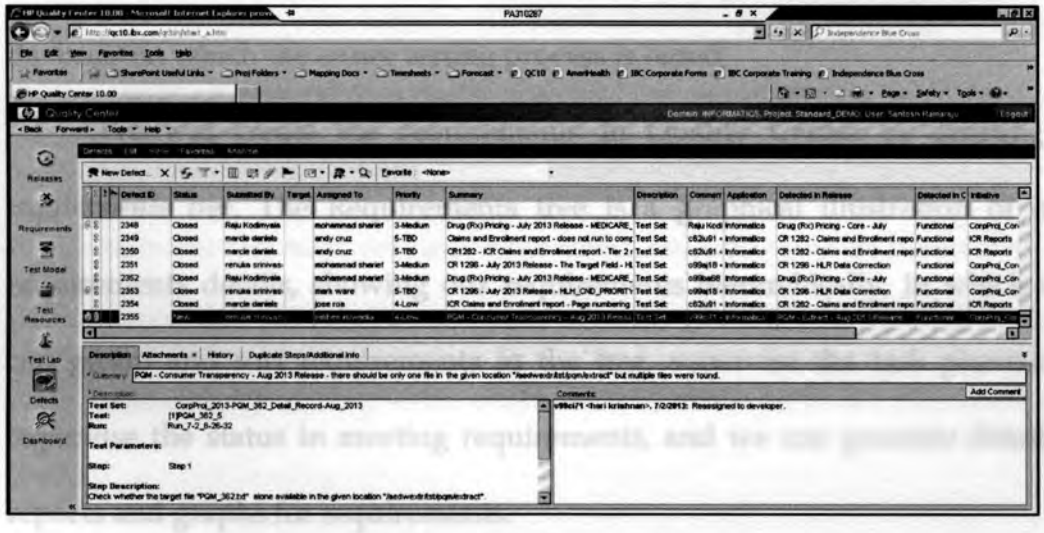

Figure 8. Defects Tab

THE QUALITY CENTER TESTING PROCESS

Quality Center gives an organized structure for testing applications before they are delivered. Because test plans advance with new or changed system requirements, we need to have central data storage for organizing and managing the testing processes. Quality Center process will guide the user through the specifying requirements, planning test cases, executing test cases, and tracking the defects, stages of the testing procedure. The Quality Center testing process consists of four stages.

Specifying Testing Requirements

Initially the testing process starts by gathering the requirements for testing in the Requirements module. Requirements describe in specific about, what is needed to be tested in our application and give the testing team with the basis on which the entire testing process is based.

We will create the requirements in Quality Center by creating a *requirements tree*. The Requirements tree is a graphical illustration of our requirements details, showing our requirements hierarchically, in which we can group and sort requirements in the tree, supervise the task provision, supervise the status in meeting requirements, and we can generate detailed reports and graphs for requirements.

Once we created test cases in the Test Plan module, we can link requirements to test cases. Later, we can start logging flaws; and to manage easily we can also link requirements with defects. In this way, we can always keep checking of our testing needs at all phases of the testing process. So if a testing requirement changes, we can immediately track and identify which test cases and defects are affected, and who is responsible for them.

Planning Test Cases

After we finish requirements gathering phase, we need to decide our testing targets. For this, we need to study our system, application environment, and testing process to summarize the testing plan for achieving our targets.

In the next step, after we conclude on our testing goals, we need to prepare a *test plan tree,* which will hierarchically divides our system into testing units, or *subjects*. For each subject in the test plan tree, we define test cases that contain one or many steps.

In every test step, we need to mention the actions to be performed on our application and the expected result. We can also increase the flexibility of a test case step by adding extra parameters to understand the requirements easily.

We can keep track of the relationship between our test cases and our requirements, by adding linking them. By linking them, we can ensure compliance with our requirements throughout the testing process.

After we finished the designing of our test cases, we can decide which test cases to be executed automatically. When we made a test case to be auto executed, we can generate a test script and then finish it using other Mercury testing tools (for example, QuickTest Professional or WinRunner).

Executing Test Cases

Executing the test cases is the core part of the testing process. As our application changes, we need to run manual and automated test cases in our project to identify defects and assess quality.

We will begin by creating *test case sets* and choosing which tests to be included in each set. A test case set is a group of test cases in a Quality Center project which is designed to achieve specific testing goals. Quality Center provides us to control the execution of test cases in a test set by putting conditions and scheduling the date and time for executing users test cases.

Once we define the test case sets, we can start executing our test cases. When we execute a test case manually, we will execute the test case steps we defined in test planning. The test case can pass or fail at each step, depending on whether the actual results match the expected output. When we execute a

test case automatically, Quality Center can auto open the selected testing tool, runs the test, and exports the test case results to Quality Center. By this we can easily manage the test results.

Adding and Tracking Defects

As the name describes, it is the phase to find and add defects to the Quality Center. Finding and repairing flaws is an important stage in the application/system development. Defects can be detected and submitted by anyone like developers, testers, and end users in all stages of the testing process. We can use Quality Center, to submit defects detected in the application/system and track them until they are repaired.

Tracking Changes

In Quality Center we can also set alerts to send auto notifications. To do this we need create alerts in Quality Center which can send emails automatically to notify them, who is responsible, when changes occur in our project that may effects the testing process. And we can also add our own follow up alerts.

We can also create auto tracking alerts. To create automatic *tracking alerts*, our Quality Center administrator can activate traceability notification rules, which are based on associations we made in Quality Center between

requirements, tests, and flaws. If an entity in our project changes, we are notified of any associated entities that may be affected by the change.

Quality Center also can enable us to add our own *follow up flag* to a specific test case, test instance, or defect to remind ourselves to follow up with an issue. So when the follow up date and time arrives, Quality Center sends the user an e-mail automatically.

Analyzing the Testing Process

In Quality center we can also analyze the total testing process by reports and graphs. Quality Center reports and graphs help the user assess the status of requirements, the test plan, test runs, and defect tracking.

We have an option to generate reports and graphs from any Quality Center at any time during the testing process execution. To generate reports and graphs at any time during the testing process from each Quality Center module, by using default or customized settings. If we want to customize a report or graph, we need to apply filters and sort conditions on the report or graph, to display the information according to our specifications. We can also save our settings as favorite views and reload them whenever needed.

Generating Reports

We can create a report from any Quality Center module. Every Quality Center module consists of various reporting options. Once we generate a report, we can customize the report by changing the report properties to show report data according to our specifications.

Generally, we can generate a standard requirements report, customize the report for a specific user account, and add it to specific user's favorites list.

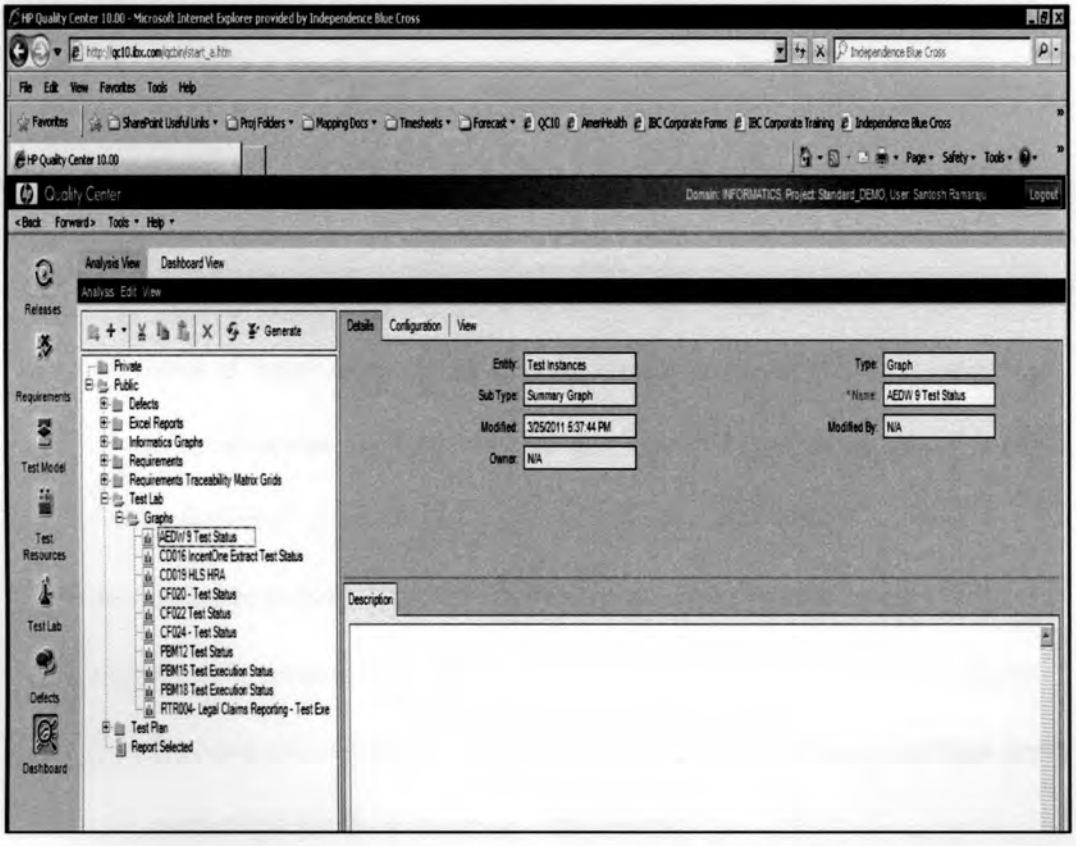

Figure 9. Creating Reports and Graphs

These reports and graphs are very critical in tracking the overall health of the software life cycle.

Chapter 8

CONCLUSION

STREAMLINED TESTING PROCESS FOR ORGANIZATIONS

Most of the long established companies must have had an experience with software that did not work as expected. Software that does not work as expected can have a huge impact on an organization. It can lead to many issues including:

- Loss of money – this can include losing customers right through to financial penalties for non-compliance to legal requirements.
- Loss of time – this can be caused by transactions taking a long time to process but can include staff not being able to work due to a fault or failure.
- Damage to business reputation – if an organization is unable to provide service to their customers due to software problems then the customers will lose confidence or faith in this organization (and probably take their business elsewhere).

- Injury or death – It might sound dramatic but some safety-critical systems could result in injuries or deaths if they don't work properly (e.g., flight traffic control software).

SUGGESTED TESTING FRAMEWORK FOR ORGANIZATIONS

This project details the analysis and advantages of using a defined testing framework incorporating specific suggested tools that fits into the quoted framework and its implementation for medium to large sized organizations.

Although this model may not fit every organization's software development and quality assurance structure it suits most company's needs. Slight alterations to this model can also be done to monogram it to suit each organization.

In short, any process that follows the standard software development life cycle can adapt to Agile Scrum and using tools like Quality Center, Version One, and Quality test professional for automation gain get optimized quality software and a lean process. Above suggested tools have been compared to their counterparts and cost and other factors have also been taken into account while making the comparison.

This research can be continued to further analyze automating software testing using tools like Quality test professional and Selenium IDE or further to test the load of the software by using tools like Smartbear Load Complete or Load Runner.

Chapter 9

RECOMMENDATIONS

Once a testing methodology is chosen, tools to be used are selected and a streamlined process has been thought of for an organization, it can be said that the organization now has a standard testing framework. As resources start adopting this framework it would be wise to start thinking about test automation when suitable is known to cut down resource costs and improve test quality for various organizations.

TEST AUTOMATION

Many paid and free source automation tools are available in today's market with their own set of advantages and limitations.

Each organization has a testing process that needs to be understood before introducing test automation. Some companies have processes based on strict standards because they work with systems that deal with critical information (such as banks or other financial institutions). Other companies could work on projects where time for testing is very limited, where it is not convenient to spend time in automation scripts (in this case, manual testing of

critical areas would be the best approach). Each company is different, so even though Test Automation can bring benefits, we need to analyze how test Automation will affect our current test process including the benefits and costs of introducing the tool.

Any test tool is intended to make the testing process more effective and efficient. A test tool can provide many benefits, but there are risks that need to be considered before introducing a tool into an organization. "Most of the risks associated with the use of test tools are concerned with over-optimistic expectations of what the tool can do and lack of appreciation of the effort required implementing and obtaining the benefits that the tool can bring."

STRESS AND LOAD TESTING

Stress and Load testing have been identified as an important part of testing lately. An early detecting of poor functionality due to user load saves lot of resources and capital.

SUGGESTED AUTOMATION TOOLS

QTP: HP QuickTest Professional is automated testing software designed for testing various software applications and environments. It performs functional and regression testing through a user interface such as a

native graphical user interface or web interface. It works by identifying the objects in the application user interface or a web page and performing desired operations such as mouse clicks or keyboard events. It can also capture object properties like name or handler ID. HP QuickTest Professional uses a VBScript scripting language to specify the test procedure and to manipulate the objects and controls of the application under test. To perform more sophisticated actions, users may need to manipulate the underlying VB script.

HP QuickTest Professional and HP Quality Center work together for additional testing capabilities. Users can use HP Quality Center with HP QuickTest assets, such as tests, shared object repositories, libraries, recovery scenarios and external data tables. A sample QTP screenshot shown below:

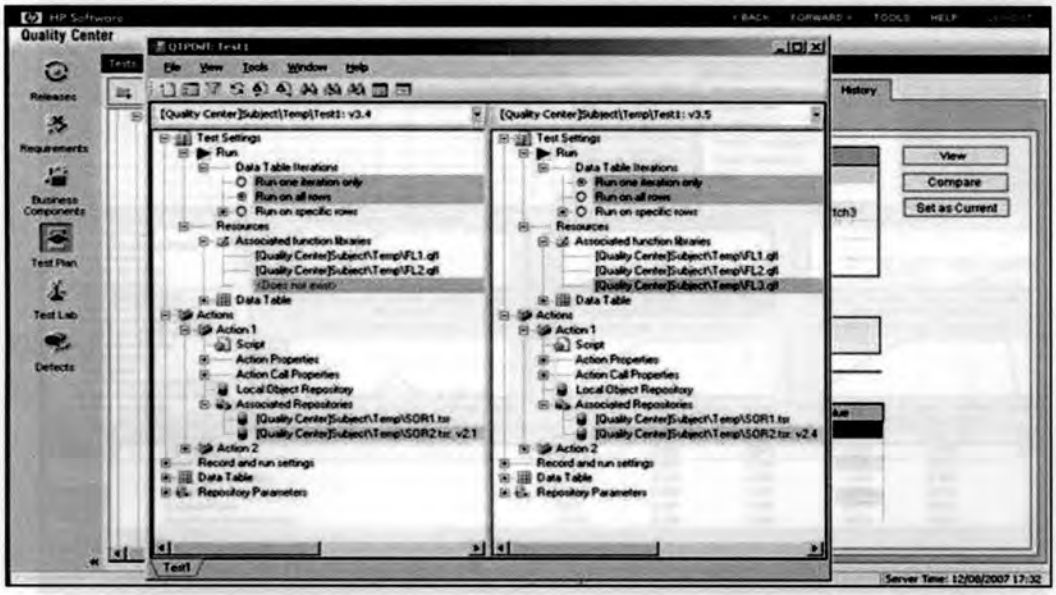

Figure 10. QTC Screenshot

Load Runner: HP LoadRunner is an automated performance and test automation product from Hewlett-Packard for examining system behavior and performance, while generating actual load.

HP LoadRunner can simulate thousands of concurrent users to put the application through the rigors of real-life user loads, while collecting information from key infrastructure components (Web servers, database servers, etc.) The results can then be analyzed in detail, to explore the reasons for particular behavior.

A screenshot of LoadRunner is shown below and depicts the behavior of the application at the current load conditions. Performance testing is generally done by having dummy users login to the application.

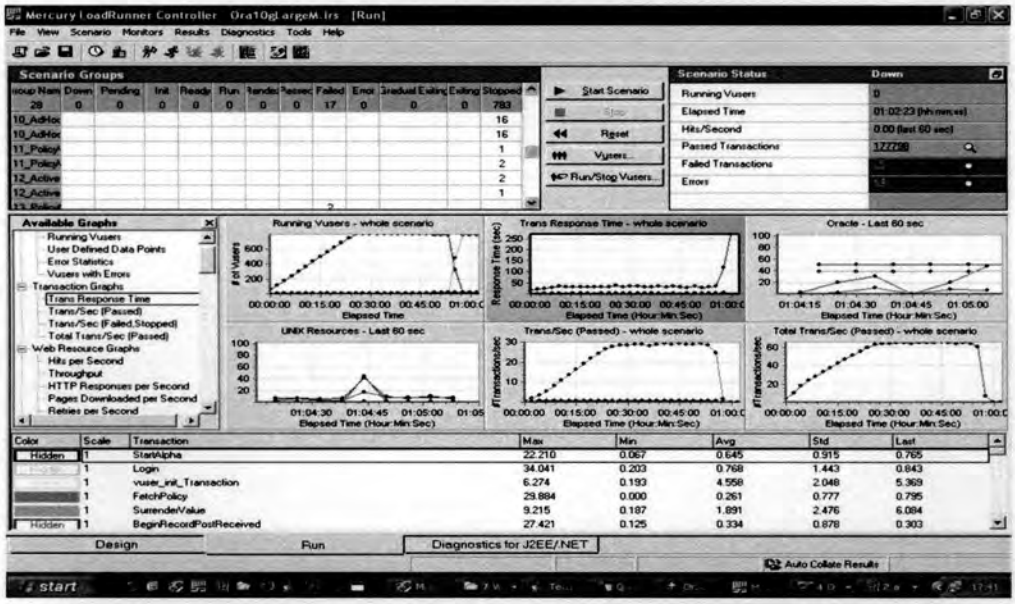

Figure 11. LoadRunner Screenshot

Stress and Load tests are a part of performance testing of an application and generally done when all of regression testing is complete and just before the application is ready to go-live.

It shows the behavior of an application under extreme load/stress and such a test is necessary to prevent crashing of the application in the production environment when user traffic is high.

REFERENCES

REFERENCES

QA and SDLC references:

- Software Quality – Concepts and Plans by Robert H. Dunn
- Software Quality Management - by John D. Cooper, Matthew J. Fisher
- www.ctg.com
- www.construx.com
- www.testingcraft.com/techniques.html
- http://www.qaielearning.com/
- www.exforsys.com/tech-articles/testing
- www.snt.ro/Content.Node/news
- www.ece.cmu.edu/~koopman/des_s99/sw_testing
- www.softwareqatest.com
- http://www.SoftwarePlanner.com
- http://www.automatedqa.com

QA Tools References:

- http://www.versionone.com/screenshots/index.asp
- http://www.automation-consultants.com/products-LoadRunner-134
- http://www.quicklearnqtp.com/2009/07/qtp-scripts-examples-for-beginner.html
- http://agile.dzone.com/articles/10-scrum-methodology-best

- http://www8.hp.com/us/en/software-solutions/software.html?compURI=1172141#.UjeodMbkuLQ
- http://www.guru99.com/quick-test-professional-qtp-tutorial.html

APPENDIX

Testing Terms

API:	Application Program Interface. A set of function calls and protocols, which a program uses to communicate with the outside world. APIs are usually provided by the operating system.
Backup and Recovery Techniques:	The methods used to protect software and data assets from loss or destruction. The establishment of alternate sites for storing software and data to be utilized in recovery efforts, and the planning for recovery and processing of information (e.g. recovery from loss of processed data due to an abend).
Baseline:	Determining the current performance, accuracy, or efficiency of a product(s) or process (es) for the purpose of measuring current performance or measuring future or past variances from current performance.
Black Box Testing:	Assumes that the internal details of the module being tested are not known or available to the tester. Testing is restricted to verifying that the module functions as demanded by the specifications document.
Data-Gathering Techniques:	Identifying or developing and using problem reports, incident reports, and the like, to gather the data that can be used for the improvement of the enterprise's information process.
Database trigger:	Code that is stored in the database and implicitly executed (fired) when a table is modified.
Debugging:	Process, which begins with an identified error and continues by trying to find out how it is possible for the error to occur.
Defect:	(See Test Discrepancy) 1. A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. 2. Human action that results in software containing a fault (e.g. omission or misinterpretation of user requirements in a software specification, incorrect translation or omission of a requirement in the design specification).
Defect Masking:	If complex areas of the system are not tested until late in the process, many defects may be hidden until higher-level defects are uncovered.
Performance Testing:	Testing to prove that the system meets its defined throughput (clock time or CPU time) and response time objectives. (Note: The same test cases used for volume or stress testing can often be used for performance testing.)

Pilot Testing:	Exercising a part of a system in a production environment. Usually a pilot test consists of one location or one product line.
Project Control:	Methods available to project management to assist them in accomplishing project objectives with available resources and within defined time spans. Project control can be a variety of manual and software methods.
Testing:	Exercising system components to find previously undiscovered errors. Therefore, a good test case is one that has a high probability of finding new errors. A successful test discovers one or more new errors. Occasionally, it may reveal underlying errors as well. Testing is a form of risk assessment that answers the question, How confidently can I trust my business to this software?
Unit Testing:	Confirm that an individual program meets its technical design requirements as defined in the program design documents for that program. Normally done by the developer
Usability Testing:	Validates that the ease of use of the system supports the level of expertise of the user, and is consistent and intuitive. This would involve testing the user interface, sequence of events, PF Keys, GUI design, etc. Some of this validation may be supported by formal design standards, but other parts will be more abstract. Can be validated by including early informal testing in the acceptance-testing phase and/or by prototyping.
User Acceptance Testing:	Validation that the delivered application or system meets the actual business needs of the end user.
Validation:	The process of evaluating code and associated documentation at the end of the change process to ensure compliance with software requirements.
Verification:	A visual inspection of an immediate deliverable for the purpose of finding defects.