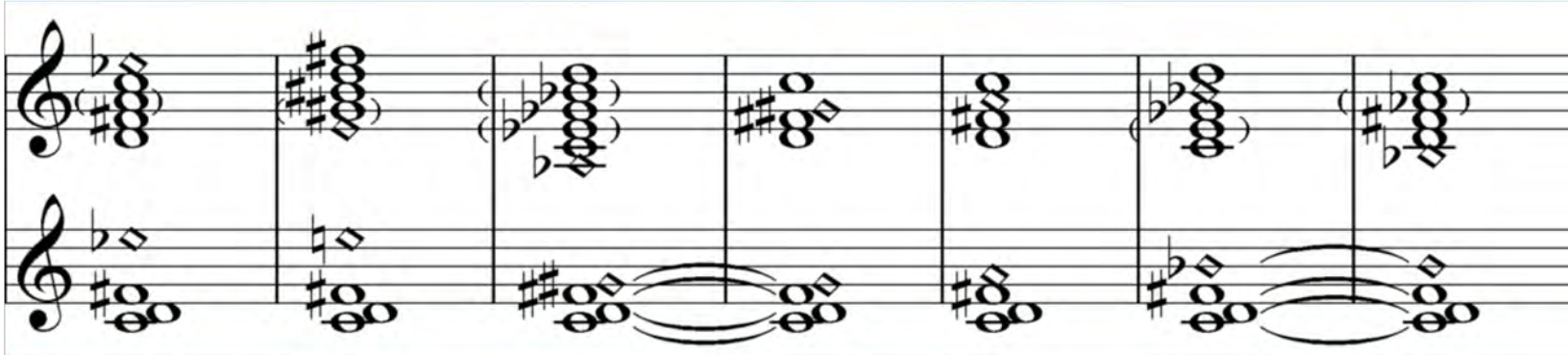




# Rational Team Concert 6



## Test Management with Rational Quality Manager 6.0

## Student Manual

Version 1.1

## © Copyright 2019 by Rod Davison. All rights reserved.

No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage or retrieval, for commercial purposes without prior written permission of Rod Davison. Creating and sharing copies for personal use is freely permitted as long as the copies are provided free of charge.

This material references, quotes and cites published standards, best practices and material created by Rational Software Corporation, IBM Ltd, the Jazz community and others who have contributed to the development of the various products mentioned in this course, as well as related methodologies and concepts. Any copyright claims by Rod Davison are intended to apply only to the material uniquely created for this work and is not intended to apply to any of the work of other authors that are cited, quoted or referenced in this material.

Original material in this document may be quoted provided that the quotations are both in accordance with the principles of fair use, properly cited and attributed to this source. Quotes of material which is the work of others should be properly attributed to the original sources and authors.

Any products that may be described in this document may be protected by one or more U.S. patents, foreign patents or pending patents or copyrights. Trademarked names may appear in this material. Rather than use a trademark symbol for every occurrence of a trademarked name, the name is used only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Every effort has been made to ensure all quotations from published copyrighted sources have been cited in accordance with accepted academic practices of fair use. Any unattributed quotes are an oversight and should be reported to the author so that the appropriate attribution can be made.

The information in this publication is distributed on an “as is” basis. While every precaution has been taken in the preparation of this material, the author assumed no responsibilities for errors or omissions, or for damages resulting from the use of the information contained herein.

## Terms of Usage

All copies of this publication are licensed to only be used as course materials in “Test Management with Rational Quality Manager 4.0” developed by Rod Davison. Copies are not licensed for resale in any form or for use in any other course or by any other individual or organization except where authorized by the author.

## Document Information

Course Title: Test Management with Rational Quality Manager 6.0  
Author: Rod Davison  
e-mail: rod@exgnosis.ca  
Version: 1.1  
Release Date: 2019-04-02  
ProTech ID: PT20386

*"The primary objective of copyright is not 'to reward the labour of authors, but to promote the Progress of Science and useful Arts.' To this end, copyright assures authors the right to their original expression, but encourages others to build freely upon the ideas and information conveyed by a work. This result is neither unfair nor unfortunate. It is the means by which copyright advances the progress of science and art."*

Justice Sandra Day O'Connor

# Student Preface

This is the manual that accompanies the course Test Management with Rational Quality Manager 6.0, and is part of the larger documentation set that you should have received for the course. The role of this manual is to provide several areas of support for you during and after the course.

First is to provide you with a record of the class content beyond what is presented in the slides.

There is only so much can be crammed into a slide or PowerPoint presentation – this manual covers the same topics as in the slides but provides additional depth and background. This means that you can go back and revisit the material later without having to remember (and you usually won't) what the instructor actually said about a particular slide, or explore areas of interest in greater detail than was possible to do in class.

Second is to ensure that you get all the planned content. Sometimes in a class not all of the topics are covered equally. This may be due to the skill level of the class, or because a class wants to go into more depth on certain topics and less on others, or any of a number of other legitimate reasons for the content as delivered in class to not quite align with the content that was originally planned. If this happens in your class, you will have all the planned content in the manual to take away with you after the course.

The third is to provide more content beyond what is presented in class. One of the challenges of working with any IBM product is that it is often quite difficult to get to the right documentation for a specific issue or to understand the documentation without some sort of context. (Also, please see the note at the start of this section about support no longer being available from IBM). A general goal for this manual is to provide a presentation of some of the standard IBM reference material in a format that is easier to follow and learn from, as well as providing enough background material so that the Quality Manager topics can be presented in context, hopefully making them more understandable.

## ***What this manual is not***

This manual is not the definitive guide to the Jazz Server, the Rational Team Concert platform, or a detailed technical exploration of Quality Manager: those sorts of deep dives into such topics is beyond the scope of the what we can cover in the two days allotted to this course. For very specific or detailed issues or questions about the any of these topics, the best approach is to consult the IBM documentation and the documentation available through the Jazz.net website. However, because this documentation may no longer be available, you probably should consult the version 5.0 documentation.

This manual is not a definitive guide or intensive course on test management or software testing – the focus is rather on how to map the concepts and practices of software test management onto the features and facilities of the Quality Manager product.

This manual is not a guide to the specific issues that you might encounter in your systems environment. Organizational protocols, custom testing processes and techniques, integration with other tools, your organization's testing policies and practices all affect how you work with Quality Manager. This manual can only reference a sort of generic environment that you can, hopefully, use as a starting point for exploring your use of the product.

## ***Prerequisites and Expectations***

This course is about using Rational Quality Manager for automating the administration of a testing project, not about the mechanics of doing software testing. It is assumed that you have some prior experience with the principles and practices of software testing in general – these will not be reviewed in class although we may make reference to them.

The general approach to test management is to apply a number of project management concepts and practices to a testing project. We will be assuming a basic knowledge of project management concepts and practices, although this will be required only at a very basic level.

## **Test Management Standards**

Throughout the course there are references to the IEEE 829 standard and a number of the modules summarize parts of the standard. Because this standard is generally unavailable to the average student, parts of it have been paraphrased for inclusion here as part of the manual support. This is not intended to be a replacement for you getting your own copy, and it is hoped that it might encourage you to convince your manager to make the investment in getting a copy.

The referenced standards represent the best practices on which the RQM design is based. Referencing the standard in this course hopefully answers in some of the conceptual questions that might arise such as “I wonder what the reason is for this feature in RQM?”

## **The RQM Product**

One of the problems that seems to plague all open source products or community sourced products is the lack of a systematic and comprehensive user guide. I find with RTC, this is particularly true since the online user manuals are very poorly organized, difficult to find, often contain more marketing hype than content for some topics and generally do not get into the depth that one would want. I noticed in preparing this document that the explanation for the various sections in the test plan was just a link to a non-existent YouTube video.



Part of what this manual is intended to do is also to pull together some of the IBM documentation that is freely available but hard to find, and to put it here in one accessible location. Now that the product has been discontinued, this manual will also serve as a historical archive of whatever documentation that I could find.

## ***Course Objectives***

Keep in mind that this is an introductory course. That means that we will touch on a range of RQM related topics without going into real deep dives on many of them. For example, we are going to look at setting up test plans, but we are not going to be able to work through all of the variations of a test plan that you might encounter in your own testing environment.

Some realistic expectations are that after the class is that you should be able to:

1. Explain the basic architecture of the Rational Team Concert platform, it's main features and components and the main features and capabilities of the Quality Manager.
2. Develop a test plan
3. Define schedules, test environments and lab resources
4. Develop a test case and test script
5. Create a test suite
6. Run tests
7. Submit a defect
8. Monitor the test effort
9. Report test results and testing activity status

In addition, you should be able to see clearly how the various industry standard best practices in software test management are automated in the Quality Manager product.

## ***Hands On Work***

Hands on work in a course like this is critical for learning. There are a series of hands on exercises for you to work on in class with some suggestions for some more advanced explorations you can try on your own.

Keep in mind we are not working in a production environment so there are some limitations in the virtual machines you are using on the laptops that may limit performance. One of the more notable limitations is that we do not have an automated build environment that integrates with RQM.

We are not looking for THE RIGHT ANSWER<sup>(tm)</sup> in many of the labs. The labs are for experimentation, practising and getting that hands-on feel for the product that is often the most critical part in learning. The lab environment is a “consequences free” set-up in the sense that you can do things in the isolated virtual machine that if you did it in your production environment, might result in a testy call from a supervisor requesting an explanation for what you are doing.

The labs are not intended to be a solitary activity. I would encourage you to collaborate and confer with your classmates . Often a lot of learning can take place just in the process of trying to explain how something works to one of your colleagues. It’s also enlightening to see how other people are approaching a problem and compare that to how you are doing it. Think of the labs as a team or class activity if you want, but if it is more your style to work on your own, just do your own thing.

Above all, try to have fun with the labs. We learn best when you are enjoying yourself.



# Table of Contents

## Module One: CLM, RTC and RQM

1.1 Introduction.....	1-1
1.2 Application Lifecycle Management.....	1-3
1.2.1 Engineering Development Processes.....	1-3
1.2.2 Software as Product.....	1-5
1.3 Collaborative Lifecycle Management.....	1-10
1.3.1 Unified Change Management (UCM).....	1-10
1.3.2 RQM Artifacts and Terminology.....	1-12
1.3.3 Five Imperatives for Application Lifecycle Management.....	1-13
1.4 Rational Team Concert.....	1-16
1.4.1 The Eclipse Architecture.....	1-16
1.4.2 Collaborative Lifecycle Management Architecture Overview .....	1-16
1.5 Overview of Jazz Team Server.....	1-20

## Module Two: Overview of Test Management

2.1 Introduction.....	2-1
2.1.1 Defining Test Management.....	2-1
2.1.2 Out of Scope Topics .....	2-2
2.2 Test Management Processes.....	2-3
2.2.1 IEEE 829 Standard for Test Documentation.....	2-3
2.2.2 Standard as Best Practice.....	2-4
2.3 Software Quality and Testing.....	2-6
2.3.1 Identifying the Need the System Meets.....	2-12
2.3.2 Defining Quality.....	2-6
2.3.3 The Good-Enough-Quality Model.....	2-9
2.3.4 Beizer's Tester's Mental Maturity.....	2-11
2.4 Process Maturity.....	2-12
2.5 Testing Maturity.....	2-16
2.5.1 Defining Testing Process Maturity.....	2-16
2.5.2 Behavioural Characteristics of the TMM Levels.....	2-18
2.5.3 Maturity Goals at the TMM Levels.....	2-20
2.6 Defining Good Testing.....	2-26
2.6.1 Impact of Poor Testing.....	2-26
2.6.2 Risk Mitigation.....	2-27
2.6.3 Good Testing.....	2-28



2.7 Sources of Testing Errors.....	2-34
2.7.1 Human Errors.....	2-34
2.8 “The Way of the Tester” .....	2-41
2.8.1 Formal Protocols.....	2-41
2.8.2 Formal and Repeatable Processes.....	2-42
2.8.3 Myers Testing Principles.....	2-43
2.9 Risk Matrix.....	2-26

## **Module Three: Rational Team Concert and Jazz Basics**

3.1 Introduction to RTC and Jazz.....	3-1
3.1.1 Starting up Jazz and RTC.....	3-1
3.2 Users and Roles.....	3-4
3.2.1 System Users.....	3-5
3.2.2 Jazz Security Roles.....	3-5
3.2.3 CLM Licences.....	3-6
3.3 Creating a User.....	3-8
3.3.1 Logging Into a User Area.....	3-9

## **Module Four: Test Plans**

4.1 Introduction.....	4-1
4.2 IEEE 829 Master Test Plan (MTP).....	4-2
4.2.1 Master Test Plan Outline.....	4-2
4.2.2 Detailed Descriptions.....	4-4
4.3 IEEE 829 Level Test Plan (LTP).....	4-11
4.3.1 Level Test Plan Outline.....	4-12
4.3.2 Detailed Descriptions.....	4-13
4.4 Software and System Integrity Levels.....	4-20
4.5 Test Plans in Rational Quality Manager.....	4-21
4.5.1 Test Plan Overview.....	4-21
4.5.2 Test Plan Development Checklist.....	4-23
4.5.3 Creating Test Plans.....	4-24
4.5.4 Managing Test Artifact Sections.....	4-26
4.5.5 Creating Master and Child Test Plans.....	4-27
4.5.6 Linking to Requirement Collections.....	4-28
4.5.7 Associating Test Plans with Requirement Collections.....	4-29
4.5.8 Reconciling Test Plans with Requirements.....	4-30
4.5.9 Linking to Development Plans.....	4-34
4.5.10 Creating Test Schedules.....	4-35
4.5.11 Estimating the Overall Size of the Test Effort.....	4-38





4.6 Planning Test Environments.....	4-39
4.6.1 Platform Coverage and Test Environments.....	4-39
4.6.2 Defining Platform Coverage.....	4-40
4.6.3 Generating New Test Environments.....	4-41
4.6.4 Adding Existing Test Environments to a Test Plan.....	4-42
4.6.5 Test Environment Types.....	4-43
4.6.6 Environment Values: Types of Machines.....	4-45
4.6.7 Test Environment Values: Any Lab Resource Type.....	4-47
4.6.8 Adding Quality Objectives.....	4-50
4.6.9 Using Shared Resources.....	4-50
4.6.10 Copying Portions of Existing Documents.....	4-51
4.6.11 Reviewing Test Plans.....	4-52

## **Module Five: Working with Artifacts**

5.1 Introduction .....	5-1
5.2 Developing Test Cases.....	5-2
5.2.1 Creating Test Cases.....	5-6
5.2.2 Managing Test Artifact Sections.....	5-8
5.2.3 Importing Test Cases.....	5-9
5.2.4 Adding New Test Cases.....	5-10
5.2.5 Adding New Test Cases.....	5-11
5.2.6 Adding Existing Test Cases.....	5-12
5.2.7 Adding Existing Test Scripts to a Test Case.....	5-13
5.2.8 Adding a New Test Script to a Test Case.....	5-13
5.2.9 Estimating the Weight of Individual Test Cases.....	5-14
5.3 Linking and Managing Requirements.....	5-15
5.3.1 Associating Test Cases with Requirements.....	5-15
5.3.2 Generating Test Cases.....	5-15
5.3.3 Reconciling Test Cases and Requirements.....	5-17
5.3.4 Requirement Reconcile Operation Status Types.....	5-19
5.3.5 Configuring Data Record Selection Criteria.....	5-10
5.4 Developing Test Suites.....	5-21
5.4.1 Adding Existing Test Suites to a Test Plan.....	5-24
5.4.2 Adding a New Test Suite to a Test Plan.....	5-25
5.5 Test Execution Records.....	5-26
5.5.1 Single Test Case Execution Records.....	5-26
5.5.2 Test Suite Execution Records.....	5-27
5.5.3 Automatic Generation of Test Execution Records .....	5-29
5.5.4 Generation of Test Execution Records from Test Plans.....	5-31
5.5.5 Editing Test Execution Records.....	5-32
5.6 Test Scripts.....	5-33
5.6.1 Execution Variables.....	5-33

5.6.2 Manual Test Scripts and Statements.....	5-37
5.6.3 Using the Manual Test Script Recorder.....	5-38
5.6.4 Adding Existing Scripts to a Test Case.....	5-44
5.6.5 Creating Manual Test Scripts.....	5-44
5.6.6 Manual Test Scripts from Test Case Design.....	5-47
5.7 Manual Test Scripts.....	5-49
5.7.1 Attaching a File to a Step.....	5-49
5.7.2 Adding Comments.....	5-49
5.7.3 Verifying Text.....	5-50
5.7.4 Providing Assisted Data Entry.....	5-50
5.7.5 Importing Manual Test Scripts.....	5-51
5.7.6 Linking Test Script Steps to Requirements.....	5-52
5.7.7 Working with Test Data.....	5-55
5.8 Managing Lab Resources.....	5-59
5.8.1 Customizing Lab Resource Properties.....	5-60
5.8.2 Creating New Test Environments.....	5-62
5.8.3 Viewing Test Environments.....	5-64
5.8.4 Creating Data for Physical Machines.....	5-65
5.8.5 Creating Virtual Images.....	5-66
5.8.6 Creating and Viewing Test Cells.....	5-67
5.8.7 Managed Virtual Images and Machines.....	5-68



# Test Management with Rational Quality Manager 6.0

## Module One

---

### *CLM, RTC and RQM*

*If you automate a mess, you get an automated mess.*  
Rod Michael

*It is not a question of how well each process works,  
the question is how well they all work together.*  
Lloyd Dobens and Clare Crawford-Mason

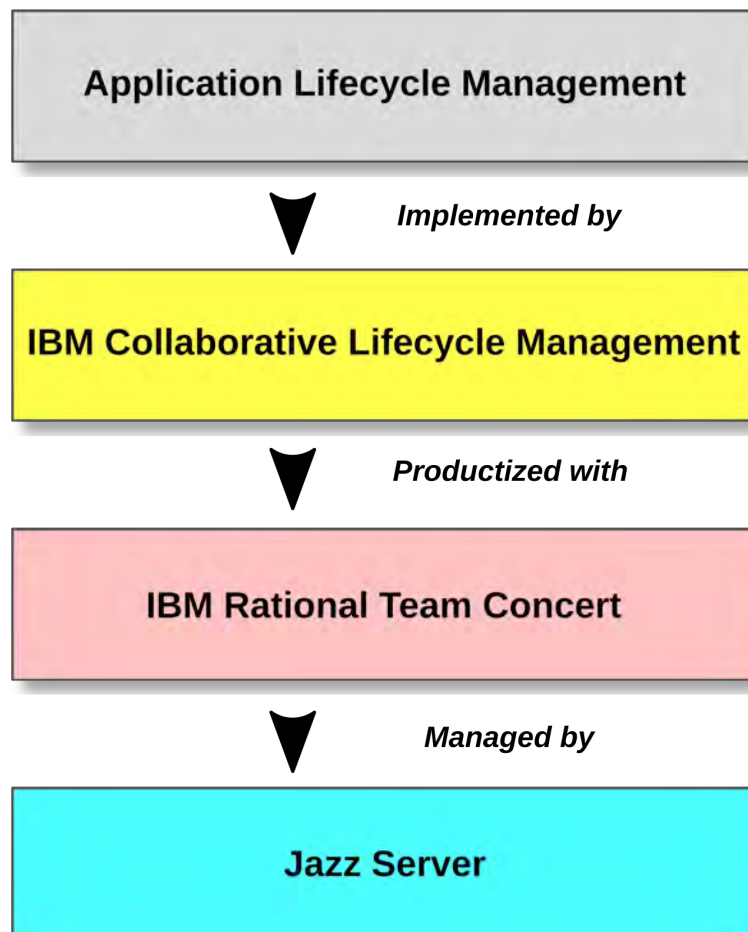
*If we deliver on time, but the product has defects,  
we have not delivered on time.*  
Phil Crosby



## 1.1 Introduction

This first module provides some conceptual background about what the Rational Quality Manager (RQM) product, the design ideas behind its architecture and functionality and how it relates to the other IBM Rational products like Rational Team Concert (RTC), the Jazz platform and IBM's Collaborative Life Cycle Management (CLM). Understanding what something is supposed to do and the design choices that were made for the product enhances our ability to both manage RQM but also gives us insights that allow us to diagnose and correct possible problems.

The first section discusses the general concepts of Application Lifecycle Management (ALM). ALM is not a product not is it something develop by IBM. ALM is a set of inter-related ideas, concepts and practices which form the basis of a methodology for managing a software as a product through its product lifecycle. Many software vendors, including IBM, have implemented products and methodologies which are customized implementations of ALM to support their products with ALM concepts in both production and support environment.



***Collaborative Lifecycle Management (CLM)*** is a formal process and multiple sets of associated artifacts that IBM has developed to implement their version of ALM. CLM defines a number of specific artifacts, including artifacts created and managed in RQM, that are stored and maintained in the Jazz Server environment. Having a basic understanding of these concepts will make it easier for you to understand the functionality of RQM and how it integrates with the rest of the RTC platform.

The Jazz platform is built on what might be called second generation Eclipse platform, and was designed to replace the aging legacy CLM applications that IBM inherited from Rational Software – including but not limited to ClearCase, ClearQuest, Test Manager, DOORs and RequisitePro. Rational Team Concert and its products are built in a modular fashion on top of the Jazz platform with various RTC modules replacing the legacy Rational Suite software. Specifically, RQM is intended to replace the older stand alone IBM Rational Test Manager.

This first module is not going to be an in depth or detailed discussion of these topics but should provide you with the essential basics you need to know about the design context of RQM so that you can navigate through the product more effectively and with more confidence, especially if you have not had any previous exposure to the RTC platform.



## 1.2 Application Lifecycle Management

Application Lifecycle Management (ALM) is a set of concepts that defines a set of best practices in software development. ALM is not a methodology or a process like Agile or DevOps, but focuses instead on broader management concerns that are common to all development process methodologies, whether they are Agile, waterfall or iterative.

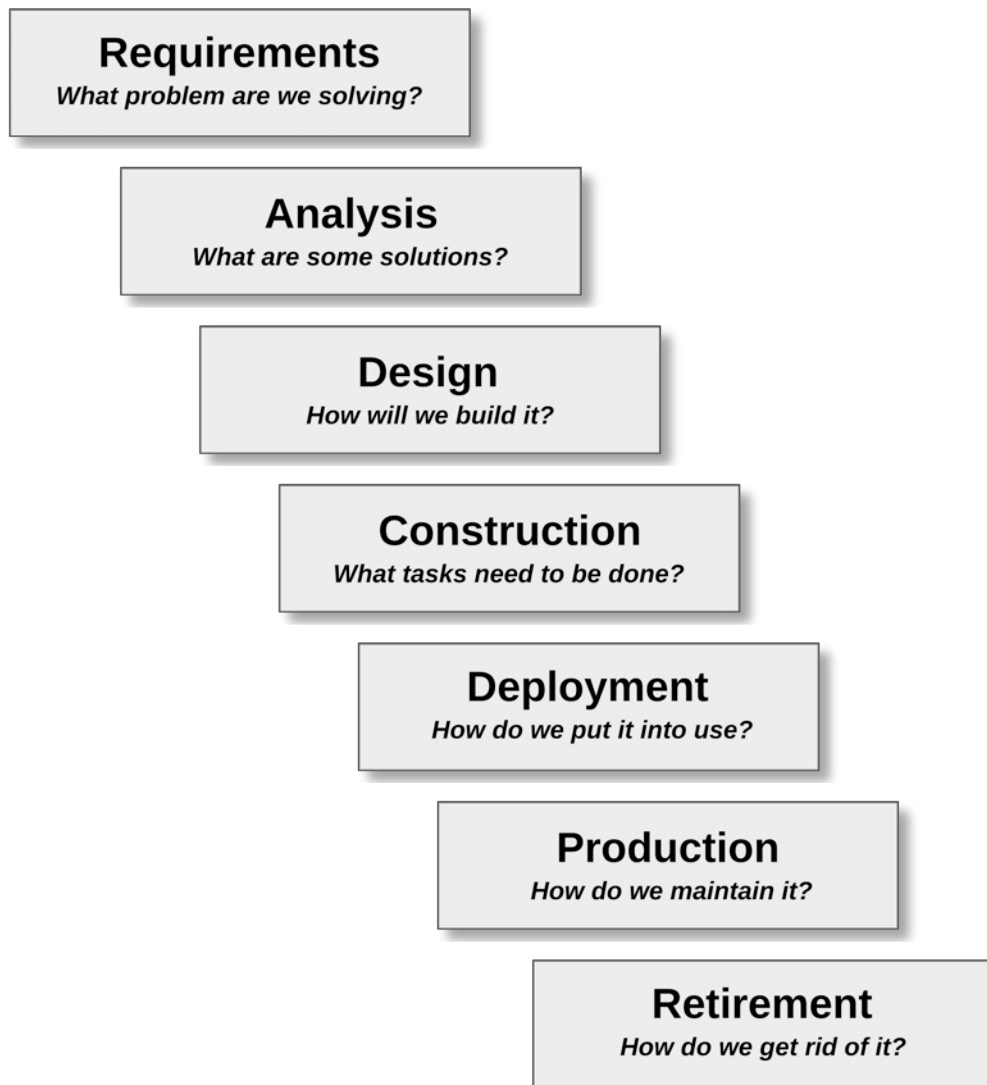
The primary difference between a software development process and ALM is that most development processes deal only with the activities up to the completion of the initial delivery of the software. ALM takes the broader perspective of “software as a product” which means the focus is not just the development phase but also includes the whole span of time the software is in production, being maintained and eventually removed from production.

### 1.2.1 Engineering Development Processes

Software development processes are special instances of a more general process model which will be referred to as the “generic engineering process in this discussion. The generic engineering process defines a set of logical steps that have to be followed every time we build something. These steps are:

1. **Requirements:** Before we can build any sort of solution, we need to understand what the problem is we are trying to solve or what the user need is we are trying to meet. We need to clearly understand what the solution should do from the stakeholders' perspective in order to meet their needs.
2. **Analysis:** Once we understand the problem, we generate solutions. The solutions are evaluated and eventually one is selected to be implemented. Choosing a solution depends on a number of factors including cost, difficulty and availability of resources. Solutions that are conceptually ideal may have to be rejected on pragmatic grounds (“We just don’t have the budget for this...”). Very often, negotiations take place between the developers and the different stakeholders to arrive at a compromise solution that is both (1) acceptable to the stakeholders even it does not completely meet their needs; and (2) can be delivered on time, within budget and will meet the agreed upon specification. In this engineering context, a “specification” is a description of what the delivered product will do and how it will behave.
3. **Design:** The design process takes a specification and creates a plan for building the solution with the resources available. In a nutshell, the specification describes “what” is to be build and the design describes “how” it will be build. Very often problems identified in the design phase may require negotiation with stakeholders to modify the specification so that the design can meet the constraints of time, budget and available resources. Generally speaking, a design is several orders of magnitude more complex than a specification.





4. **Construction:** The phase that usually consumes most of a project's time and resources is construction which, as you would expect, is turning the design into a finished product whether it is writing code for a software application or building a house. Most cost overruns and delivery delays with project occur in this phase. Once a construction mistake has been made, "rework" is required to remove what has been build incorrectly, then redo that part of the construction. One of the main goal of the requirements, analysis and design phases is to eliminate as many errors that could lead to these sorts of mistakes so that rework, during the construction phase is kept to a minimum. This is same thinking as the old carpenter's adage "Measure twice and cut once."



5. **Deployment:** Once the product is built, it has to be put into its operational environment. The process of deployment often involves beta and acceptance testing as well as a transition plan make to make the product operational in the business environment. With software, configuration issues are addressed during deployment as well as making any modifications to business processes that might be necessary, including training users on the new product. Very often the problems that are encountered in deployment may require some redesign to work around the issues discovered during the deployment phase.
6. **Production:** This is usually the longest phase of a product's lifecycle: the product is in use in the operational environment and needs to be maintained and supported. Maintenance usually involves dealing with change requests, modifications needed because of changing requirements, bug fixes, architectural changes in the deployment environment and a whole range of other tasks necessary to keep the product current and responsive to the needs of the users and stakeholders.
7. **Retirement:** The basic question this phase tries to answer is "How we migrate from this product to its replacement?" Many organizations today are stuck with mission critical applications that are out dated, increasingly expensive and difficult to maintain; however, because there was no initial retirement migration plan, the organization has no idea how to untangle their business processes and data from these legacy applications.

The problem with most standard software development processes is that they are not concerned with the software after it's deployed, preferring instead to hand over any maintenance of the software to a support team, or in the worst case, just not make any provision for maintenance.

The different development processes – whether Agile, Iterative, DevOps or Waterfall – all implement the same underlying engineering development process, but vary in how those activities are organized; but the all share the same tunnel vision of looking only at the development phase of a software product.

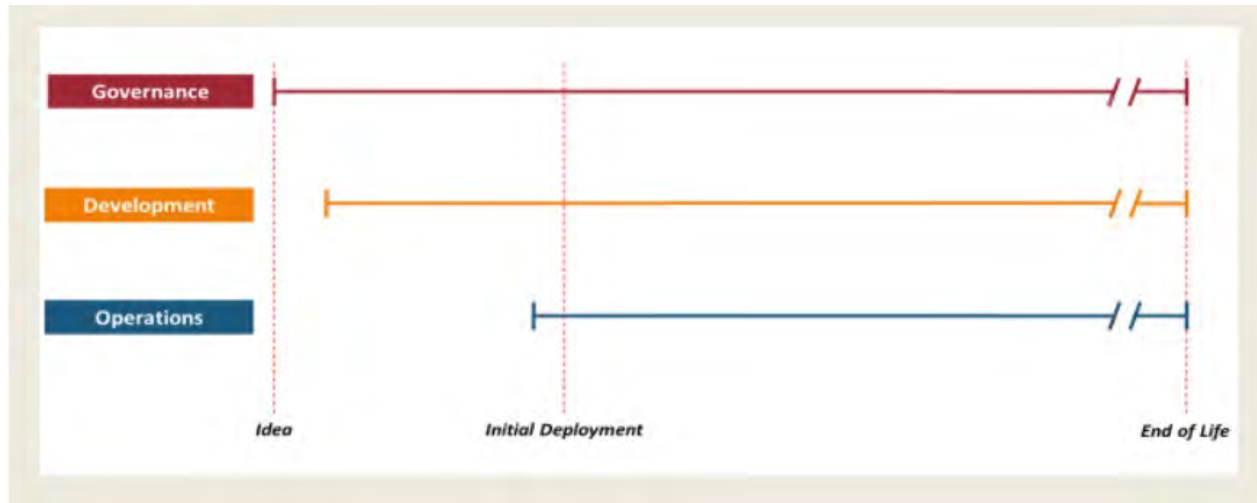
## 1.2.2 Software as Product

*The material in this section is derived from James Chappell's excellent summary of ALM from his 2008 paper sponsored by Microsoft.*

A common description of an application's lifecycle is "the entire time an organization spends money of the product from the initial idea to the end of the application's life when it is no longer in use." The diagram following shows that there are three events that are considered milestones in an application lifecycle:

1. **Idea or Inception:** This occurs in advance of the start of the development process and usually involves looking at whether there is a case to be made for developing the application, examination of the business case, deciding whether to build or buy the application, and other sorts of portfolio management issues and feasibility considerations.

2. **Deployment:** The point in time where the completed product is moved out of development and into production and begins to be maintained in the operational environment. This event marks usually marks some sort of hand-off between the development team and a support team.
3. **Retirement:** At some point the application has to be retired, which may not be a simple process – for example, it may involve transforming or archiving massive amounts of data or ensuring that automation support for critical business processes is not interrupted.



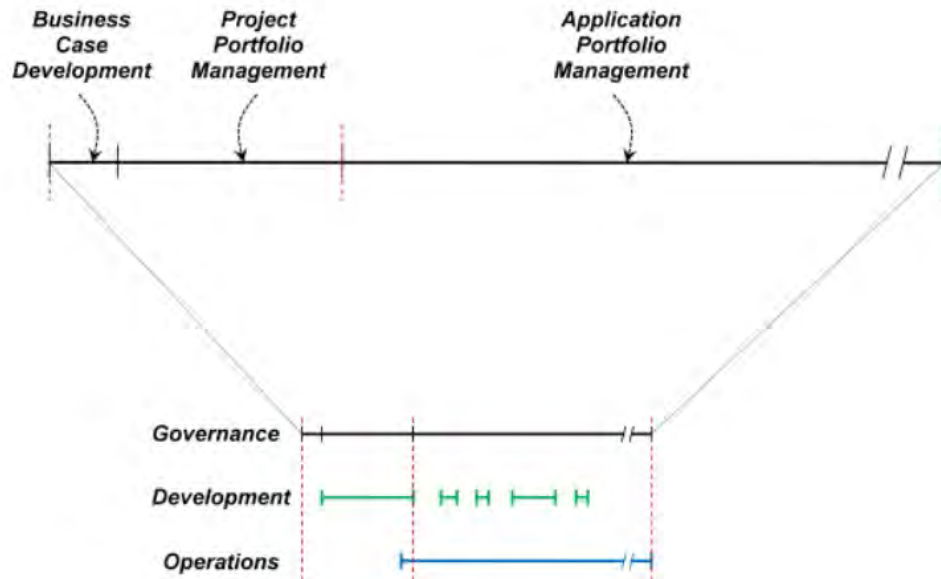
Spanning these milestone events are three main areas of concern. These concerns focus on aspects of a product lifecycle that do not overlap with the issues and concerns of whatever software development process is being used, but supplement and complement them.

## Governance

Governance starts with business case development and feasibility studies. Once development begins, governance is implemented through project portfolio management where the role of the application in the overall business context and IT architecture is continuously evaluated with respect to some set of organizational wide metrics, standards and valuations. After an software product is deployed, it becomes part of the organization's portfolio of applications and becomes an organizational asset which is subjected to an ongoing assessment of its associated benefits and costs.

Some of the responsibilities of portfolio management are deciding when updates, revisions, and even retirement make business sense, as well as ensuring that the application does not either duplicate the function of other applications, or leaves gaps in business process support functionality that could create risk and business level liabilities.

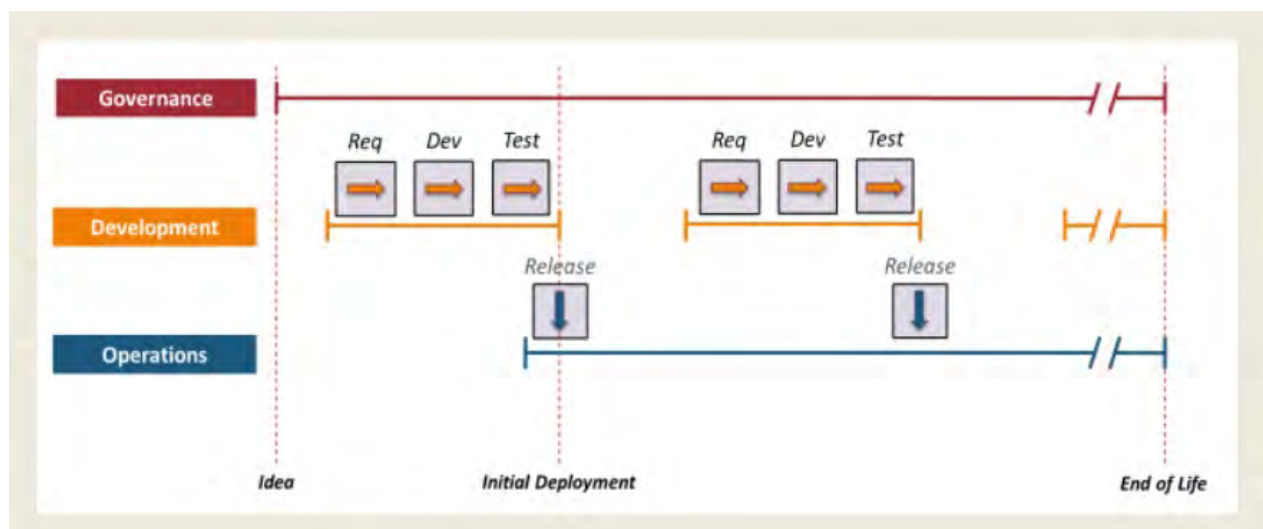


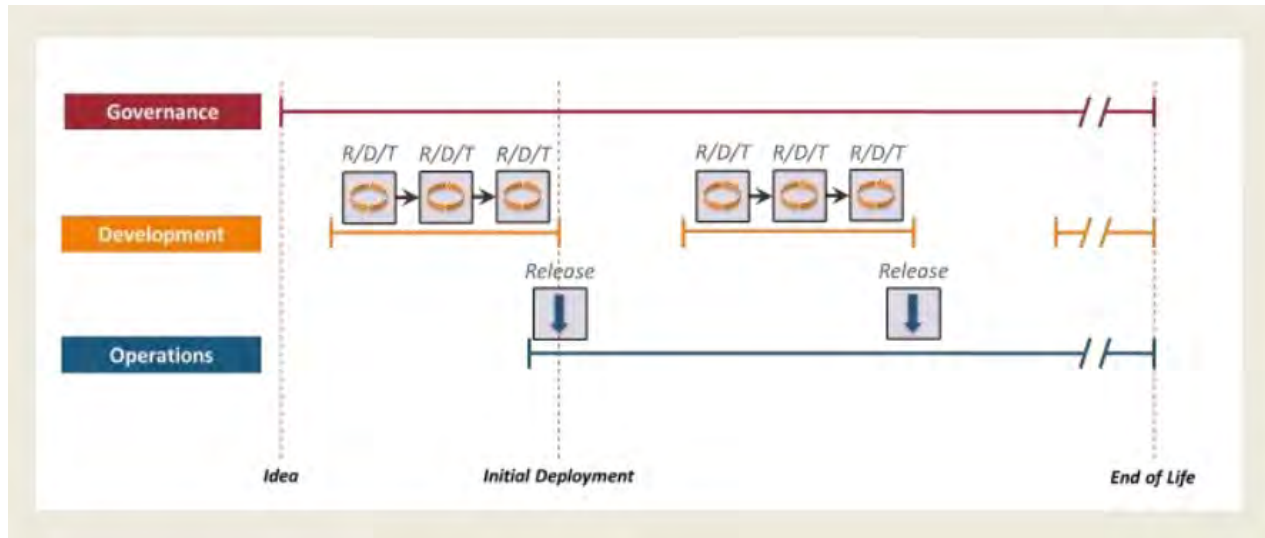


One aspect of governance that is increasingly critical for many organizations is the assessment of security risks and liability issues arising from hacked data, compliance with new regulatory frameworks and other similar issues. These issues might result in a decision to mitigate risk by retiring an application and migrating functionality to other applications.

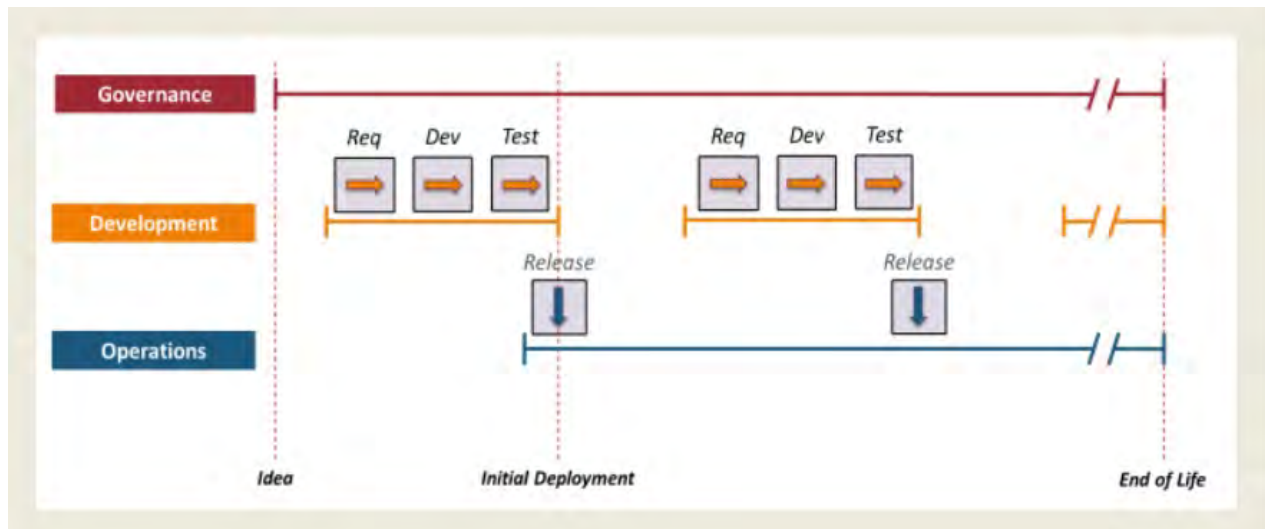
## Development

ALM does not dictate the specific tasks undertaken in a software development process but takes a broader scope and higher level view of the development phase. A typical question that ALM deals with is the choice of software development process that will be used both in development and the process used during deployment for support and enhancement programming. The ALM may specify multiple development process models that are used in a hybridized manner.





ALM is also concerned with the transmission and sharing of information and artifacts among the various individuals and groups (developers, testers and business analysts for example) who will be working with the application at any point in the product lifecycle. The overall goal of ALM is to ensure that everyone has correct and accurate information they need to accomplish their work in a timely and cost effective manner.

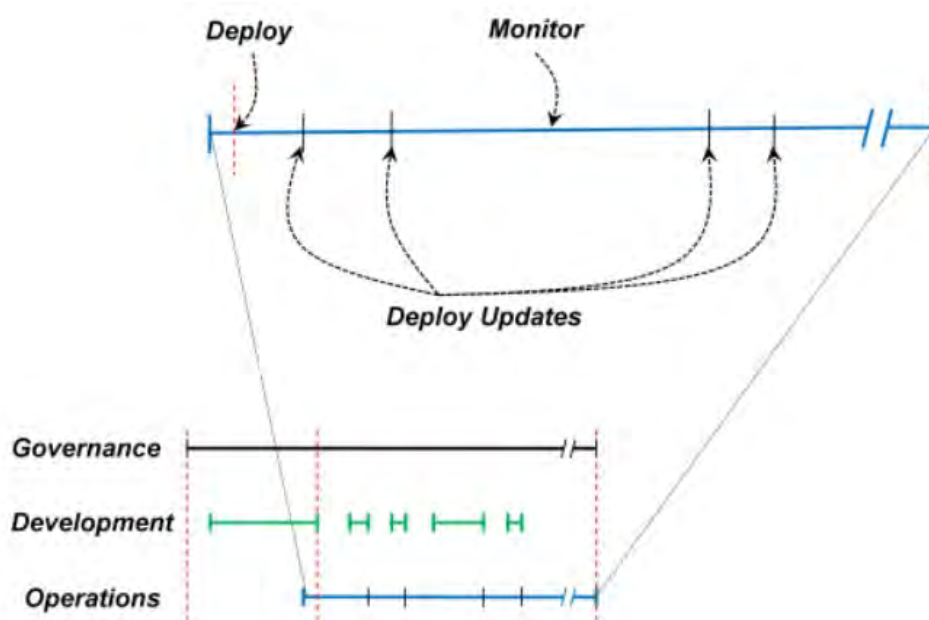


## Operations

During deployment, governance makes decisions and development makes changes. Operations provides the context for making those two threads work together. Operations monitors usage, performance, service level targets, change requests and bug reports. Operations also monitors changes in the application's technical environment, such as operating system changes or architectural changes, that could impact the application.



This data is used by governance to make a decision to effect some change, and by development to quantify and scope the needed changes. Operations also manages the integration of the product into its operational environment, including the installation of any upgrades and enhancements.



## 1.3 Collaborative Lifecycle Management

Collaborative Lifecycle Management (CLM) is the IBM Rational implementation of the concepts and ideas of ALM.

Rational Software originally developed the idea of a lifecycle software development process in the 1990s which eventually became the Rational Unified Process which had the goal of automating the whole of the development lifecycle from end to end. Rational bought a number of products that each automated some aspect of the development activities, like ClearCase for configuration management. However, this buying spree bankrupted Rational resulting in them being bought out by IBM who embraced the Rational vision of end to end process automation in software development and deployment.

The fundamental problem faced in achieving this goal was that while the different products were each the best in their class, they were all made by different companies and were radically different from each other in design and architecture. The end result was they could not be integrated with each other. This was especially problematic for the idea of traceability, or being able to associate, for example, test cases with the requirements they are supposed to validate. Clients using the Rational Suite found it had become more time consuming to get the Rational Suite tools to work together using various work-arounds than to not use the tools at all. Clearly not a desirable state of affairs.

### ***1.3.1 Unified Change Management (UCM)***

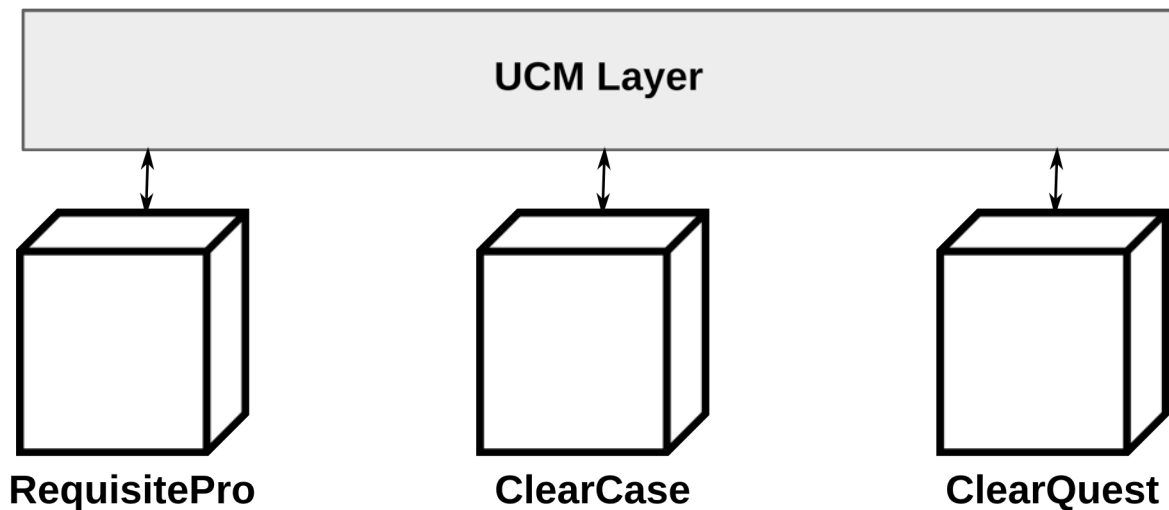
The concept of UCM was to create a high level layer of abstraction where a set of standard concepts, artifacts and tasks would be defined independent of any specific Rational Suite tool. These artifacts would hide product specific implementations of the artifacts and would be used to automatically generate the various product specific artifacts out of sight of the users. As an analogy, think of two different CPUs, that use two different instruction sets and assembly language. Rather than try to port code from one to the other by rewriting the assembly language instructions, code is written in an abstract high level language, like C, and then compiled into the appropriate assembly language, the CPU specific artifact.

The UCM approach proved to be unworkable for several reasons which will be mentioned in the next section, however the concepts that were introduced by the UCM model became the basis for the CLM model which is part of the Jazz Server and RTC design.

The following are some of the basic UCM concepts that have migrated into the IBM CLM model.







## Project

A project is a logical unit that is mapped to the development structure of an application or system. A project contains the configuration information like components, activities and policies that are needed to manage and track the work on a specific product of a development effort, such as an auction Web site or an order fulfillment process for an e-business.

## Component

A component is a group of file and directory elements such as the source code and other relevant files, like images, that are versioned together. The team develops, integrates, and releases a component as a single unit of work. Components constitute parts of a project and projects often share components. Components provide separation of concern and organize elements into well defined entities. Components may be reusable across different projects.

## Baselines

A baseline identifies specific versions of each of the elements in a component that represents the integrated or merged work of team members. It represents a version of a component at a particular stage in project development, such as the first design, a beta release, or a final product release. Throughout the project cycle, the project manager creates and recommends baselines and changes their attributes to reflect project milestones.

## Activity

An activity is an object that records the set of files, called a change set, that a developer creates or modifies to complete and deliver a development task, such as a bug fix. Examples of other activities include an update to a help file or the addition of a menu item to a GUI component. The activity is the result of some change request or other request.

## Work Areas and Streams

A work area is a development area associated with a change that is used by a developer to perform an activity.

A stream maintains a list of activities and baselines and determines which versions of elements developers are working on. Generally developers check out a change set from their stream into their work area and then check it back into their stream where it is merged with the work of other developers who are using the same stream. There are multiple ways to set up streams to support a specific methodology or preferred mode of work.

While it is not important for this course that you know the details of UCM artifacts, what you should know is that they are the conceptual building blocks that CLM projects are constructed from.

### 1.3.2 RQM Artifacts and Terminology

Rational Team Concert (RTC) is the replacement for the Rational Suite and will be discussed in the next section. The various Rational Suite products have been replaced by a set of web based applications that provide the CLM functionality.

There are a few items that we need to understand about UCM/CLM that are important for what we need to do in this course. Most of the software testing specific terminology used in RQM is taken from the “Glossary of Terms Used in Software Testing Version 2.1” published by the International Software Testing Qualifications Board. The full ISTQB glossary is available at <http://www.istqb.org/downloads/glossary.html>.

## Artifact

This is a common term used in RTC generally to refer to

*An entity that is used or produced by a software development process. Examples of artifacts are models, source files, scripts, and binary executable files.*

In the case of RQM, artifacts would be specific test cases, test plans and test reports for example.



Each lifecycle application in Rational Team Concert – requirements, testing and configuration management – works with their own specific types of artifacts, however there are traceability linkages between the artifacts in different components. For example, a requirement artifact in the Requirements Composer Application describes expected behaviour which is linked to a code module in the Configuration Management Application that implements the behaviour, and is validated by a set of tests in the Quality Manager Application that ensure the code performance satisfies the original requirements.

Execution of a software delivery project



If there are changes the requirements, then it is important to be able to update the tests accordingly so that the changes to the code module can be validated correctly. We will look at this process later in the course.

### 1.3.3 Five Imperatives for Application Lifecycle Management

By Carolyn Pampino <https://jazz.net/library/article/637>

#### Overview

Many organizations are faced with hastened delivery schedules due to competitive pressures and the need to innovate. Yet software development is difficult, and the software systems that are maintained and delivered by the world's IT and device development or-

ganizations are astoundingly complex. Teams challenged by reduced time to delivery must do so without increasing their budgets or sacrificing quality. Their strategy, instead, must be to improve software development efficiency. A solution to this dilemma is to improve Lifecycle Collaboration with Application Lifecycle Management.

Designed for the execution of a software delivery project, Application Lifecycle Management solutions coordinate people, processes, and tools in an iterative cycle of integrated software development activities, including planning and change management, requirements definition and management, architecture management, software configuration management, build and deployment automation, and quality management. In addition to the capabilities, the fundamental features of an ALM solution include traceability across lifecycle artifacts, process definition and enactment, and reporting.

The most important benefit of an ALM solution is coordinating the people, processes, information, and tools involved in a project to deliver innovation to your stakeholders. Because there is no one-size fits all solution, we advise our clients to focus on the following imperatives as they implement an ALM approach best suited to their environment and culture:

- Use Real-time Planning
- Establish Lifecycle Traceability of related artifacts
- Enable In-context Collaboration
- Cultivate Development Intelligence
- Practice Continuous Process Improvement

## Real-time planning

We plan because we want to accomplish a goal and want to know when we are done. The only way to know when the work is complete is to ensure the plans are fully integrated with project execution and always up to date. The IBM Rational Solution for Collaborative Lifecycle Management offers fully integrated real time planning.

## Lifecycle traceability

Traceability isn't simply one of those "nice to have" capabilities in the software development lifecycle. Traceability helps you understand what everyone else on the team is doing. For example, while the requirements analyst knows very well what requirements she has written, she still needs to know whether a given requirement will be addressed during a specific development iteration and, if so, which one. Or she wants to know if the implementation of that requirement has been tested and with what result.



An ALM solution that allows for lifecycle artifact traceability helps teams to answer the hard questions about the status of their project. By linking related artifacts, teams are better equipped to answer questions such as "which requirements are affected by defects?" and "which work items are ready for test?"

Traceability helps each team member understand what the rest of the team is doing and how it impacts the overall workload. If you are working in a regulatory compliance environment, traceability helps you answer auditor's questions such as "What changes went into this build, what tests were run and with what result?"

## **In-context collaboration**

Collaboration isn't just about being friendly and collegial with each other. Collaboration contributes to higher quality and improved value to the stakeholders, which means, collaboration is a key to innovation. Collaboration features within an ALM solution can improve a team's ability to connect with each other, to respond to changing events, and to improve project predictability.

Collaboration tools can also help teams focus on what matters. Teams should seek every opportunity to automate manual, non-creative tasks. A good ALM solution enables build and test automation, but automation can also apply to status reporting and information access. Project and personal dashboards play an important role in bringing automated information to the team by providing transparency into their work and access to real-time data with team reports and queries. A well-designed user interface automates access to information, by bringing information to the user instead of forcing a manual 'context switch' to access another application. This form of automation naturally leads to better collaboration.

## **Continuous process improvement**

Process is more than a documented set of procedures. We design processes based on best practices gleaned from industry experience as a means to improving a team's collaboration and to help them succeed. Most behaviour is habitual. When you define or change a process, you are asking an entire team of people to change their habits and adopt behaviours that at first may be difficult to understand. It can be quite hard to change one habit in one person. Yet, process changes frequently require new ways of thinking and new modes of behaviour for a multitude of people. A well-designed ALM solution allows you to change that process incrementally, improve the team dynamic and continue to refine toward greater efficiencies.

## 1.4 Rational Team Concert

As mentioned in the previous section, IBM originally attempted to create a CLM type solution for the legacy products of the Rational Suite by implementing their UCM protocol. However, this idea was abandoned for a very simple reason. Many of the products had been the “best of breed” when they were released, but they were in many cases over 20 years old. The products were obsolete and were based on technology that was also obsolete and difficult to deploy in a modern IT environment.

The decision was finally made to migrate away from ClearCase, ClearQuest and the other products to a more modern platform based on current technology instead of trying to patch together obsolete technology with the UCM band-aid.

### 1.4.1 The Eclipse Architecture

Eclipse began as an in-house integrated development environment (IDE) at IBM called IBM VisualAge, which IBM open sourced and gave to the Eclipse foundation where it became extremely popular among developers in a very short period of time.

One of the main drivers of the popularity of the Eclipse IDE was its “plug-gable” architecture. As described by Wikipedia:

*With the exception of a small run-time kernel, everything in Eclipse is a plug-in. Thus, every plug-in developed integrates with Eclipse in the same way as other plug-ins; in this respect, all features are “created equal”. Eclipse provides plug-ins for a wide variety of features, some of which are from third parties using both free and commercial models. Examples of plug-ins include for Unified Modeling Language (UML), for Sequence and other UML diagrams, a plug-in for DB Explorer, and many more.*

This idea (and success) of the plug-able architectural model became conceptual starting point for a new generation of IBM CLM products.

### 1.4.2 Collaborative Lifecycle Management Architecture Overview

*By Tim Feeny*

#### Overview

The lead ALM solution from IBM Rational software is the Rational solution for Collaborative Lifecycle Management (CLM), which consists of Rational Requirements Composer, Rational Team Concert, Rational Quality Manager and Rational Software Architect with Design Management

The Jazz platform is strategic for implementing the Rational solution for CLM. The Jazz platform is all about collaboration; it allows individual products to integrate better and more easily, and allows the users of those products who take on distinct roles to work



more effectively together than has historically been possible. The Jazz platform supports a more collaborative approach to ALM tools and their users.

That approach is open. It leverages the standards used by the Internet. Just as the internet works with data spread around the world, the approach is to allow development team data to be spread around an organization, and yet be as easily accessible as surfing the world-wide-web. Through the support for OSLC, integrations can be provided for both Jazz-based and classic products. Additionally, third party vendors and customers can consume or provide their own OSLC interfaces to support heterogeneous environments made up of Rational, third party, and/or customer solutions. The combination of Jazz and OSLC provides the ability to have unprecedented collaboration, transparency, automation, and traceability.

Effective ALM can help address current business requirements, enhance productivity with automation, and accelerate decision-making. The key word here is “effective.” ALM is not a one-size-fits-all type of solution; instead, to be effective, you should implement ALM in a way that best suits your development environment and your company’s culture. IBM Rational software professionals have defined Five ALM Imperatives that can help ensure an effective ALM implementation. These imperatives have been developed after more than 20 years of implementing ALM, have been used time and time again, and can be applied to just about any implementation:

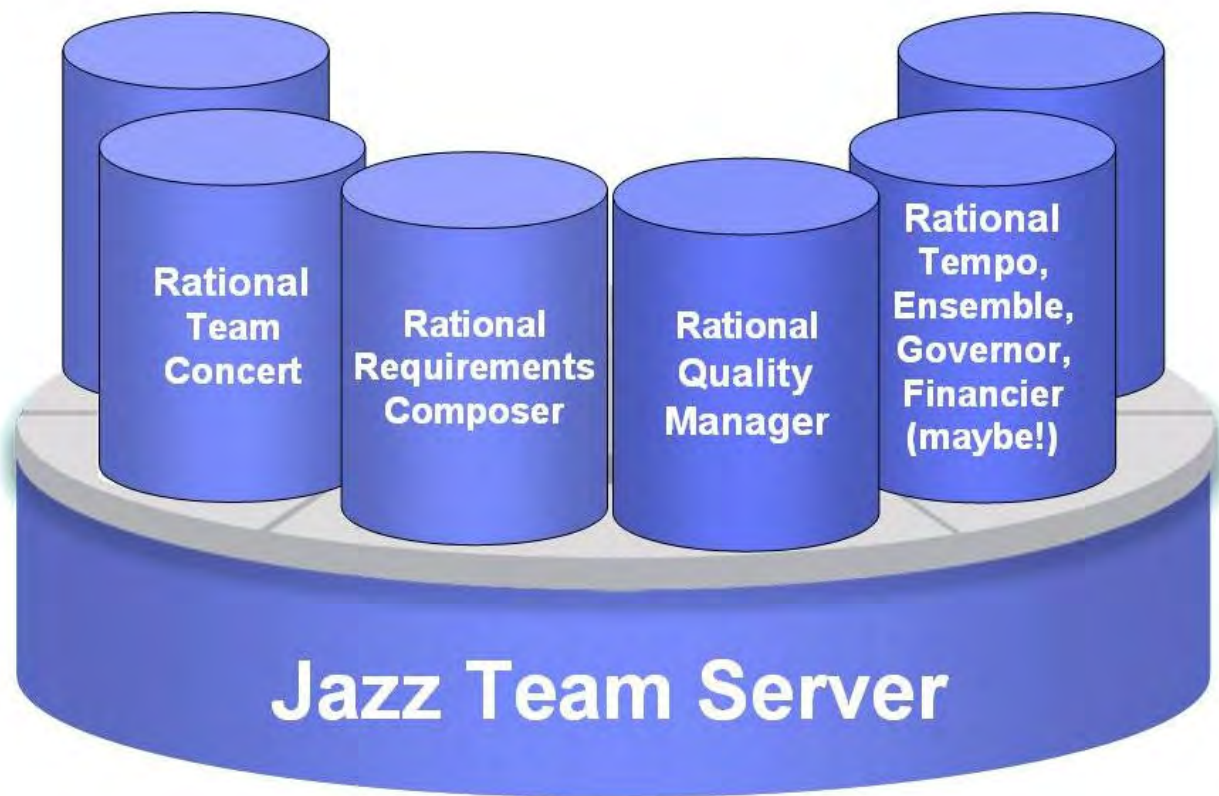
- Maximize product value with in-context collaboration
- Accelerate time to delivery with real-time planning
- Improve quality with lifecycle traceability
- Refine predictability with development intelligence
- Reduce costs with continuous improvement.

In summary, "CLM" expresses the proper scope and differentiated value of our solution. This approach leverages the value that Jazz brings across both IT and systems delivery, delivered via five ALM imperatives.

### ***Solution components***

The Rational solution for CLM, which is covered by this architecture document, is composed of the following applications running on a common Jazz Team Server: Requirements Management (RM), Change and Configuration Management (CCM), Quality Management (QM), and Design Management (DM). These applications are used by a set of products, including the Rational solution for CLM. These products are:





- **Rational Requirements Composer (RRC):** helps teams define and use requirements effectively across the project lifecycle including rich editors for both textual and visual requirements capabilities for lean requirements definition, including the creation of use cases, sketches and storyboards. (Note: In later versions of RTC, the original RRC application has been replaced with the DOORS requirements product)
- **Rational Team Concert (RTC):** integrates change management, source control management, continuous builds (software delivery automation), project planning and reporting/dashboard capabilities.
- **Rational Quality Manager (RQM):** provides a shared test management hub for test planning, creation, and execution as well as workflow control, tracking, and end-to-end traceability.
- **Rational Software Architect with Design Management (RSA-DM):** enables teams to share, collaborate and manage design information across the application development lifecycle.

In addition to these applications, the Rational solution for CLM provides two types of reporting capabilities:



- **Rational Reporting for Development Intelligence (RRDI):** allows creation and viewing of chart and dashboard style reports. It is an optional install in CLM 2012. When installed and configured it provides users with an additional set of predefined development intelligence reports, as well as the ability to author new reports. Rational Reporting for Development Intelligence is based on the Cognos Business Intelligence (BI) platform.
- **Rational Reporting for Document Generation (RRDG):** provides support for generation of document style reports. In CLM 2012, Rational Reporting for Document Generation is included as part of Rational Requirements Composer, Rational Team Concert, and Rational Quality Manager. Authoring functionality for document style reports is provided by a separate purchase of Rational Publishing Engine (RPE).

In addition to these applications, the Rational solution for CLM includes a set of process and practices and supporting tool extensions, all packaged in a Rational Method Composer (RMC) configuration. The CLM processes and practices include everything that you need to run the solution. They describe the context and usage model for the products that support the solution -- step-by-step guidance on how to apply the solution, including product guidance in tool mentors. The practices are self-documenting and include Jazz process templates (including Rational Team Concert process and work item templates) for enacting the solution.

Rational Team Concert, Rational Quality Manager and Rational Requirements Composer share a common installer that deploys the shared Jazz Team Server plus the Change and Configuration Management, Quality Management, and Requirements Management applications. The trial licenses that are installed in the Jazz Team Server provide access to the capabilities of these products. This means that you can install any of these products to enable the full set of collaborative lifecycle management capabilities provided by all the products.

## 1.5 Overview of Jazz Team Server

Jazz Team Server is a Java-based web application that runs on an application server and provides the fundamental framework that the RTC components like RQM plug into. The terms RTC and Jazz Team Server are often used interchangeably because the distinction between them is subtle. Generally, you can think of the Jazz Team Server as the framework that applications like RQM are plugged into, and RTC as the finished product of Jazz Server + Applications.

Jazz Team Server provides the foundational services that enable a group of applications to work together as a single logical server. After you install Jazz Team Server and applications, such as Change and Configuration Management (CCM), Quality Management (QM), and Requirements Management (RM), you run the Server Setup wizard to configure the server. The Server Setup wizard registers the installed applications with Jazz Team Server. Applications that are registered with the same Jazz Team Server can communicate with each other. After you associate project areas from different applications with each other, you can link artifacts across those project areas. For example, you can link requirements in an Requirements Management project area with work items in a CCM project area, and you can link requirements and work items to test plans and test cases in a QM project area.

### The Repository

Jazz Team Server includes an extensible repository that provides a central location for application-specific information. Data is stored in the repository as top-level objects called items. A single repository can contain multiple project areas and their artifacts.

The repository works in conjunction with one of the following supported relational databases:

- Apache Derby: An open source relational database system most suited for small repositories
- IBM Db2: A commercially available product that offers solutions for repositories of various sizes
- Oracle
- SQL Server

However, in this class we will be using what is called the “evaluation” configuration which uses the Apache Derby database.



## User administration

Jazz Team Server provides a single synchronized user database that is shared by the server and all registered applications. You can create and manage user accounts, and assign licenses to users directly in the centralized Jazz Team Server web client or in the applications that are registered with the server. You can also synchronize user records in the repository with user records in a Lightweight Directory Access Protocol (LDAP) directory.

## Project administration

In each application, teams work within the context of a project area. A project area defines the project deliverables, team structure, process, and schedule. Project administrators are responsible for creating and managing project areas. A common administration task is to add users as members of a project area and to assign roles to those members. You can create and manage project areas in each application by using the project area editor. However, a more efficient method is to use the Lifecycle Project Administration user interface, in the Jazz Team Server web client, to create a lifecycle project. When you create a lifecycle project, you select a template that defines the project areas to create, the process to use for each project area, and the associations to establish between those project areas.

After you create a lifecycle project, you can use the Lifecycle Project Administration user interface to add users as members to one or more of the project areas that belong to the lifecycle project. You can also assign roles to those members for each project area.

## Reporting

Jazz Team Server includes a reporting component that can help you track the actions, behaviours and progress of a team or project. Visualizing data about the development process can make certain trends visible that might otherwise be hidden or obscured. By making information available at a glance, reports can enable effective decision-making.

The reports component includes a data warehouse for storing read-only historical and aggregated data. The data warehouse is optimized for efficient queries and quick response times. A report engine generates reports by accessing the data that is stored in the data warehouse.

## Dashboards

Dashboards are a visual web interface component that you can organize to display high-level information about project status. From within a dashboard, you can easily access more complete information. Dashboards visually display how data integrates from multiple Jazz components.



## Module Two

### Overview of Test Management

---

*More than the act of testing, the act of designing tests is one of the best bug preventers known. The thinking that must be done to create a useful test can discover and eliminate bugs before they are coded - indeed, test-design thinking can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest.*

Boris Beizer

*It is not enough to do your best:  
you must know what to do, and THEN do your best.*

W. Edwards Deming

*On two occasions I have been asked, 'If you put into the machine wrong figures, will the right answers come out?'*  
*I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.*

Charles Babbage





## 2.1 Introduction

This module is the only “theory” module in the course and will not refer directly to the RQM product. This module is not intended to be an instructional module in the sense that it is going to teach you how to do test management; that topic alone would take significantly longer than the time allotted for this entire course. Instead, we just want to provide a quick review or overview of some the core concepts of test management which are key to understanding the features and functionality of RQM.

The primary design goal of RQM is to provide integrated automation support for software test management activities and processes. Professional software test management follows a set of standard industry best practices which encompass shared concepts, methodologies, practices, documentation and evaluation criteria. Most of these best practices are described in the ISTQB body of professional knowledge and other standards like the IEEE 829-2008 Standard for Software and System Test Documentation.

For the rest of this module, we will be reviewing some of the high points of these standards or best practices as preparation for actually doing test management with RQM. The key point to remember is that Rational Quality Manager does not “do” test management for you, rather it automates your existing test management process. This means that implementing a poor or defective test management process with RQM will not fix the underlying problem, it will just make them more visible. Or, in the words of the 19th century pathologist Ron Weinstein, “A fool with a tool is still a fool.”

However, automating a robust test management process with RQM will produce significant savings in time and effort as well as administration costs while allowing faster responses to changing testing requirements.

### 2.1.1 Defining Test Management

Test management is project management done in accordance with professional standards and practices. The ISTQB defines test management to be:

*The planning, estimating, monitoring and control of test activities, typically carried out by a test manager.*

If you replace the word “test” with “project,” then the definition is essentially the same as the definition for project management – in this case the “project” being managed is to test a system or application. The basic project management concepts and practices – project planning, resource allocation, measurement and metrics – all appear within the test management vocabulary as well.

The primary difference between test management and project management in general is the more rigorous framework of standards and practices that test management must follow – the rationale for which will be explained in this module and referred to in subsequent modules.

## 2.1.2 Out of Scope Topics

There is a lot more to being a test manager than just being able to manage a test process. According to the ISTQB, the responsibilities of a test manager include:

1. Lead the test management within an organization, project or program to identify and manage critical success factors with management commitment at CEO/Board level
2. Take appropriate business-driven decisions on a test management strategy and implement organization wide commitment and compliance based on quality KPIs
3. Assess the current status of the test management, propose step-wise improvements and show how these are linked to achieving business goals within the organizational context of test management organization or project/program
4. Set up a strategic policy for improving the test management and the testing, and implement that policy in an organization
5. Analyze specific problems with the test management and its alignment with other roles or management areas in the project/organization, and propose effective solutions
6. Create a master test plan with matching governance dashboard to meet or exceed the business objectives of the organization or a project/program
7. Develop innovative concepts for test management (project) organizations which include required roles, skills, methodologies (tools) and organizational structure
8. Establish a standard process for implementing test management in an organization (project/program) with standardized delivery based on quality KPIs
9. Lead an organization to improve the test management process and manage the introduction of changes
10. Understand and effectively manage the human issues associated with test-project management and implement necessary changes

Obviously these topics are out of scope for what we will be discussing in this overview but it is important to keep in mind that being a good test manager does involved more than being able to manage the processes described in this module – in the same way that a good project manager also has to deal with people issues, organizational politics and those other “intangibles.”

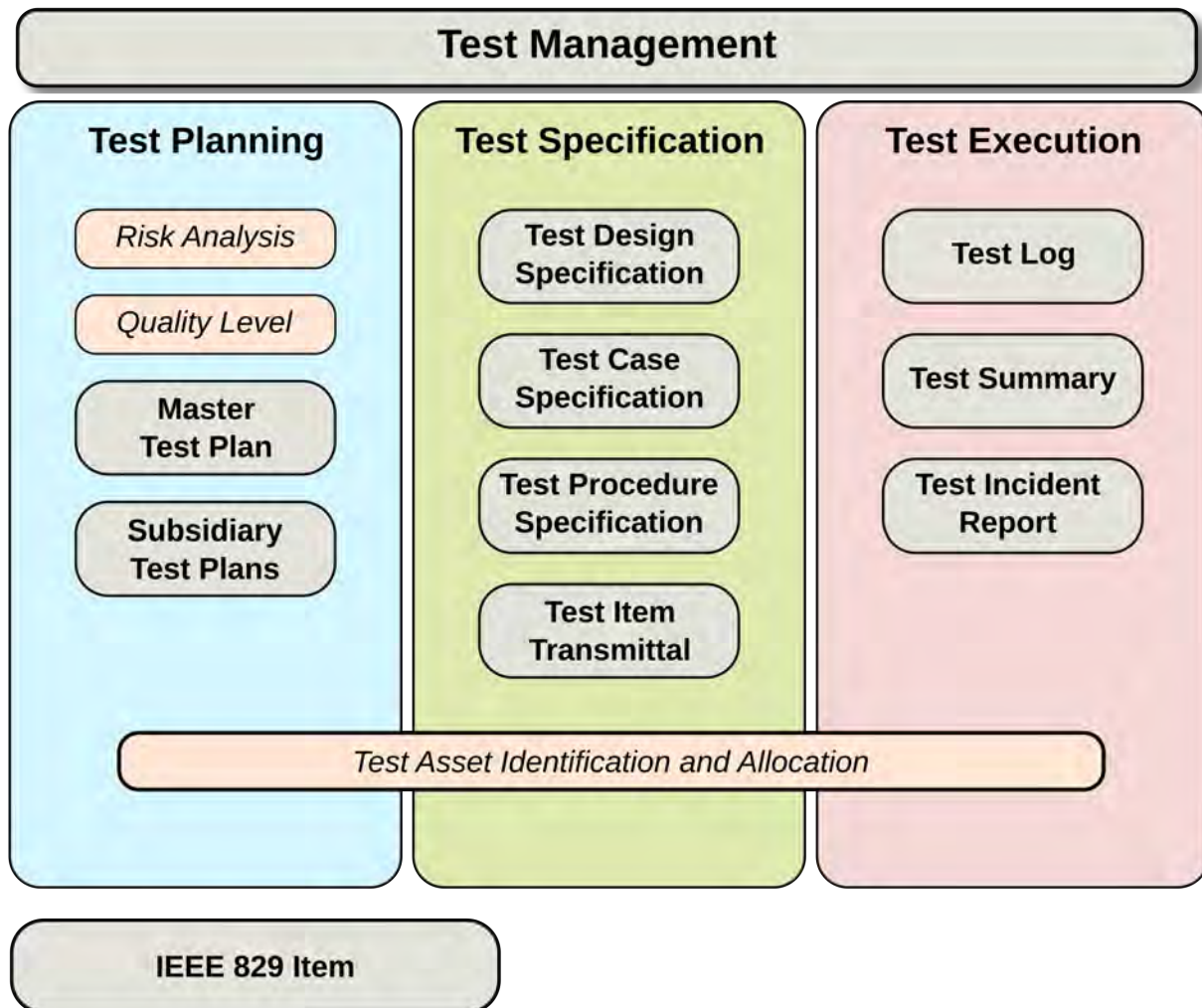
Bjarne Stroustrup, the inventor of C++ made this comment about software development but it holds quite true for project management and test management as well.

*Design and programming are human activities; forget that and all is lost.*



## 2.2 Test Management Processes

The diagram below illustrates the basic processes that are standardized for test management. Most of the process definitions used in RQM are taken from the IEEE 829 standard discussed in a bit later in this section. The processes in the grey boxes are specifically defined in the IEEE standard and are incorporated more or less directly into the functionality of RQM. The other processes are outside the IEEE standard but are also needed for effective test management and are also supported in RQM.



### 2.2.1 IEEE 829 Standard for Test Documentation

This standard, which is available from the IEEE, is the industry standard definition for test management core processes. This 2008 version of the standard is the third with previous versions being released in 1996 and 1983 (referred to as 820-2008, 829-1996 and 829-1983 respectively).

This standard is to be superseded by the ISO/IEC/IEEE 29119 five part standard which was initiated in 2007 and is made up of:

1. **ISO/IEC/IEEE 29119-1:** Concepts & Definitions (September 2013)
2. **ISO/IEC/IEEE 29119-2:** Test Processes (September 2013)
3. **ISO/IEC/IEEE 29119-3:** Test Documentation (September 2013)
4. **ISO/IEC/IEEE 29119-4:** Test Techniques (December 2015)
5. **ISO/IEC/IEEE 29119-5:** Keyword Driven Testing (November 2016)

However, Rational Quality Manager 6.0 uses the terminology and concepts of the IEEE 829 standard so we will stick with that.

## **Transition to Process Standard versus Documentation Standard**

There were some significant changes that took place in the 2008 version of the IEEE 829 standard. According to the standard, the ones that are important to us are:

1. Changed focus from being document-focused to being process-focused in keeping with IEEE/EIA Std 12207.0 TM-1996 a while retaining the structure and format of the standard test documentation.
2. Added the concept of an integrity level to assist organizations in determining a recommended minimum set of testing tasks and concurrent selection of test documentation needed to support those tasks.
3. Added a Master Test Plan (MTP) for documenting the actual management of the total test effort.
4. Added a Level Interim Test Status Report to be issued during the test execution activity.
5. Added a Master Test Report for when there are multiple Level Test Reports that need consolidation. The Master Test Report may also summarize the results of the tasks identified in the Master Test Plan.

The two important implications of this are the codification of a risk or integrity level as part of the testing process and the development of an overall master test plan and a corresponding master test report.

### **2.2.2 Standard as Best Practice**

The IEEE and other standards are derived by describing in a generic sense the practices and methods of expert testers and test managers. When we look at the concept of test maturity or professional test excellence we see that the documentation or processes specified represent and describe the way these experts plan, design and execute their testing.



The standards were originally developed to create a common vocabulary and set of artifacts based on what most professional testers were using in various forms. The goal was to create a paradigm for the testing profession to share expertise, insights and experiences – much in the same way the accounting profession developed the generally accepted accounting standards.

The next few sections will take a look at the underlying professional attitudes, mindset and approaches to testing and test management that the IEEE and similar standards are based in. Remember that the standards are not intended as a set of rules to be obeyed but rather a description of what the experts in the field do.

## 2.3 Software Quality and Testing

When we start to do any sort of testing, we need to be able to set some standard parameters to guide our testing efforts like:

1. How much testing we should be doing?
2. To what level of rigour or detail our testing should be done to?
3. How we know that we are finished testing or have done enough testing?

There are a couple of standard answers to the last question which are often along the lines of:

1. You never stop testing.
2. When you run out of time and money.
3. When they pry it out of your fingers as you fight them off to run just one more test.

Embracing any of these answers is a symptom of a badly managed testing process. There is, in fact, only one right answer – we are finished testing when then our testing plan has been executed and has met the quality control goals for the project.

For example, it seems intuitively obvious that we would want higher levels of quality for the software that runs a heart pacemaker or a nuclear reactor cooling system then we would for an app that updates you on sports scores. Part of that decision is risk, which we will look at later, but also answers to more general questions like:

1. What features are important and should have a higher priority for testing?
2. What features are most frequently used and should be tested first?

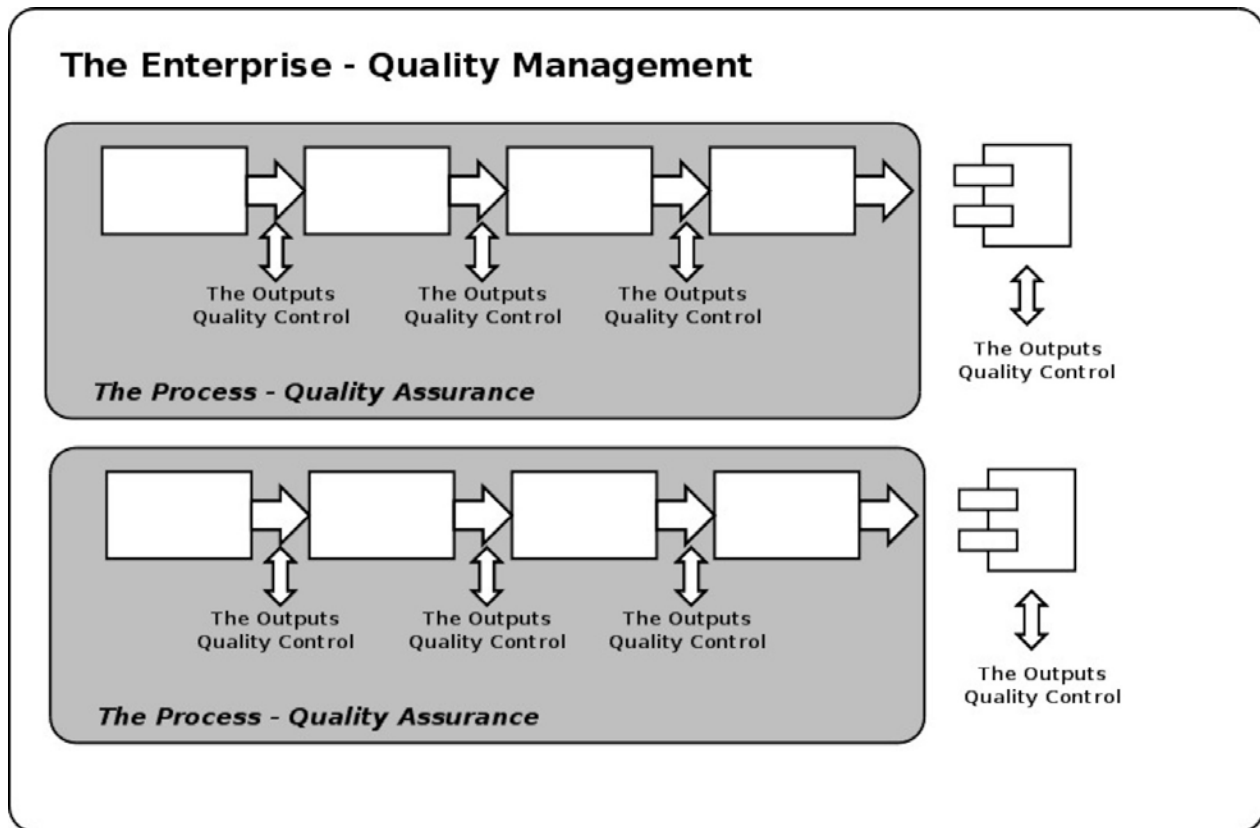
The answers come from our quality assessment which forms the basis for the decisions we make in our test planning and in our test designs.

### 2.3.1 Defining Quality

This is not a course on software quality, but we will introduce a few concepts that are related to both what we will be doing and will also give some insight to the opening comments above.

**Quality management** deals with the culture of the organization – “That's the way we do it here” beliefs of a company – with regards to quality standards and processes, as well as the other dimensions of staffing, education, infrastructure and resources necessary to support a quality program in an organization.





A critical function of quality management is to identify what the expectations of the organization are in terms of quality and then communicate those to stakeholders expressed in terms of a quality mission. The quality mission is a framework and set of principles used by people in the organization to guide their decision making process with respect to quality in their area of responsibility and in the various business processes they participate in. Quality management is the systematic way of ensuring that organized activities happen according to plan and meet the quality expectations as defined in the quality mission.

**Quality control** is the planning and organization of the various kinds of metrics we use when we do testing. Quality control specifies exactly what we should be testing, how much testing needs to be done, and what sorts of results are acceptable for a test to pass. These metrics are used to measure how our quality management efforts are succeeding as well as to provide data to be used in making quality related and other business decisions. For example, if we see that we are getting an unacceptable number of tests failing, then we may use those results to examine the production process to see why so many errors are occurring.

**Quality assurance** is the evaluation and modification of the development and production processes in order to identify where defects are occurring and the causes of those defects so that we can take measures to eliminate the source of the defects, or what we call the root cause of a defect.



**Testing** refers to the actual practices, techniques and procedures we use to do the measuring and evaluation activities required by quality control. While the testing activities can be done anytime and in any context, the processes and practices that make for good testing are always the same. The body of testing knowledge describes how to test, while quality control describes what to test, how much testing to do and how to interpret the results that we get from testing.

## Quality Attributes

Nowadays we think in terms of “quality attributes” which we can define as those properties of a product which will be used by some stakeholder to assess the product's “quality.” Quality is always perceived by stakeholders and is not an inherent property of a product. For the purposes of this discussion, the term “product” is used in the most generic sense to refer to goods, services or anything else.



To stress the importance of the perception of quality, suppose in the example above that the above cell phone plan is offered by a wireless phone company. The quality of the plan, and ultimately the decision to buy the plan, will be based on what the value of the attributes have to each of the two stakeholders.

Philip Crosby in his 1992 book *Completeness* discusses the idea that quality is determined by how it impacts the success of the stakeholders, which he groups into employees, customers and suppliers. But the more importantly, that book represented a shift away from his work of 30 years earlier that emphasized quality as “zero defects” towards a view that we create quality by understanding what various stakeholders expect from a product at a specific time.





Quality, in this view, is a combination of the attributes of the product and how it is perceived through the filter of expectations that stakeholders have for that product. But the reality is that these expectations always change over time depending on the environmental, business and other factors that impact the stakeholders.

Even the idea of fitness for use is a subjective quality assessment since different users may have different ideas of what would constitute “use”. Likewise, conformance to a standard really relies on whose standards we are using.

### 2.3.2 The Good-Enough-Quality Model

The idea behind Good Enough Quality (GEQ) is that even within an organization, we may need to adapt our idea of what we mean by an appropriate level of quality for a particular project at a specific time and within a specific business context. These quality decisions in turn determine the metrics and processes that we use to achieve quality that is “good enough” for that specific project.

“Good Enough” means that we have to choose a level of quality that reduces our risks to an acceptable level, meets our stakeholders expectations in an acceptable manner while still keeping our efforts, costs and time lines reasonable given our resources.

As James Bach notes:

*Here are some of the basic assumptions that I believe are part of the Good Enough paradigm:*

- *We are obliged to cope with a world full of complexity, unknowns, limitations, mistakes, and general imperfection.*
- *People are by far the most variable and vital components of software projects.*
- *Everything has a cost, and what we want always exceeds what we can afford.*
- *Quality is ultimately situational and subjective.*
- *To achieve excellence in something as complex as software, we have to solve a lot of difficult problems, make a lot of trade-offs, and resolve contradictory values. Excellence does not come easily or mechanically.*
- *Software engineering methods are useful to the extent that they are designed with these assumptions in mind.*

*The way we do quality control and how we go about our testing, particularly setting the parameters for testing, is determined by deciding what we need to do to achieve that good enough quality and at what cost. In a sense, our good enough quality can be thought of as the “quality philosophy” that we use to frame our quality control decisions.*

The following are some examples of good enough quality from an article by Paul Szymkowiak and Philippe Kruchten which is a bit tongue in cheek but gives the sense of some kinds of GEQ that we often find in organizations – of course you should note that a lot of these can be dysfunctional – just because you have GEQ level, it doesn't mean you picked a good one.

1. **Not Too Bad** ("We're not dead yet"). Our quality only has to be good enough so we can continue to stay in business. Make it good enough so that we aren't successfully sued.
2. **Positive Infallibility** ("Anything we do is good"). Our organization is the best in the world. Because we're so good, anything we do is automatically good. Think about success. Don't think about failure, because "negative" thinking makes for poor quality.
3. **Righteous Exhaustion** ("Perfection or bust"). No product is good enough; it's effort that counts. And only our complete exhaustion will be a good enough level of effort. Business issues are not our concern. We will do everything we possibly can to make it perfect. Since we'll never be finished improving, someone will have to come in and pry it from our fingers if they want it. Then they will bear the blame for any quality problems, not us.
4. **Customer Is Always Right** ("Customers seem to like it"). If customers like it, it must be good enough. Of course, you can't please everybody all the time. And if a current or potential customer doesn't like the product, it's up to them to let us know. We can't read their minds. Quality by market share?
5. **Defined Process** ("We follow a good process"). Quality is the result of the process we use to build the product. We have defined our process and we think it's a good process. Therefore, as long as we follow the process, a good enough product will inevitably result.
6. **Static Requirements** ("We satisfy the requirements"). We have defined quality in terms of objective, quantifiable, noncontroversial goals. If we meet those goals, then we have a good enough product, no matter what other subjective, non-quantifiable, controversial goals might be suggested.
7. **Accountability** ("We fulfill our promises"). Quality is defined by a contract. We promise to do certain things and achieve certain goals. If we fulfill our contract, that is good enough.
8. **Advocacy** ("We make every reasonable effort"). We advocate for excellence. Throughout the project, we look for ways to prevent problems, and to find and fix the ones we couldn't prevent. If we work faithfully toward excellence, that will be good enough.
9. **Dynamic Tradeoff** ("We weigh many factors"). With respect to our mission and the situation at hand, a product is good enough when it has sufficient benefits and no critical problems, its benefit sufficiently outweighs its noncritical problems, and it would cause more harm than good to continue improving.



### 2.3.4 Beizer's Tester's Mental Maturity

In his book "Software Testing Techniques" Boris Beizer proposed what he called the phases in a tester's mental life which he proposes as an answer to the question "What is the purpose of testing?" His answer is that there is an attitudinal progression characterized by five phases.

1. **Phase 1:** There is no difference between testing and debugging. Other than in support of debugging, testing has no purpose.
2. **Phase 2:** The purpose of testing is to show that software works.
3. **Phase 3:** The purpose of testing is to show where software doesn't work.
4. **Phase 4:** The purpose of testing is not to show anything, but to reduce the risk of not working to an acceptable value.
5. **Phase 5:** Testing is not an activity. It is a mental discipline that results in low-risk software without much testing effort.

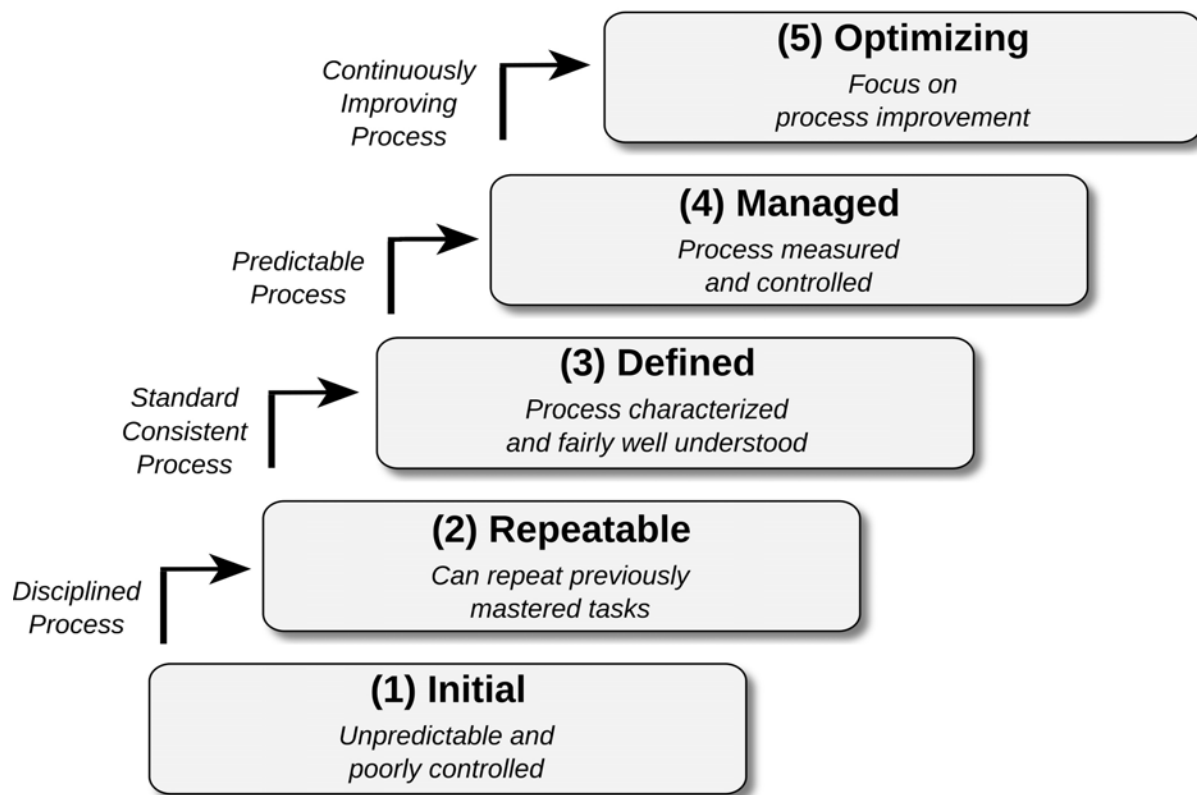
What we see here is an evolving of the testing activity from a rote "run tests, record result" to a careful consideration of choosing testing activities and methodologies that result in the last two phases where the focus is on the quality of the software. But what about the part about "not much testing effort?" That would come from developing highly effective test plans and processes that do exactly the testing needed and when it is needed to ensure the final result meets the correct quality objectives.

What is interesting to note that this evolution in test thinking parallels what happens in organizations as they mature in their testing activities.

## 2.4 Process Maturity

One of the limitations we have as testers is that we can only be as mature in our testing as the developers are in their development process. The Software Engineering Institute (SEI) really kick started the process maturity movement in software with the publication of the Capability Maturity Model or CMM. We are not going to deal with the CMM except in a very superficial way as a lead in to the Testing Maturity Model (TMM).

For example, if the development process is chaotic and never gets around to figuring out what should happen when invalid conditions occur during operation of the software, there is no way to test invalid conditions because we have no idea of what is supposed to happen. Likewise we can't think about how to reduce the risk of software is the risks are not known.



The classic CMM levels are indicated in the graphic. However, there is a more practical way of describing all of the levels when you actually go and identify what characterizes organizations at each level. What we have to keep in mind is that these levels represent the stages an organization goes through over time to master their business and production processes. There are no “bad” levels per se.



Think of the stages of growth of a person. While a person is at the child stage, they are taken care of by a parent, are not expected to have a job nor make important decisions themselves. This stage of maturity is entirely appropriate for a six year old. Where it becomes dysfunctional is when a person is still at this level of maturity when they are 25 years old.

Similarly, all organizations start at level one as start-up. But as they grow, they have to mature or else they tend to not be able to meet the challenges of an established business. A lot of business fail because they experience a lot of growth, but the company is still being managed as a start up – usually because the founder insists on still running the company that way. The founder can make all the decisions when there are three employees at the beginning but not when there are 3,000.

## **Level 1: No Process Level**

In much of the popular literature on the subject, this level is often made to sound like some dystopian, chaotic free for all. However, when you actually take a look at organizations that are at this level they tend to be of two types:

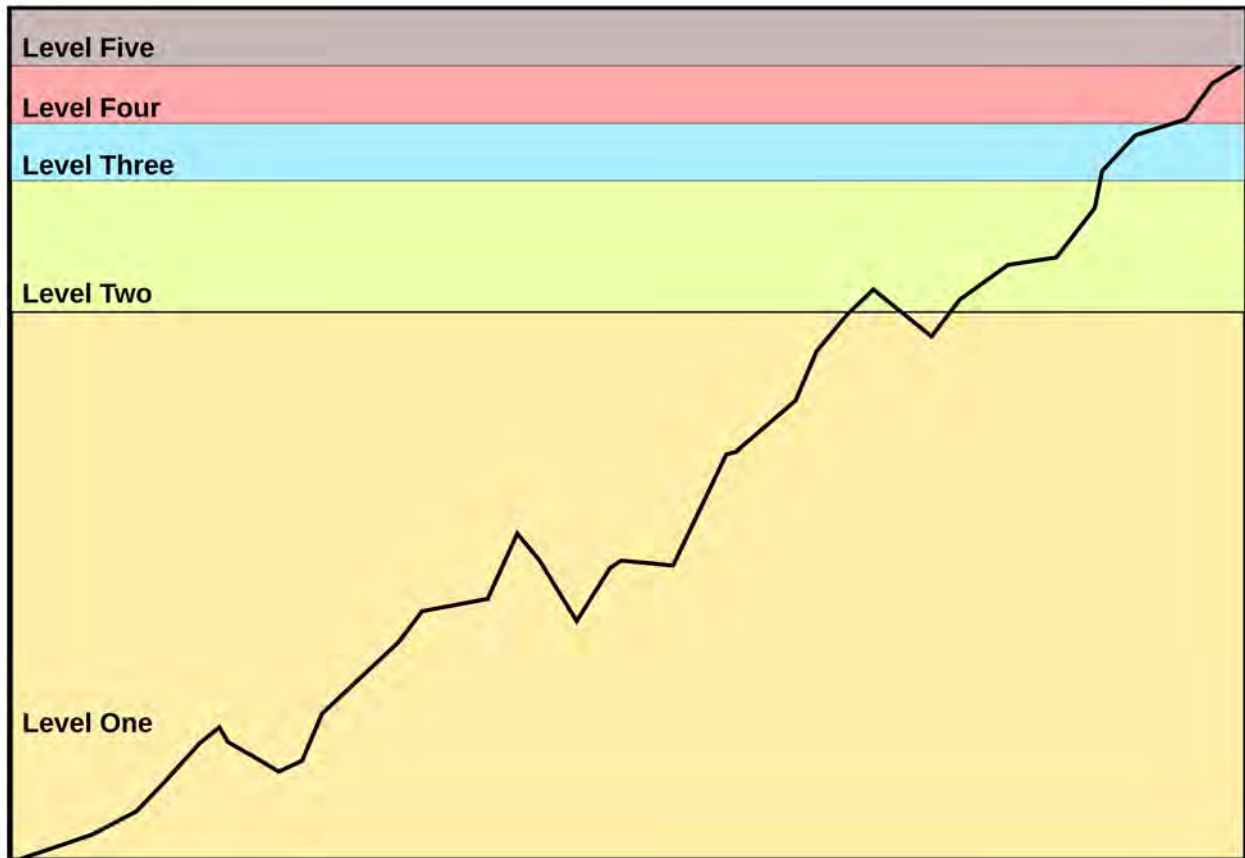
The first type is the start up organization where the emphasis is on the personal skills and productivity of a few key people. It's often counterproductive to focus on a software development process when there are only two people – the programmer and someone running the business side – doing all the work.

The second type is the organization that has grown past the start-up stage and is still hasn't made the transition to process based production. The organization is still being run as a start-up with reliance on key individuals ("We can't let Chani go on vacation because she is the only one who knows how our database is designed.") Eventually, these organizations either have to mature or they tend to fail since they have outgrown the level one entrepreneurial model where one person makes all the decisions and calls all the shots.

## **Level 2: Repeatable Results**

What characterizes organizations at this level is that every project is on time, on budget and to spec. Projects are not late, over budget or defective in any way. The organization has mastered all of the development processes needed to replicate success on every project from project management to vendor management to software testing. It does the right things, and does those things right.

This is also the level that is hardest for most companies to reach since it requires a major reworking of the corporate culture. As one of my clients pointed out "It took us 10 years to get to CMM level 5, but 8 of those years were spent getting to level 2." This is a fairly typical – most organizations spend most of their time on their way to level 5 trying to get to level 2.



### Level 3: Standardized Process

This level is characterized by a standardized process that has various customized implementations for different types of projects; however, what really defines a level 3 maturity is that the process is institutionalized. That means the standardized process is embraced by everyone in the organization as being “the way we do things around here.” Adherence to the process is enforced by the people everywhere in the organization, not by management. In a level three organization if you don’t follow the process, it is your peers that are likely to call you out on it, not your manager.

A main benefit of having a standardized process is that the organization is able to collect data on their process – data that identifies what works and what doesn’t. Having a standardized process (not just defined but actually in day to day use by everyone) is essential for QA to actually be done effectively.

If you think of the organization like a basketball team, level 1 is a bunch of guys showing up at the court and just throwing the ball at the hoop now and then. To get to level 2, each individual player has to master the skills of dribbling, passing, shooting, defending as well as learn the rules of the game. At level 3, we can take those five level 2 players and integrate them into a team that can execute complex plays as a unit and adapt their plays to meet the challenges of each opponent.



## Level Four: Proactive Quality

Once an organization has been at level 3 for a while, it can start to prevent defects from occurring by developing best practices and establishing a review culture. For example, if we know from experience that certain design decisions lead to certain kinds of errors, we can then establish a best practice that says something like “Don’t do x because these bad results will happen, instead do y because that will work.” These best practices are based on the organization’s past experience and what they have learned from previous project which is now accessible through the data that is collected from the standardized process.

Doing reviews evolves in a natural way from having institutionalized standards for defect prevention and associated best practices. When work is done, it is reviewed to ensure that it was done in a way that followed the best practices or standards of the organization. However, these best practices and standards themselves are also constantly evolving to keep the corporate knowledge base of best practices current; something made possible because of the constant collection of data when the process is used.

## Level Five: Optimization

At level 4, work is done then reviewed. At level 5, the best practices and reviews are integrated into the work itself. At level 5 the work is reviewed and tested while it is being done. Consider for example how we use a spell checker. Level 4 is analogous to running a spell checker on your document at various times while writing. At level 5 the spell checker identifies misspellings as you are typing and offers suggestions to immediately correct the word.

At level 5, the integration of best practices into the development activities and the continuous “tuning” of those best practices means the process improves just by using it.



## 2.5 Testing Maturity

In the previous section we took a look at the CMM levels to lead us into this discussion of something called the Testing Maturity Model (TMM) which was developed by Ilene Burnstein, Taratip Suwannasart, and C.R. Carlson, at the Illinois Institute of Technology.

Their reason for doing this was:

*Testing is a critical component of a mature software development process. It is one of the most challenging and costly process activities, and in its fullest definition provides strong support for the production of quality software. In spite of its vital role in software development, existing maturity models have not adequately addressed testing issues nor has the nature of a mature testing process been well defined.*

*We are developing a Testing Maturity Model to address deficiencies these areas. The TMM will complement the Software Engineering Institute's Capability Maturity Model (CMM) by specifically addressing issues important to test managers, test specialists, and software quality assurance staff. The TMM will contain a set of maturity levels through which an organization can progress toward testing process maturity, a set of recommended practices at each level of maturity, and an assessment model that will allow organizations to evaluate and improve their testing process.*

The TMM picks up where the CMM leaves off. The CMM takes as its primary focus the activities and capabilities needed to create software and, as a result, tends to gloss over some the details of software testing. To compensate for this oversight, Burnstein and her colleagues have developed a complementary testing maturity model to “fill in the gaps” when it comes to testing within the CMM framework.

The objectives of Burnstein for the TMM were to define:

1. **A set of levels that defines a testing maturity hierarchy.** Each level represents a stage in the evolution to a mature testing process. Movement to an upper level implies that lower level practices continue to be in place.
2. **A set of maturity goals for each level** (except Level 1), and the activities, tasks, and responsibilities needed to support them. Organizations will strive toward testing maturity by focusing on the goals defined for each level.
3. **An assessment model** that consists of three components: a set of maturity goal-related questions designed to assess test process maturity, a training program designed to select and instruct the evaluation team that is to conduct the maturity assessment, and an assessment method that allows an organization to assess itself based on responses to the questionnaire and interview data.

### 2.5.1 Defining Testing Process Maturity

The attributes of a mature testing process are described below, each supporting one or more of the process maturity criteria described in the CMM. A mature testing process has the following:





1. **A set of defined testing policies.** There is a set of well-defined and documented testing policies that is applied throughout the organization. The testing policies are supported by upper management, institutionalized, and integrated into the organizational culture.
2. **A test planning process.** There is a well-defined and documented test planning process used throughout the organization that allows for the specification of test objectives and goals, test resource allocation, test designs, test cases, test schedules, test costs, and test tasks. Test plans reflect the risks of failures; time and resources are allocated accordingly.
3. **A test life cycle.** The test process is broad-based and includes activities in addition to execution-based testing. There is a well-defined test life cycle with a set of phases and activities that is integrated into the software life cycle. The test life cycle encompasses all broad-based testing activities; for example, test planning, test plan reviews, test design, implementation of test-related software, and maintenance of test work products. It is applied to all projects.
4. **A test group.** There is an independent testing group. The position of tester is defined and supported by upper management. Instruction and training opportunities exist to educate and motivate the test staff.
5. **A test process improvement group.** There is a group devoted to test process improvement. They can be a part of a general process improvement group, a software quality assurance group, or a component of the test group. Since the test process is well defined and measured, the test improvement group can exert leadership to fine-tune the process, apply incremental improvement techniques, and evaluate their impact.
6. **A set of test-related metrics.** The organization has a measurement program. A set of test-related metrics is defined, data is collected and analyzed with automated support. The metrics are used to support the appropriate actions needed for test process improvement.
7. **Tools and equipment.** Appropriate tools are available to assist the testing group with testing tasks and to collect and analyze test-related data. The test process improvement group provides the leadership to evaluate potential tools and oversees the technology transfer issues associated with integrating the tools into the organizational environment.
8. **Controlling and tracking.** The test process is monitored and controlled by the test managers to track progress, take actions when problems occur, and evaluate performance and capability. Quantitative techniques are used to analyze the testing process and to determine test process capability and effectiveness.
9. **Product quality control.** Statistical methods are used for testing to meet quality standards. "Stop testing" criteria are quantitative. Product quality is monitored, defects are tracked, and causal analysis is applied for defect prevention.

## ***2.5.2 Behavioural Characteristics of the TMM Levels***

### **Level 1 - Initial:**

Testing is a chaotic process; it is ill-defined and not distinguished from debugging. Tests are developed in an ad hoc way after coding is done. Testing and debugging are interleaved to get the bugs out of the software. The objective of testing is to show that the software works. Software products are released without quality assurance. There is a lack of resources, tools, and properly trained staff. This type of organization would be on Level 1 of the Capability Maturity Model (CMM) developed by the Software Engineering Institute. There are no maturity goals at this level.

### **Level 2 - Phase Definition:**

Testing is separated from debugging and is defined as a phase that follows coding. It is a planned activity; however, test planning at Level 2 may occur after coding for reasons related to the immaturity of the test process. For example, at Level 2 there is the perception that all testing is execution-based and dependent on the code, and therefore it should be planned only when the code is complete.

The primary goal of testing at this level of maturity is to show that the software meets its specifications. Basic testing techniques and methods are in place. Many quality problems at this TMM level occur because test planning occurs late in the software lifecycle. In addition, defects propagate into the code from the requirements and design phases, as there are no review programs that address this important issue. Post-code, execution-based testing is still considered the primary testing activity.

### **Level 3 - Integration:**

Testing is no longer a phase that follows coding; it is integrated into the entire software life cycle. Organizations can build on the test planning skills they have acquired at Level 2. Unlike Level 2, planning for testing at TMM Level 3 begins at the requirements phase and continues throughout the life cycle. Test objectives are established with respect to the requirements based on user and client needs and are used for test case design and success criteria. There is a test organization, and testing is recognized as a professional activity. There is a technical training organization with a testing focus.

Basic tools support key testing activities. Although organizations at this level begin to realize the important role of reviews in quality control, there is no formal review program, and reviews do not yet take place across the life cycle. A test measurement program has not yet been established to qualify process and product attributes.



**Level 4 - Management and Measurement:**

Testing is a measured and quantified process. Reviews at all phases of the development process are now recognized as testing and quality control activities. Software products are tested for quality attributes such as reliability, usability, and maintainability. Test cases from all projects are collected and recorded in a test case database to test case reuse and regression testing. Defects are logged and given a severity level. Deficiencies in the test process are now often due to the lack of a defect prevention philosophy and the porosity of automated support for the collection, analysis, and dissemination of test-related metrics.

**Level 5 - Optimization, Defect Prevention, and Quality Control:**

Because of the infrastructure provided by the attainment of maturity goals at Levels 1 through 4 of the TMM, the testing process is now said to be defined and managed its cost and effectiveness can be monitored. At Level 5, there are mechanisms that fine-tune and continuously improve testing. Defect prevention and quality control are practised. The testing process is driven by statistical sampling, measurements of confidence levels, trustworthiness, and reliability. There is an established procedure to select and evaluate testing tools. Automated tools totally support the running and rerunning of test cases, providing support for test case design, maintenance of test-related items, defect collection and analysis, and the collection, analysis, and application of test-related metrics.

### ***2.5.3 Maturity Goals at the TMM Levels***

The operational framework of the TMM provides a sequence of hierarchical levels that contain the maturity goals, sub-goals, activities and tasks, and responsibilities that define the testing capabilities of an organization at a particular level. They identify the areas where an organization must focus to improve its testing process.

This section describes the maturity goals for all levels except Level 1, which has no maturity goals.

#### **Level 2 - Phase Definition**

At Level 2 of the TMM, an organization has defined a testing phase in the software life cycle that follows coding. It is planned and repeatable over all software projects. It is separated from debugging, which is an unplanned activity. The maturity goals at Level 2 follow.

1. **Develop Testing and Debugging Goals**

This means an organization must clearly distinguish between the processes of testing and debugging. The goals, tasks, activities, and tools for each must be identified. Responsibilities for each must be assigned. Management must develop plans and policies to accommodate and institutionalize both processes. The separation of these two processes is essential for testing maturity growth, since they are different in goals, methods, and psychology. Testing at this level is now a planned activity and therefore can be managed, whereas debugging cannot.

2. **Initiate a Test Planning Process**

Planning is essential for a process that is to be repeatable, defined, and managed. Test planning involves stating objectives, analyzing risks, outlining strategies, and developing test design specifications and test cases. In addition, the test plan must address the allocation of resources and the responsibilities for testing on the unit, integration, system, and acceptance levels.

3. **Institutionalize Basic Testing Techniques and Methods**

To improve test process capability, basic testing techniques and methods must be applied across the organization. How and when these techniques and methods are to be applied and any basic tool support for them should be clearly specified. Examples of basic techniques and methods are black-box and white-box testing strategies, use of a requirements validation matrix, and the division of execution-based testing into sub-phases such as unit, integration, system, and acceptance testing.



## Level 3 - Integration

This critical maturity level is essential for building quality into software products early in the software life cycle.

At this level, the testing phase is no longer just a phase that follows coding. Instead, it is expanded into a set of well-defined activities that are integrated into the software life cycle. All of the life cycle phases have testing activities associated with them. Integration support associates testing activities with life cycle phases, such as requirements and design.

At this level, management supports the formation and training of a software test group specialists responsible for testing. This group also acts as a liaison with the users or clients, ensuring their participation in the testing process.

Basic testing tools now support institutionalized test techniques and methods. Technical and managerial staff are beginning to realize the value of review activities as a tool for defect detection and quality assurance.

The maturity goals at Level 3 follow.

1. Establish a Software Test Organization

A software test organization is created to identify a group of people who are responsible for testing. Because testing in its fullest sense has a great influence on product quality, and because testing consists of complex activities usually performed under tight schedules and high pressure, management realizes it is necessary to have a well-trained and dedicated group of specialists in charge of this process. The test group is responsible for test planning, test execution and recording, test-related standards, test metrics, the test database, test reuse, test tracking, and evaluation.

2. Establish a Technical Training Program

A technical training program ensures a skilled staff is available to the testing group. Testers must be properly trained so they can perform their jobs efficiently and effectively. At Level 3 of the TMM, the staff is trained in test planning, testing methods, standards, techniques, and tools. At the higher levels of the TMM they learn how to define, collect, analyze, and apply test-related metrics. The training program also prepares the staff for the review process, instructing review leaders and instituting channels for user participation in the testing and review processes. Training includes in-house courses, self-study, mentoring programs, and support for attendance at academic institutions.

### 3. Integrate Testing into the Software Life cycle

Management and technical staff now recognize that test process maturity and software product quality require that testing activities be conducted in parallel with all life cycle phases. Test planning is now initiated early in the life cycle. A variation of the V-model is used by the testers and developers. User input to the testing process is solicited through established channels for several of the testing phases.

### 4. Control and Monitor the Testing Process

According to R. Thayer, management consists of five principal activities: planning, directing, staffing, controlling, and organizing. Level 2 of the TMM introduces planning capability to the testing process. In addition to staffing, directing, and organizing capabilities, Level 3 introduces several controlling and monitoring activities. These provide visibility and ensure the testing process proceeds according to plan.

When actual activities deviate from test plans, management can take effective action to correct the deviations and return to test plan goals. Test progress is determined by comparing the actual test work products, test effort, costs, and schedule to the test plan.

Support for controlling and monitoring comes from standards for test products, test milestones, test logs, test-related contingency plans, and test metrics that can be used to evaluate test progress and test effectiveness.

## Level 4 - Management and Measurement

The principal focus areas at this level are to broaden the definition of a "testing activity" and to accurately measure the testing process.

Controlling and monitoring functions can now be fully supported by an established test measurement program. Staffing activities are supported by a training program. The definition of a testing activity is expanded to include reviews, inspections and walk-throughs at all phases of the life cycle.

Software work products as well as test-related work products such as test plans, test designs, and test procedures are all reviewed. This expanded definition of testing covers activities typically categorized as verification and validation activities.

The primary goal of this broadened set of testing operations is to uncover defects occurring in all phases of the life cycle and to uncover them as early as possible. Review results and defect data are saved as a part of project history. Test cases and procedures are stored for reuse and regression testing.



The maturity goals at Level 4 follow.

1. Establish an Organization-Wide Review Program

At TMM Level 3, an organization integrates testing activities into the software lifecycle. At Level 4, this integration is augmented by the establishment of a review program. Reviews are conducted at all phases of the lifecycle to identify, catalogue, and remove defects from software work products and to test work products early and effectively.

2. Establish a Test Measurement Program

A test measurement program is essential in the evaluation of the quality of the testing process, to assess the productivity of the testing personnel and the effectiveness of the testing process, and for test process improvement. Test measurements are vital in the monitoring and controlling of the testing process. A test measurement program must be carefully planned and managed. Test data to be collected must be identified; how they are to be used and by whom must be decided. Measurement data for every test lifecycle phase must be specified. Measurements include those related to test progress, test costs, data on errors and defects, and product measures such as software reliability.

3. Software Quality Evaluation

One purpose for software quality evaluation at this TMM level is to relate software quality issues to the adequacy of the testing process. Software quality evaluation involves defining measurable quality attributes and defining quality goals to evaluate software work products. Quality goals are tied to testing process adequacy because a mature testing process must lead to software that is at least correct, reliable, usable, maintainable, portable, and secure.

## **Level 5 - Optimization, Defect Prevention, and Quality Control**

Several test-related objectives are at the highest level of the TMM. Organizations at this level test to ensure the software satisfies its specification and is reliable and that the organization can establish a certain level of confidence in its reliability. There are tests to detect and prevent faults. Prevention applies to requirements, design, and implementation faults.

Because the testing process is now repeatable, defined, managed, and measured, it can be fine-tuned and continuously improved. Tool support is available to collect and analyze test-related data. Test planning and test execution also have tool support. Through achievement of the maturity goals at TMM Levels 1 through 4, the testing process is now planned, organized, staffed, controlled, and directed.

Management provides leadership and motivation and supports the infrastructure necessary for the continual improvement of product and process quality. Test-related measurements help suggest and evaluate test process improvement procedures, methods, tools, and activities. Changes can be monitored and managed.

The maturity goals at Level 5 follow.

1. Application of Process Data for Defect Prevention

Mature organizations are able to learn from their history. Following this philosophy, organizations at the highest level of the TMM record defects, analyze defect patterns, and identify root causes of errors. Techniques such as Pareto diagrams are used [6]. Action plans are developed, actions are taken to prevent defect recurrence, and there is a mechanism to track action progress. At TMM Level 5, defect prevention is applied across all projects and across the organization. A defect prevention team is responsible for defect prevention activities, interacting with developers to apply defect prevention activities throughout the life cycle.

2. Quality Control

At TMM Level 4, organizations focus on testing for a group of quality-related attributes such as correctness, security, portability, interoperability, usability, and maintainability. At TMM Level 5, organizations use statistical sampling, measurements of confidence levels, trustworthiness, and reliability goals to drive the testing process.

The testing group and the software quality assurance (SQA) group consist of quality leaders who work with software designers and implementors to incorporate techniques and tools that reduce defects and improve software quality.

Automated tools support the running and rerunning of test cases and defect collection and analysis. Usage modeling is used to perform statistical testing. The cost to achieve quality goals is measured relative to the cost of not testing for quantitative quality goals.

3. Test Process Optimization

At the highest level of the TMM, the testing process is subject to continuous improvement across projects and across the organization. The test process is quantified and can be fine-tuned so that capability growth is an ongoing process. An organizational infrastructure exists to support this continual growth. This infrastructure, which consists of policies, standards, training, facilities, tools, and organizational structures, has been put in place through the goal achievement processes that constitute the TMM hierarchy.





Optimizing the testing process involves

1. Identifying testing practices that need to be improved.
2. Implementing the improvements.
3. Tracking improvement progress.
4. Continuous evaluation of new test-related tools and technologies for adaptation.
5. Support for technology transfer.

## 2.6 Defining Good Testing

Software testers are often stereotyped as obsessive compulsive individuals who are ruled by procedures and a fascination with minutiae and detail. This definitely is not true; all different types of people work as testers and there is no "testing" personality type that characterizes someone who works as a software tester. However, it is an accurate description of the way in which professional testers approach their work – not just in the world of software testing but in the any area of testing, whether it is structural engineering, medical testing, criminal forensics or some other area of testing practice.

Testers often speak of a “testing mindset” or “putting on their testing hat” to describe how they approach a testing problem. This sort of professional mindset is not unique to testers; many professions talk about having to be in a particular mindset to do their work effectively.

When I was doing some anthropological investigation in to the culture of the medical profession, I noticed doctors often talked about their patients in a very dispassionate and depersonalized manner (“I just saw the heart attack in room 207 and now the gall bladder is being prepped.”) When I asked doctors why they seemed to be so cold and remote when it came to dealing their patients, I got very similar explanations from many of them. The following from one doctor is typical of what I was told.

*I have a responsibility to provide the best possible medical care to every patient I see. I can't do that if I get emotionally involved because then I can't make good medical decisions. I'm a parent and when I walk into the emergency room and see a little boy the same age as my son with his face all mangled because of a car accident, my instinct is to go to pieces and break down and cry, but he needs me right then and there to do everything I can to rebuild his face. I have to depersonalize the job because I have to be thinking about the best surgical method to reattach his nose to his face rather than feeling traumatized by his suffering. After I'm done my job and he has gotten my best efforts, I go home and hug my own son and become human again.*

Doctors often say they can't and won't treat members of their own family because their emotional attachments are so strong that they can't “be a good doctor” to them. Medicine is not the only profession we see this – similar mindsets are found in first responders, members of the military and a range of other professions.

### 2.6.1 Impact of Poor Testing

Just like doctors and other professions have to be concerned about ensuring they do the best job possible, so it is with testers. If we are testing something, we are attempting to make sure it works the way it should and that the risks associated with its functioning are identified and minimized. Testers are very aware that they are the last line of defence to prevent the failure of whatever they are testing having dangerous or catastrophic results.



In Agile testing, this is even more critical because if we produce bad tests, they will be used to guide development which will inevitably lead to a failure. The responsibility to ensure our testing is of the highest possible quality is even more important as we become Agile testers.

## The Therac-25 Case

One of the classic software issue is the infamous Therac-25 incident, which has been one of the most studied software failures on record. As described in Wikipedia:

*The Therac-25 was a radiation therapy machine... involved in at least six accidents between 1985 and 1987, in which patients were given massive overdoses of radiation. Because of concurrent programming errors, it sometimes gave its patients radiation doses that were hundreds of times greater than normal, resulting in death or serious injury.*

One of the findings of investigators was that these errors were the result of poor engineering practices but there was also inadequate testing done so that the errors managed to slip through into the delivered production models. There was a breakdown in the way the testing in this risk critical system was designed and tested. Some of the findings of subsequent investigations:

1. The software was designed so that it was realistically impossible to test it in a clean automated way.
2. The code was not independently reviewed or tested.
3. The design of the software was not examined in determining how it might fail.
4. There was poor evaluation of how the error system worked: "The system noticed that something was wrong and halted the X-ray beam, but merely displayed the word "MALFUNCTION" followed by a number from 1 to 64. The user manual did not explain or even address the error codes, so the operator pressed the P key to override the warning and proceed anyway."
5. There was no configuration testing of the Therac-25 with the various combinations of software and hardware until it was assembled at the hospital.

Failures of this system resulted in a number of deaths that could have been avoided if the testing of the system had been done consistently with what are considered the professional best practices for software testing. However, I will also suggest that if an Agile testing approach had been followed, these errors would probably have been picked up during the development of the software.

### 2.6.2 Risk Mitigation

There are many cases similar to the Therac-25 case where missed errors have resulting in the failure of software systems that have significant negative impacts and consequences. For example, airline computer systems regularly cause air travel chaos,

crashes of airplanes due to software errors, bank systems being unavailable causing economic havoc and many more situations.

Recall from a previous module that as we get more “mature” in our mental life as a tester, we go from running tests to thinking in terms of minimizing risk. The testing profession, not just software testers but all kinds of testers, understand that the role of their profession is to avoid buggy or poor quality products getting into use where they can have catastrophic outcomes. Hence the emphasis on “doing testing right.”

For many years, there was no significant impact if software failed, we could just do ctrl-alt-delete or reboot the system and carry on. However, software is now what we call “mission critical” in the words of Grady Booch who defines it this way:

*[Industrial strength software are].. applications that exhibit a very rich set of behaviours, as, for example, as in reactive systems that drive or are driven by events in the physical world, and for which time and space are scarce resources; applications that maintain the integrity of hundreds if thousands of records of information while allowing concurrent updates and queries; and systems for the command and control of real-world entities, such as the routing of air or railway traffic. Software systems such as these tend to have a long life span, and over time, users come to depend on their proper functioning.*

In this sense, there is a general understanding in the testing profession that if software testers fail in doing good testing, then bad things can happen, including people dying. In the next section, we will address the issues of what exactly “good” testing is.

### 2.6.3 Good Testing

In the testing profession generally, not just software testing but also engineering, medical and other fields where testing is done, there are four primary criteria for good testing.

1. **Validity.** Testing is valid when we are actually testing what we think or claim we are testing.
2. **Accuracy.** Testing that is accurate have low number of false positives and false negatives. We will explain what those are in a moment.
3. **Reliability.** Testing is reliable when we are assured that the results of the test depend only on the responses of what is being tested to the tests and not to some other factor such as how the tests are being run.
4. **Comprehensiveness.** Comprehensive testing means that we have done adequate testing and nothing has “slipped through the cracks.”



## Validity

Testing validity means that we are actually testing what we think or assume we are testing.

For example, we may decide to measure intelligence. A test could be designed that measures intelligence by multiplying the circumference of a person's head by the square root of their age. This is a test and produces results that we can manipulate statistically. We can even produce visually stunning graphics describing the results, but the test is not valid because it does not actually measure intelligence.

Consider a more realistic example: the polygraph device is called “a lie detector,” but doesn't measure a person's truthfulness. Instead it measures certain physiological responses such as respiration, heart rate and galvanic skin response under the hypothesis that when people lie, they get nervous which is manifested in sweaty palms, rapid breathing and other effects.

However, the hypothesis is flawed. Psychopaths do not get nervous lying and almost anyone can be trained to lie without showing signs of nervousness. On the other hand, people answering truthfully can be made nervous by other factors including intimidation by the questioner or just the experience of taking a polygraph test.

In fact, the standard advice given by experts is to not take a polygraph if you are innocent because you might fail for other reasons, but if you are guilty, take one because you might pass it. The polygraph is not a valid test of truthfulness.

Tests may be subtly invalid which means that when we look at the tests they appear to test some feature of the system but when we examine what is actually happening when we run the test, we find that the result of the test is due to some other part of the system and not the feature we thought we were testing, as in the following example.

*Testers for the login feature developed a series of tests to ensure that the login feature would reject any user id that was not in the data base. A test case was developed that used a made up id and password and the test passed and it was assumed that the underlying feature was able to correctly identify that a user id was not in the credentials database. However on closer examination of the output of the logs associated with the test runs, it was discovered that the made up password violated the rules for a well formed password (one capital letter, one digit..etc) and the login failed because of an illegal password and the login attempt was rejected BEFORE the credential lookup occurred.*

*Testers assumed that they were testing the credentials lookup but in fact they were testing the password validity check. Realizing their mistake, the test password was changed and suddenly a test that had previously passed now failed. The original test as constructed was invalid.*

## Accuracy

Consider the following matrix of all possible outcomes for running a single test. To keep it simple, we will assume that we are working with a single test intended to identify a particular class of faults. In testing terminology we say that the test passes when it does not detect a fault and that it fails when it detects the fault.

	<i>Fault</i>	<i>No Fault</i>
<i>Test Passes</i>	<b>False Negative</b>	<b>Good</b>
<i>Test Fails</i>	<b>Good</b>	<b>False Positive</b>

The two desirable cases occur when the test works exactly as it should: either there is a fault and our test fails telling us a fault exists, or the test passes because there is no fault.

### ***False Negatives***

A false negative occurs when there is a fault and the test passes or fails to detect the fault. The effects of a false negative can be catastrophic. For example:

1. A bridge passes its structural inspection since one of the tests passed instead of showing a structural failure. Several weeks later the bridge collapses during rush hour killing twice.
2. A standard AIDS test does not pick up a new mutation to the HIV virus. A carrier of the new mutation is declared to be HIV free and proceeds to spread the new mutation within the community.
3. The carbon monoxide detector fails to detect toxic levels of CO in the room.

A lot of the activity that we do in software testing is trying to design tests that minimize the possible false negatives that could possibly occur. We tend to be more sensitive to false negatives because, as examples tend to indicate, false negatives often result in potential catastrophic failures.

### ***False Positive***

A false positive takes place when our test fails, meaning it indicates a fault exists but in reality there is no underlying fault. The consequences of a false positive are not nearly as dramatic as a false negative (no bridge ever collapsed because our test said there was a failure that didn't really exist), however the consequences can be just as problematic.

For example, the standard \$2 roadside drug detection test used by many US law enforcement agencies has, according to investigative reporting by ProPublica:



*[The test] uses a single tube of a chemical called cobalt thiocyanate, which turns blue when it is exposed to cocaine. But cobalt thiocyanate also turns blue when it is exposed to more than 80 other compounds, including methadone, certain acne medications and several common household cleaners.*

In other examinations of the test kit, there were several instances in which the test indicated the presence of cocaine but no sample had even been added to the tube. The problem is that these false positives have led to a number of innocent people being sent to jail, lives destroyed and, in the longer term, a massive potential liability for various governments who use the tests from those who have been wrongly convicted.

False positives can be costly just from an operational perspective. The false identification of an error can lead to expensive and time consuming debugging or investigative procedures that are essentially a waste of time – looking for errors that don't exist. These costs are compounded if we have to do recall or expensive upgrade process to try and fix something that was never broken.

However, most of the damage that a false positive does occurs when it results in a business decision that itself has negative consequences. False positives can result from either a flawed test or the way in which the test is run. Consider this report from CBS news.

*Four years ago, a Houston teenager named Josiah Sutton was convicted of rape and sentenced to 25 years in prison. He was convicted primarily on the basis of DNA tests performed by the Houston police department's crime lab.*

*Anna Werner and David Raziq decided to investigate. They dug up transcripts and lab reports from several cases and sent them to a group of experts, including University of California criminology professor William Thompson, who has reviewed DNA evidence from labs all over the country. w did the Houston police crime lab stack up?*

*"It's the worst I've seen," says Thompson. "This doesn't meet the standard of a good junior high school science project."*

*His findings convinced police to have the DNA from Sutton's case retested at an independent lab. That test was conclusive: Sutton's DNA was not found in the sample. That convinced a judge who ordered him released on bail. Today, he is waiting to see if he will be pardoned.*

*But the story doesn't end there. Hundreds of other convictions in Houston are now being called into question. Harris County district attorney Chuck Rosenthal and assistant DA Marie Munier must review 10 years worth of cases their office has prosecuted using evidence from the lab.*

The realization that the testing process had produced at least one false positive had enormous repercussions. Not only does it cast into doubt the correctness of all the other positive test results, it has opened up the possible for massive damages from lawsuits brought by those who may have been wrongly convicted on the basis of a flawed test.



## Reliability

Testing is said to be reliable when the results of the tests depend only on what is being tested. For example, if a set of tests are performed a number of times and the results depend on when the tests are run or who runs them, then the tests are not reliable.

If I run the tests this morning and find an error and my colleague runs the same tests this afternoon on exactly the same item under test and there is no error found, then is there an error or not? Do I have a false positive or false negative for one of the results?

The most common cause of unreliable test results is the failure to follow a process that can be replicated to verify the results later, which includes not using the same tests when we repeat the testing process. Just like in science where an experiment has to be able to be replicated in order to check the results, so should any form of testing.

When testing is done in a non-repeatable manner there is no way to evaluate how accurate the tests actually are. For example the \$2 road side cocaine tests are not just inaccurate but unreliable because:

*The tests use three tubes, which the officer can break in a specific order to rule out everything but the drug in question — but if the officer breaks the tubes in the wrong order, that, too, can invalidate the results. The environment can also present problems. Cold weather slows the color development; heat speeds it up, or sometimes prevents a color reaction from taking place at all. Poor lighting on the street — flashing police lights, sun glare, street lamps — often prevents officers from making the fine distinctions that could make the difference between an arrest and a release.*

It should be apparent that there is a strong link between reliable tests and accurate tests, but the terms do refer to different aspects of the testing process. Accuracy is a property of an individual test which means that we can execute the testing process flawlessly but we may still get a false positive or false negative. On the other hand, reliability is about how we execute the tests. We might have a highly accurate test but if we execute it in an unreliable manner, we can get false positives or negatives.

## Comprehensiveness

This is actually a very important and central issue in testing and deals with how much testing we do, what sort of testing and to what level of exhaustiveness. A very common problem with non-comprehensive testing is that things “slip through the cracks.” We forget about things that might happen and focus only on the things that probably will happen. For example from CNN money:

*Steve Valdez knew he'd face scrutiny when he went to a Bank of America branch to cash a check written to him by his wife, since he doesn't have a personal account with the bank.*

*Although the Tampa, Fla., resident brought two forms of picture identification, the teller said he needed to provide a thumbprint to cash the check.*

*The problem: Valdez was born without arms and wears a prosthetic. He pleaded with the bank manager to use the photos, but to no avail.*

*"It was really unfathomable," said Valdez, who was told that he could either bring in his wife or open an account of his own.*





*A thumbprint is a longstanding requirement for any non-account holder to cash a check in order to prevent check-cashing fraud, according to a BofA spokesman, who admits that the bank should have offered alternatives to Valdez.*

Comprehensiveness is often described as the problem of knowing how much testing to do so we don't wind up with our own "Steve Valdez" case and yet not doing so much testing that we use up massive amounts of resources for rather small decreases in risk. We will return to the topic of comprehensiveness throughout this course.

## 2.7 Sources of Testing Errors

By this point it should be very apparent that the Agile tester bears the critical responsibility for engaging in high quality testing activities that are as error free as possible. Error we make now can impact the overall success of the project.

However, we are fortunate that a substantial body of work has already been done that provides a framework for us to ensure high quality testing. This framework encompasses all of the protocols, procedures and techniques that testers have developed over the last half century or so to ensure high quality testing. This is a very important point to understand – we do not toss out all of our traditional testing methods and protocols since they were designed to help us avoid testing errors, instead we double down on them and rely on them even more so that we did in the traditional model.

The rest of this section explore the idea of how we eliminate sources of error in our testing

There are three kinds of errors that can result in poor testing:

1. **Test Design Errors.** These are errors that are made in designing tests, choosing test data, selection of pass-fail criteria, composition of test suits and other structural and logical errors associated with test cases, scripts, scenarios and data.
2. **Test Procedural Errors.** Even the tests are free of any test design errors, we can still get poor test results from executing tests incorrectly. Test procedural errors are mistakes in the actual running and analysis of the tests.
3. **Human Errors.** There are certain sorts of errors that testers make simply because we are people, which includes social influences, perceptual errors and cognitive errors. The one problem with human errors is that we cannot just decide not to make them – they are an artifact of how we think and interact with the world.

Since we are assuming that you already understand testing basics, we will not go into the first two points in this course – those topics are more appropriately dealt with in a standard testing techniques course. We will not address the human errors in detail beyond this module, but it is important to understand how we deal with them in testing, especially when we are working in a collaborative environment.

### 2.7.1 Human Errors

Human errors are generally artifacts of our cognition. We cannot decide not to make these errors since we are often totally unaware of when we make them. What we can do is create protocols for testing that eliminate the opportunity to make those errors.

By analogy, consider someone who wants to quite smoking. There are two general approaches to becoming smoke free. The first is to just decide to stop smoking, which often doesn't work because it depends on the person deciding to do something, and the deci-



sion making process might be flawed (“I’ve been smoke free for three days so one little puff shouldn’t hurt...”).

The other approach is remove the opportunity to make that sort of potentially bad decisions. In this case, the person may move into a totally smoke free environment where there is no opportunity to smoke. No matter what the person decides to do, they won’t be able to smoke because they have removed the opportunity to make that decision.

The following are a sample of just a few of the sorts of human errors we need to be on the look out for.

## Self Fulfilling Prophecy

This is also known as the “experimenter effect” and is one the primary reasons that we use double blind experiments. The self-fulfilling prophecy works this way:

1. The tester or someone the tester reports to has certain expectations of the outcome of the testing based on his or her beliefs, such as “This component is bug free.”
2. Those expectations may be communicated to others through subtle cues, usually subconsciously.
3. People, including the one sending the cues, tend to respond to these cues by adjusting their behavior to satisfy the communicated expectations.
4. The resulting effect of the adjusted behaviour is that the original expectation becomes true

In other words, a prediction is made based on a belief, subsequent behaviour becomes biased to do things that are consistent with that belief, then the outcomes of those behaviours validates that belief. The important aspect of this effect is that it happens entirely subconsciously – you cannot decide to just not do it because your mind just doesn't work that way.

Generally we find that this effect seems to centre around making subjective decisions – if we remove the opportunity to make decisions, consciously or unconsciously, the can mitigate the effect.

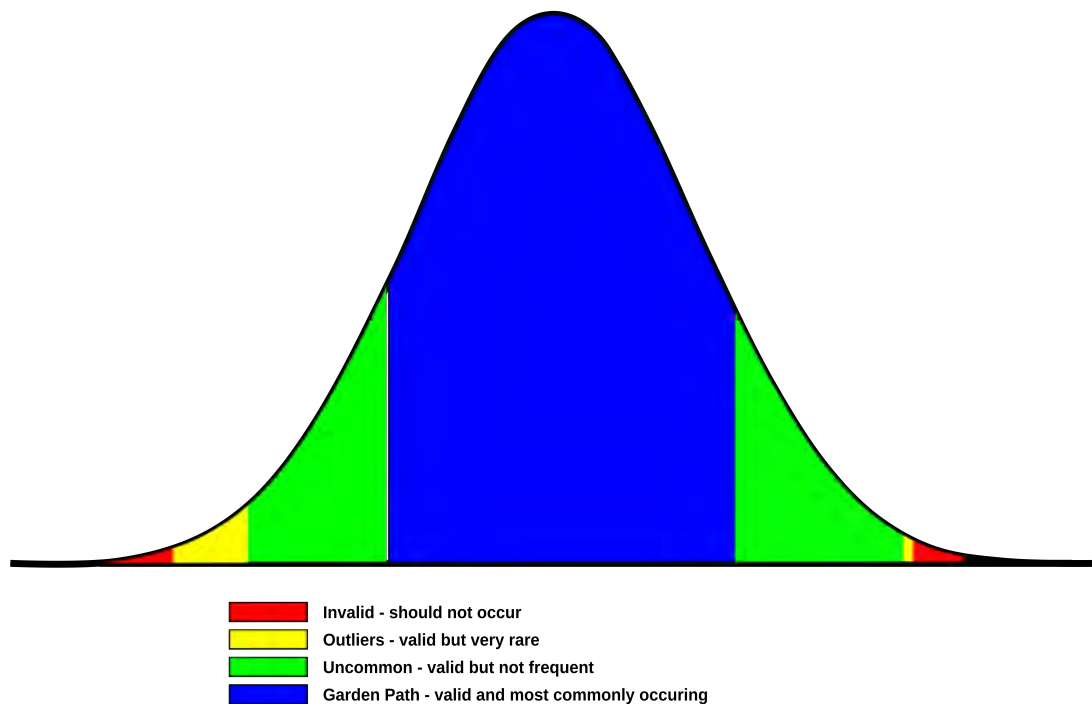
For example, the experimenter effect in social and medical sciences is well known. If we are doing a drug trial and the doctor who is administering the treatments knows who is getting the drug and who is getting a placebo, then her expectations will influence the results. The proposed explanation is that the doctor subconsciously treats the subject differently, perhaps not even being aware that they are doing it, and this is picked up the subjects who then manifest the placebo or nocebo effects depending on what they now subconsciously believe about what they are being administered.

In a double blind study, the doctor is unaware of which patient is getting which treatment. In effect, we have removed the opportunity for the doctor to treat the patients in a differential manner, and we see much more accurate and reliable results.

In fact, this effect is so well known that a study in the social or medical science is often rejected immediately if it is not a double blind study.

## Garden Path Thinking

Another trap that testers can fall into is to test with data representative of the typical or most commonly expected valid data as opposed to unexpected or invalid data. Test data which may reveal the existence of program deficiencies are ignored because “that case would never happen in real life”. In other words, the tester chooses the “garden path” or most “normal” form of data.



Testing with data which is “realistic” or is statistically most representative of the mainstream cases is not as useful in finding bugs in programs as the extreme cases.

This deficiency in testing can also arise from inaccurate generalizations about the population the input data will come from based on a non-representative sample. This is most likely to occur when the tester is unfamiliar with the data domain.

This sort of thinking often misses “outliers” which are valid cases that are very statistically infrequent and are often missed by the developers (remember Steve Valdez and his missing arms).



It is important to stress that this sort of thinking generally characterizes how programmers usually write code. They write the code to handle the most common cases, then add to the code to handle cases that deviate from the “garden path” or most common cases, but also in a rather ad hoc manner. One of the benefits Agile testers bring to the team is working with developers while doing test driven development or acceptance test driven development to help them avoid garden path thinking and the sorts of errors that arise from it.

In the diagram we see the usual sort of normal distribution of potential inputs for some component. Generally a programmer starts designing how the application should process cases from two places: the most common valid cases at the mean of the distribution and the extreme invalid cases that obviously should be rejected.

The problem is that many times the developer does not take into account two kinds of cases. The first are the outliers which are valid cases that occur so statistically infrequently that no one thinks of them (like Steve Valdez). The second are the invalid cases that are “almost” valid or might appear to be valid on first examination but in fact are invalid.

As an example of the second type, if a name input field is defined to have only ASCII data, then the name François Côté might appear to be valid if the programmer is thinking in terms of UTF-8 or more modern character encoding schemes, but it is not valid ASCII data.

## Selective Perception

Selective perception occurs when we respond to what we believe or expect to be there rather than what we actually perceive. This is the result of our brain “jumping to a perceptual conclusion” based on the identification of a few salient features.

For most of us, perception is reality. Yet, our perceptions do not always mirror reality, they are heavily influenced by our expectations and prior experiences. We observe stimulus with a preconceived notion of how it should appear and find it difficult replacing that notion with the facts. Instead of reality, we “selectively perceive what we expect and hope to see”. This biasing phenomenon is defined as selective perception.

The relevance for testing is that testers have been known to declare that tests pass, i.e. the expected outcomes match the actual outcomes, but later a review of the results shows that this was not the case. The testers did not see the failed test cases when they examined the results of the tests because they expected to see a “success” result.

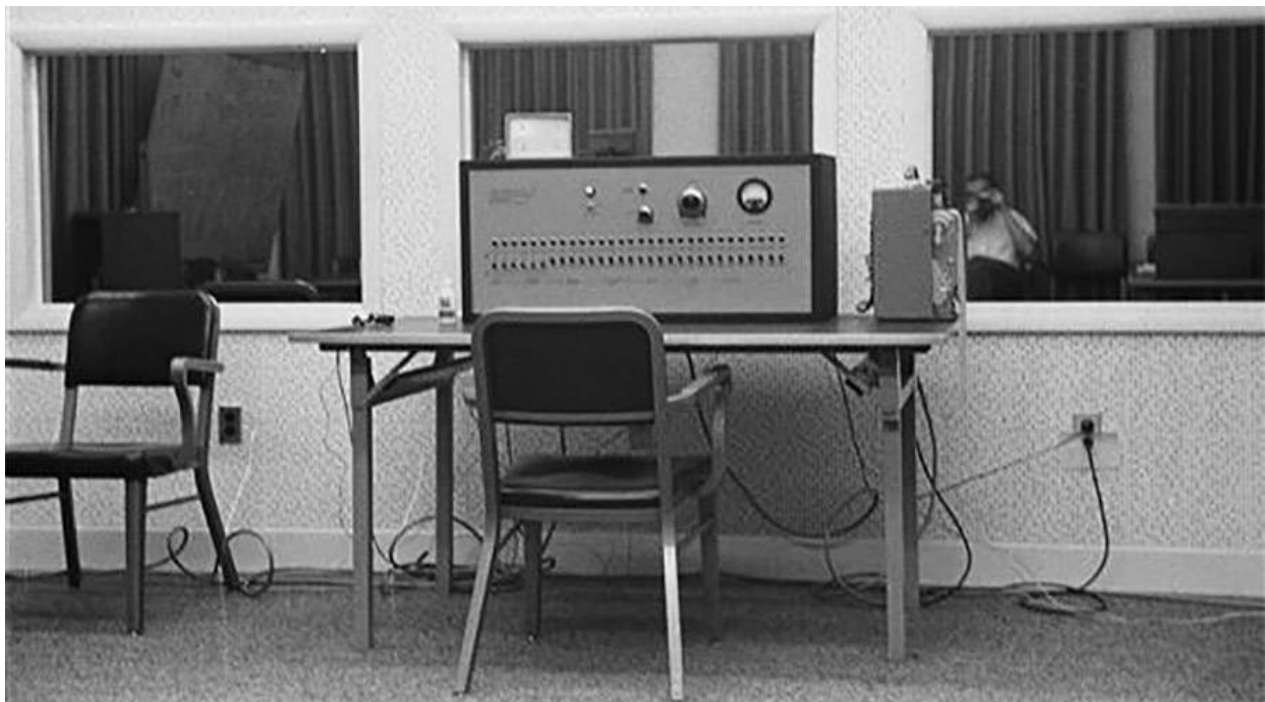
## Obedience to Authority

This is the first of two powerful social forces that affect testing outcomes that we will look at, but these are not the only ones. While we do not have to go into the details, the seminal work was done by Stanley Milgram in the 1960s who, in a very famous set of experi-

ments, showed that it was possible to have ordinary moral people perform horrendous acts simply because a person in authority told them to.



The relevance for testing is that the results of testing are not impartial if there is a power relationship between the tester and a person in authority, the project manager for example, who orders certain outcomes be achieved by the testing.



One of the pressures that software testers are under is testing while being ordered to make sure that a piece of software is proven to be compliant or safe or bug free, perhaps in order to make a release date. In this case the software testers are no longer actual testers, but are expected by the authority to rubber stamp their approval.

I personally have been face to face with senior management and told that they cannot understand why my testing team cannot just take the programmers word for it that there are no bugs in the software because there is a release date to be met.

For testers, this is a particular concern. We need to be part of the team, but we cannot let ourselves get into a power dynamic with other team members. Generally we can ameliorate this effect by working in collaborative relationships but still maintaining a separate reporting relationship. A good example of this might be Military Police in most armies. While they are integrated into the overall organizations, military police generally maintain a separate command structure to ensure that no power relationship might exist between a military police lieutenant and a colonel he might be investigating for a crime.

## Peer Conformance

We are all used to the concept of peer pressure and how people make choices to be accepted socially, however there is another more powerful kind of peer pressure that can an impact on testing outcomes. In a series of experiments, Solomon Asch demonstrated these powerful effects of peer pressure, or yielding to group pressures even when no specific request to do so is made. While not as dramatic as Milgram's study, a significant portion of his subjects displayed conforming behaviour as the result of subtle group pressures.

In his experiments, subjects were given a choice to report a fact that was clear and obvious (in the experiment it had to do with the lengths of lines) or to go along with the rest of the group who all agreed on an obviously wrong answer.





In the original study in 1958, 37 of the 50 subjects conformed to the majority at least once, and 14 of them conformed on more than 6 of the 12 trials. On average, when confronted with a unanimous wrong answer by the other group members, the subject agreed with the group 32% of the time.

What is the explanation for this conforming behaviour? After the experiment, most of them the subjects who conformed said that they knew the group answers were wrong and did not believe the conforming answers they gave, but they agreed with the group anyway. Their agreement was not based in wanting to be part of the group, instead they most commonly said that they could not believe that they were smarter than the rest of group collectively, and because the group had agreed, there must have been some factor that everyone in the group was aware of but the subject was not – they believed the group was better informed than they are.

In other words, they all believed that their own answer was correct, but when they saw everyone else agreeing on a different answer, they started to second guess themselves, often wondering if they had missed something or misunderstood something about the task that everyone else seemed to understand. Often they deferred to the group consensus even though they were unclear as to why their own answer was so different.

Imagine a junior software tester in a meeting with a number of senior programmers, testers and software designers. The unanimous opinion of the group is that the software is functioning to spec. There is pressure on the software tester to ignore the stack of negative test results in front of him and agree with the group. This would be an ideal situation for Asch's peer pressure.

Just a quick note that even though we are well aware that this sort of effect can take place in a traditional environment, I would suggest the opportunity for this effect is even greater in an Agile team.





## 2.8 “The Way of The Tester”

Software testing is not the only profession that has to worry about these sorts of human errors: teachers, doctors, therapists, researchers, investigators and a host of other professions have to deal with exactly the same effects. The reason is that these are because of both the way our brains are wired to process information and also because of the way we have learned to work and interact with others.

So how do we deal with it? There are a couple of commonly used ways to counterbalance these effects. Some of these are

### ***2.8.1 Formal Protocols***

Most of the errors we looked at are artifacts of how our brain processes information or how we work in social groups. We cannot just decide to not be involved in a self-fulfilling prophecy, for example, because the effect is either taking place below our threshold of awareness or we are incapable of acting in other ways.

We often use a formal and rigorous set of protocols so that everything is done in a standardized and uniform way by everyone. A good example is the double blind protocol for medical testing to avoid the experimenter and placebo effects. What a standardized protocol does is remove the “wobble room” we have in making decisions.

There are several reasons for doing this. The first is that if we look at a lot of the errors we make as testers, they occur because we make subjective decisions. We can control the error introduced by the subjective decisions by having an explicit and formal way of making decisions that removes the subjective component.

The second reason is that by formalizing and externalizing the decision making process, others can review and identify possible errors we made by checking out decision making processes.

Consider for example the problem of defining what underage drinking is. The intent is generally stated as ensuring that people who drink are mature enough to make good decisions about alcohol consumption. In the real world, there are some people who are mature enough to make good decisions about alcohol at fifteen and others who can't make good decisions at thirty. If we pass a law that says you have to be mature enough to drink, we have chaos.

Now the decision of whether someone is mature enough to drink is left up to a subjective judgment call by a police officer, and different officers will undoubtedly make different calls, which is generally goes against the idea that the law should be applied equally and fairly to all. What we do in this sort of case is remove the need for the officer to make a judgment by stating that a person must be at least eighteen – or twenty-one or whatever age is chosen in that jurisdiction – to be considered mature enough to drink. Now the process is objective and removes the potential for bias on the part of the individual officers.

It should be pointed out that this is not a perfect solution since not everyone is suddenly able to make mature decisions when they turn twenty-one, but what it does do is remove subjectivity and bias from a decision making process, the advantages of which tend to outweigh the disadvantages.

Several of the techniques we use as part of our formal protocols are:

1. Algorithmic techniques as opposed to intuitive techniques.
2. Engaging in reviews and evaluation of the testing process itself.
3. Following accepted principles of software testing. We will look at some of these in just a moment.

## ***2.8.2 Formal and Repeatable Processes***

All testing is planned and documented so that it is repeatable in the same way that a science experiment is documented so that it also is reproducible. We have standardized ways of working with the testing life cycle and processes in order to have reliable testing. When ad hoc or informal testing is being done, we often lose the quality or reliability.

We use formal and standardized processes for the following reasons.

1. It allows for repeatable tests and reliable results.
2. It allows for formal reviews of the testing process.
3. It allows us to use testing resources more effectively including the use of automated tools.
4. It is essential for testing process improvement.

It's not just enough to follow a rigorous process but we also need to be able to review and improve the process. There are three parts to this.

The first is to document everything. Keep in mind Demming's Principle: If you didn't write it down, it didn't happen. This is especially important in an Agile environment where there is constant churn of requirements and continuous change. Without accurate and up to date documentation, it is easy to descend into project chaos.

The second is identifying what the assumptions that are being made in the testing process and documenting them. We will always make assumptions but by identifying and documenting them, we can later go back and see if they may have influenced the results and need to be changed. We may find ourselves also having to document the assumptions of others, such as stake holders and developers.

Third, the testing profession is characterized by an adherence to a methodical and systematic process with as little reliance on subjective observation and recollection as possible. As a result, we try to rely on reviews and having many different eyes on a testing



activity. In most organizations, a test case is not ready to be used until it has been critically reviewed by other testers.

### ***2.8.3 Myers Testing Principles***

In his 1979 book *The Art of Software Testing*, Myers identified a number of principles that he stated characterized good software testing. These have stood the test of time and form the basis of a lot of our current software testing practices and are still valid for Agile testing. These are reproduced here for discussion.

#### **A necessary part of a test case is a definition of the expected output or result.**

Without clearly stating the expected result we run into the problems of self fulfilling prophecies and selective perception – or as some testers put it “the eye sees what it wants to see”. Without being able to compare expected output to actual output, it is easy to miss errors. In the worst possible case, violating this principle could mean a test run which uncovered errors but were then ignored by the tester.

#### **Programmers should avoid attempting to test their own code.**

This principle clearly helps testers avoid a number of the problems we have discussed, just as we cannot proofread our own writing, we cannot test our own code. In fact anyone writing tests for their own code is prone to a self-fulfilling prophecy – they are likely to write tests that will be biased to pass.

More importantly, if errors have been introduced by an incomplete or faulty understanding of system requirements, then those same errors would not be caught by test cases designed under the same misunderstandings.

#### **A programming organization should not test its own code.**

The influence of conformity and authority are clearly minimized by doing the testing outside the organization which developed the code. In addition, the same types of mistakes made by programmers testing their own code can also occur at an organizational level. Beta testing is one way to implement this principle.

### **Thoroughly inspect the result of each test.**

In many cases where code has failed, postmortems have revealed that earlier tests uncovered errors but that these were overlooked because nobody took the time to study the results. One of the best ways to ensure the results are thoroughly inspected is to use automated testing and reporting tools.

### **Test cases must be written for invalid and unexpected, as well as valid and expected input conditions.**

The natural tendency is to concentrate our testing on typical cases characterized valid and expected inputs. However, programs in production often show errors when they are used in some new or novel manner. When we use invalid or unexpected input conditions, the likelihood of finding errors increases.

### **Examining a program to see if it does not do what it should is only half the battle. The other half is seeing whether the program does what it is not supposed to do.**

Basically this means that a program has to be examined for unwanted side effects. A program which produces the correct results but corrupts the input file is doing something it is not supposed to do and thus still contains errors.

### **Avoid throw away test cases unless the program is truly a throw away program.**

With an undisciplined software testing approach, test cases are created on the fly. The tester sits at a terminal, generates test input and submits it to the program. The test data disappear when the test is complete. This sort of approach just doesn't work in an Agile environment. The test cases are now part of the set of development artifacts and need to be subject to the same level of configuration management and control as the source code.

Such throw away cases represent a lost resource. Test cases should be saved and recycled for several reasons. First, when the system is modified, the saved tests can be used to test for all the scenarios identified in the original test plan and thus avoids the temptation to test only the newest additions. Second, saved test cases allow for regression testing to ensure that new features have not introduced unwanted side effects.

### **Do not plan a testing effort under the tacit assumption that no error will be found.**

Testing should be viewed as a process which locates errors and not one that proves the program works correctly. To do otherwise opens us to a self fulfilling prophecy. By using a formal protocol, we can eliminate the sort of assumptions that would bias a test planning effort.



**The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.**

Meyers admits this result is counter intuitive but empirically verifiable. Basically this principle states that errors come in clusters. For example, in the IBM 370 operating system, 47% of errors found were located in 4% of the modules in the system. One possible explanation is that the rate of finding errors may be proportional to the of time that a section of code spends in execution compared to other sections. However as we shall see it is more likely that programmers tend to make the same mistakes over and over at specific points in the code.

**Testing is an extremely creative and intellectually challenging task.**

Historically, testing has been viewed as something of a lower level function in software organizations with the programmers occupying higher rungs on the corporate food chain. The influence of this view is often seen by the lack of resources that testers have to work with or the assumption that it is better in the long run to cut back on testing in order to give the programmers more time to code.

In once case, a company offered their testers a bounty of \$25 for each bug found. Much to their amazement, the level of discovered bugs sky rocketed. Or at least until they found that testers were paying programmers \$10 for each bug they deliberately placed into the code.

**Testing is the process of executing a program with the intent of finding errors.**

A good test case is one that has a high degree of probability of detecting an as-yet undiscovered error. A successful test case is one that detects an as-yet undiscovered error. This is one place where Myers is out of date. This is more a description of the sort of traditional

## 2.9 Risk Matrix

### A Critical Tool for Assessing Project Risk

written by: Sidharth Thakur. edited by: Ginny Edwards 6/13/2015

<http://www.brighthubpm.com/risk-management/88566-tool-for-assessing-project-risk/>

Confused about how to manage risks? Can't decide which to deal with first? Then the risk assessment matrix is the tool that you need to prioritize and develop an effective strategy. Find out more about this helpful tool.

Severity Frequency	Catastrophic	Critical	Marginal	Negligible
Frequent	Extreme	High	Serious	Moderate
Probable	High	High	Serious	Moderate
Occasional	High	Serious	Moderate	Low
Remote	Serious	Moderate	Moderate	Low
Improbable	Moderate	Moderate	Moderate	Low
Prevented	None	None	None	None

A risk assessment matrix is a project management tool that allows a single page – quick view of the probable risks evaluated in terms of the likelihood or probability of the risk and the severity of the consequences.

A risk assessment matrix is easier to make, since most of the information needed can be easily extracted from the risk assessment forms. It is made in the form of a simple table where the risks are grouped based on their likelihood and the extent of damages or the kind of consequences that the risks can result in. A sample risk assessment matrix can be downloaded for free from [here](#).



Making a risk management matrix is the second step in the process of risk management, and it follows the first step of filling up a risk assessment form to determine the potential risks. The preparation of risk assessment forms is a more elaborate task and involves determining risks, gathering risk data, determining the probability and the impact levels of the risks, understanding consequences, assigning priorities and developing risk prevention strategies. On the other hand, a risk assessment matrix just provides the project team with a quick view of the risks and the priority with which each of these risks needs to be handled.

Also in project planning, a different type of risk assessment template can be created in Excel and used to assess the overall risk of initiating a project.

### ***How to Place Risks in the Matrix***

As mentioned above, in a risk assessment matrix risks are placed on the matrix based on two criteria:

1. **Likelihood:** the probability of a risk
2. **Consequences:** the severity of the impact or the extent of damage caused by the risk.

### ***Likelihood of Occurrence***

Based on the likelihood of the occurrence of a risk the risks can be classified under one of the five categories:

1. **Definite:** A risk that is almost certain to show-up during project execution. If you're looking at percentages a risk that is more than 80% likely to cause problems will fall under this category.
2. **Likely:** Risks that have 60-80% chances of occurrence can be grouped as likely.
3. **Occasional:** Risks which have a near 50/50 probability of occurrence.
4. **Seldom:** Risks that have a low probability of occurrence but still can not be ruled out completely.
5. **Unlikely:** Rare and exceptional risks which have a less than 10% chance of occurrence.

### ***Consequences***

The consequences of a risk can again be ranked and classified into one of the five categories, based on how severe the damage can be.

1. **Insignificant:** Risks that will cause a near negligible amount of damage to the overall progress of the project.



2. **Marginal:** If a risk will result in some damage, but the extent of damage is not too significant and is not likely to make much of a difference to the overall progress of the project.
3. **Moderate:** Risks which do not impose a great threat, but yet a sizable damage can be classified as moderate.
4. **Critical:** Risks with significantly large consequences which can lead to a great amount of loss are classified as critical.
5. **Catastrophic:** These are the risks which can make the project completely unproductive and unfruitful, and must be a top priority during risk management.

### ***Using the Risk Assessment Matrix***

Once the risks have been placed in the matrix, in cells corresponding to the appropriate likelihood and consequences, it becomes visibly clear as to which risks must be handled at what priority. Each of the risks placed in the table will fall under one of the categories, for which different colors have been used in the sample risk assessment template provided with this article. Here are some details on each of the categories:

1. **Extreme Risk:** The risks that fall in the cells marked with 'E' (red color), are the risks that are most critical and that must be addressed on a high priority basis. The project team should gear up for immediate action, so as to eliminate the risk completely.
2. **High Risk:** Denoted with 'H' with a pink background in the risk assessment template, also call for immediate action or risk management strategies. Here in addition to thinking about eliminating the risk, substitution strategies may also work well. If these issues cannot be resolved immediately, strict timelines must be established to ensure that these issues get resolved before the create hurdles in the progress.
3. **Medium Risk:** If a risk falls in one of the orange cells marked as 'M' , it is best to take some reasonable steps and develop risk management strategies in time, even though there is no hurry to have such risks sorted out early. Such risks do not require extensive resources; rather they can be handled with smart thinking and logical planning.
4. **Low Risk:** The risks that fall in the green cells marked with 'L', can be ignored as they usually do not pose any significant problem. However still, if some reasonable steps can help in fighting these risks, such steps should be taken to improve overall performance of the project.





## Module Three

### *Rational Team Concert and Jazz Basics*

*It doesn't matter how many resources you have. If you don't know how to use them, they will never be enough.*  
Unknown

*I like the saying: "Failure isn't an option. It's mandatory."  
That's how I think about complex systems at large scales. The individual components are going to fail in lots of different ways and the challenge is to think about how to build systems that can deal with that failure gracefully and transparently*  
Ryan Day

*If you're doing it wrong, things break all the time and people see you fix them and think you're doing a great job. If you're doing it right, they think you don't do anything and you should be fired.*  
Anon Sysadmin



## 3.1 Introduction to RTC and Jazz

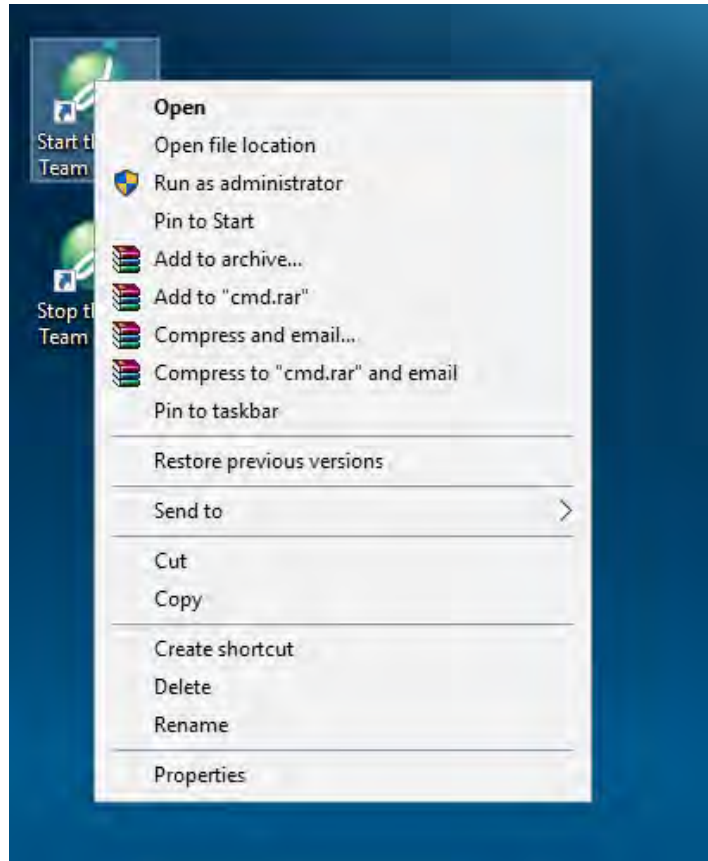
Rational Quality Manager is part of the Rational Team Concert (RTC) CLM suite and Jazz is the server that runs it all. While this is not a course in RTC administration or the details of the Jazz server, it is helpful to get familiar with how they operate and how to interact with the interface before we jump into RQM.

Because the RTC platform is complex and can be confusing if you are not used to it, this module also serves as an orientation into the RTC environment, features and some of the basic sorts of tasks that you do as a user of RTC.

You will also be acting as a project administrator for a RQM testing project. This may be something that you may never actually have to do, but having to work in these roles during class will give some insights into how the whole RTC platform works which should enable you to use RQM creatively and resolve the sorts of day to day issues that you might have using the product.

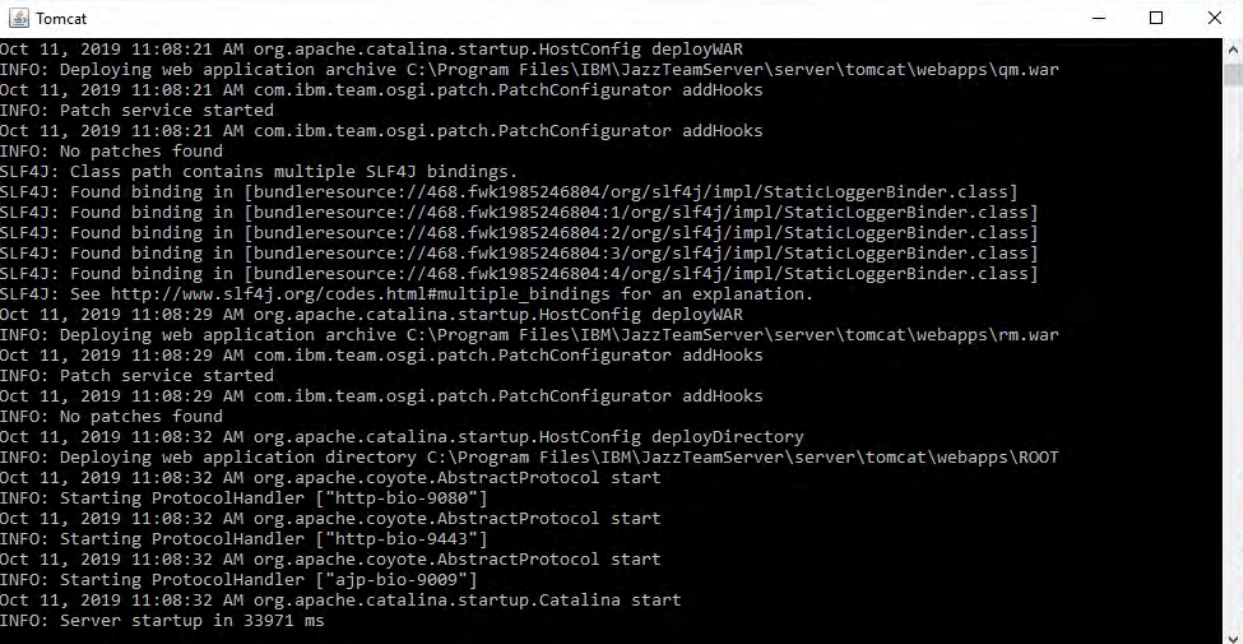
### 3.1.1 Starting up Jazz and RTC

RTC is a web based product which means we access it via a web browser. However the first thing we have to do before we can use RTC is start up the Jazz server. You will be doing this in the labs as well but in the manual we will walk through the process.



It is strongly suggested that you start up the server using the process described in Lab 3-1 so that you can follow along with what the instructor is doing. Or you can follow along with the instructor as he starts up the Jazz Server.

Once the Server has finished booting up and is ready – which may take some time – we can log into the product using a web browser.

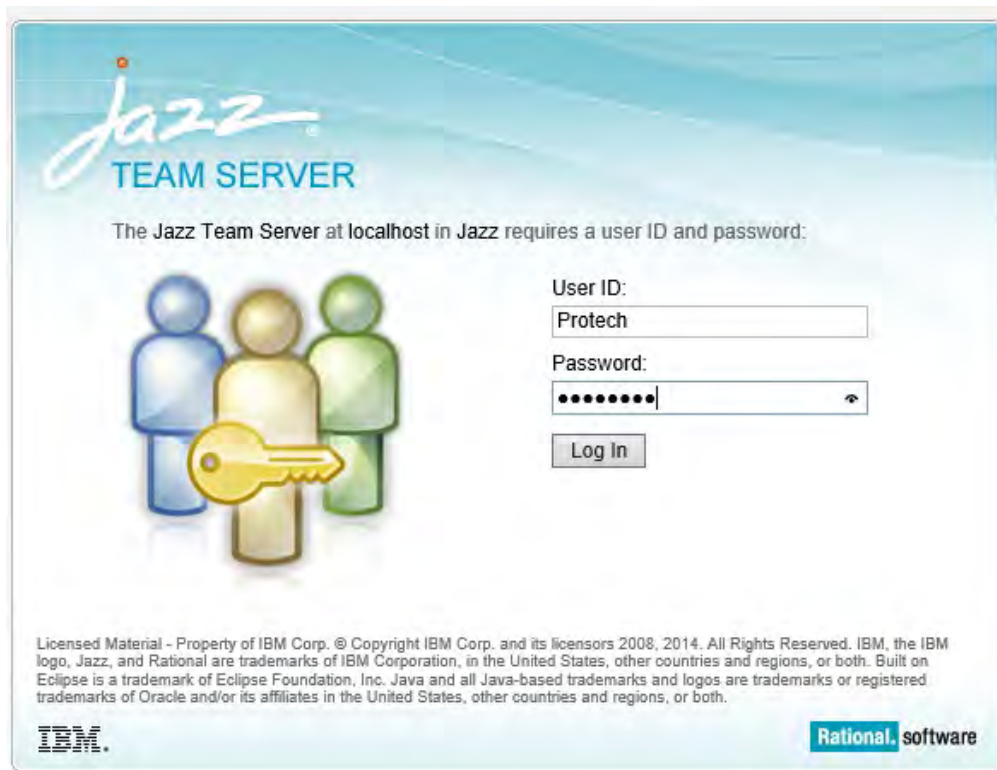


```
Tomcat
Oct 11, 2019 11:08:21 AM org.apache.catalina.startup.HostConfig deployWAR
INFO: Deploying web application archive C:\Program Files\IBM\JazzTeamServer\server\tomcat\webapps\qm.war
Oct 11, 2019 11:08:21 AM com.ibm.team.osgi.patch.PatchConfigurator addHooks
INFO: Patch service started
Oct 11, 2019 11:08:21 AM com.ibm.team.osgi.patch.PatchConfigurator addHooks
INFO: No patches found
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [bundleresource://468.fwk1985246804/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [bundleresource://468.fwk1985246804:1/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [bundleresource://468.fwk1985246804:2/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [bundleresource://468.fwk1985246804:3/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [bundleresource://468.fwk1985246804:4/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
Oct 11, 2019 11:08:29 AM org.apache.catalina.startup.HostConfig deployWAR
INFO: Deploying web application archive C:\Program Files\IBM\JazzTeamServer\server\tomcat\webapps\rm.war
Oct 11, 2019 11:08:29 AM com.ibm.team.osgi.patch.PatchConfigurator addHooks
INFO: Patch service started
Oct 11, 2019 11:08:29 AM com.ibm.team.osgi.patch.PatchConfigurator addHooks
INFO: No patches found
Oct 11, 2019 11:08:32 AM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory C:\Program Files\IBM\JazzTeamServer\server\tomcat\webapps\ROOT
Oct 11, 2019 11:08:32 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-9080"]
Oct 11, 2019 11:08:32 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-9443"]
Oct 11, 2019 11:08:32 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-9009"]
Oct 11, 2019 11:08:32 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 33971 ms
```

**Do NOT close the window command window where the server was started or you will shut down the server!**

For this part of the module, you are going to log in as the Administrator using the following credentials that have been set up for you on the RTC on your virtual machine.





id: Protech      password: Pa\$\$w0rd

Use the URL

`https://localhost:9443/jts/admin`

When you log in, you will be at the administrative area for RTC. We won't be dealing with any RTC administration topics in this course except for some basic functions we need to work with RQM.

There is a sample project loaded called "JKE Banking" of "Money that Matters" which comes as an IBM supplied example of an RTC project. This project is there for you to explore so you can see some of the artifacts and functionality of RQM and the other RTC modules in a mid-project sort of state.

Rational Jazz Team Server (localhost:9443/jts)

One of the Client Access Licenses expires in 56 days

Server Administration

Home Server Users Reports

Protect Search Users

## Welcome to the Jazz Team Server Administration Home

As a **Jazz Administrator**, you have access to the following tasks:

### Jazz Team Server Administration

#### Manage Server

Configure the Jazz Team Server settings, manage license keys, and check server status information.

[Manage Server](#)

[About administering the Jazz Team Server](#)

Server Status	
Database Status	Connected
Public URI	<a href="https://localhost:9443/jts">https://localhost:9443/jts</a>
Diagnostics	1 warning

#### Manage Users (19)

The Jazz Team Server provides a single synchronized user database that is shared by the server and all registered applications. You can create and manage user accounts and user license assignments directly in the centralized Jazz Team Server administration or the applications registered with the server.

[Create Users](#)

### Task Guide

The Task Guide is your quick reference for recommended next steps and related tasks. For more information on each task, see the content sections within this page.

#### JAZZ TEAM SERVER ADMINISTRATION

- [Setup Server](#)
- [Create Users](#)
- [Assign Client Access Licenses](#)
- [Create Lifecycle Projects](#)

#### EXPLORE RELATED TOPICS

- [Try out the Money That Matters sample in Lifecycle Projects](#)
- [About adding members to projects](#)
- [About adding and modifying users as members of project areas](#)





## 3.2 Users and Roles

The "C" in CLM stands for collaborative which means that by implementing CLM in RTC that RTC must be a multi-user application where users can share work and communicate with each other on tasks.

Users are managed by the Jazz server on behalf of all the RTC components like RQM. There are a number of different ways user information can be administered depending on the whatever larger corporate environment is Jazz is running in, but a discussion of this is not part of this course. Instead, we will just use the default Jazz internal user management system.

If we navigate as administrator to a display of the users, we see something like this:

**Active Users**

You can manage user accounts and licenses on the Users pages in applications or the Jazz Team Server. However, to edit application-specific user settings, use the Users page in the application. Common user information is synchronized between applications and the server.

Use this page to create, import, and edit user accounts and assign client access licenses to users. To create a new account, click the "Create User" action above. To edit an account and assign licenses, click on a user name in the list below. To search for a particular user, enter a portion of the name, user ID, or email address in the filter-box below.

Previous | 1 - 19 of 19 | Next

Type Filter Text

Name	User ID	E-Mail	Actions
AI	ai	ai@jkebanking.net	
Bob	bob	bob@jkebanking.net	
Build	build	build@jkebanking.net	
Curtis	curtis	curtis@jkebanking.net	
Data Collection User	etl_user	etl_user	
Dave	dave	dave@jkebanking.net	
Deb	deb	deb@jkebanking.net	
Marco	marco	marco@jkebanking.net	
Protech	Protech	protech@protech.com	
Rebecca	rebecca	rebecca@jkebanking.net	
Sally	sally	sally@jkebanking.net	
Tammy	tammy	tammy@jkebanking.net	
Tanuj	tanuj	tanuj@jkebanking.net	
Ursula	ursula	ursula@jkebanking.net	
ccm_user	ccm_user	unknown	
jts_user	jts_user	unknown	
lpa_user	lpa_user	unknown	

There are a number of users that have already been created as part of the "Money Matters" sample project. These users are easy to spot since they all have emails in the domain "jkebanking.net".

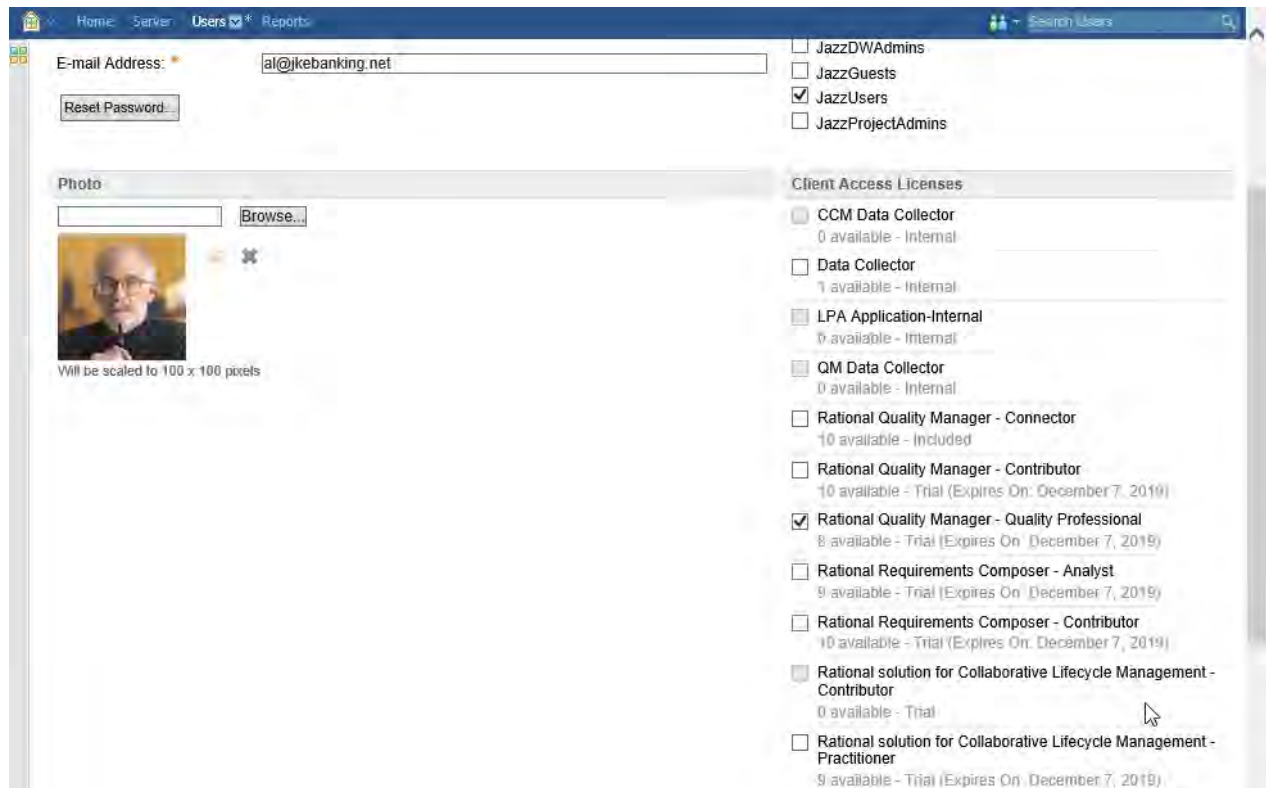
### 3.2.1 System Users

System users are special accounts that are created by the RTC component applications or the Jazz platform to run various background administrative tasks. For example, the dcc\_user is the data collector which does various background data updating tasks and

the `qm_user` is created by the Quality Manager component to perform various background tasks in that component. The system user for the Requirements Composer (or DOORS) is `rm_user`. The system users all are identified by their id ending with `"_user"` and since they are background processes, it is impossible for anyone to actually log in as one of the system users.

### 3.2.2 JAZZ Security Roles

In the screen shot following, we are looking at the display of a user called `AI` who was created during the installation of the "Money That Matters" sample project. `AI` has the usual sort of user information. Every user in RTC has a role that is defined by a combination of their security roles and their product licences. For example, `AI` is a `JazzUser` but has no licences currently assigned to him.



Jazz uses security roles to determine whether a specific user is allowed to read, write, or administer various resources. These roles apply across all of the RTC applications but each application can further refine access to their specific artifacts by the use of application specific licences.





Jazz applications use the following predefined security roles:

1. **JazzAdmins:** Administrators of a Jazz server with full read-write access to everything. A user with this role has complete read and write access to the repository, but is primarily responsible for creating users, administering the server, administering the data warehouse, and managing licenses.
2. **JazzProjectAdmins:** A user with this role can create and modify artifacts in each of the CLM applications, but is primarily responsible for creating and modifying project areas and process templates.
3. **JazzUsers:** A user with regular read-write access to the Jazz server repository. This would normally be required for a user to be able to use a license that requires allows them to modify DOORS or other RTC artifacts. However, the licenses they have or don't have restrict which project artifacts they have read-write access to. This security role is generally assigned to project team members who will be working on the project.
4. **JazzGuests:** Users with read-only access to the Jazz application. This role is normally associated with stakeholders and other parties who many need to review artifacts for feedback or audit purposes, but who are not part of the team and should not have any need to modify or change anything.

Notice that it possible to have any combination of roles however the best practice is that each user should have the lowest level of access they need to do their job. In addition, it is also considered a best practice to have only one or two users with JazzAdmin roles.

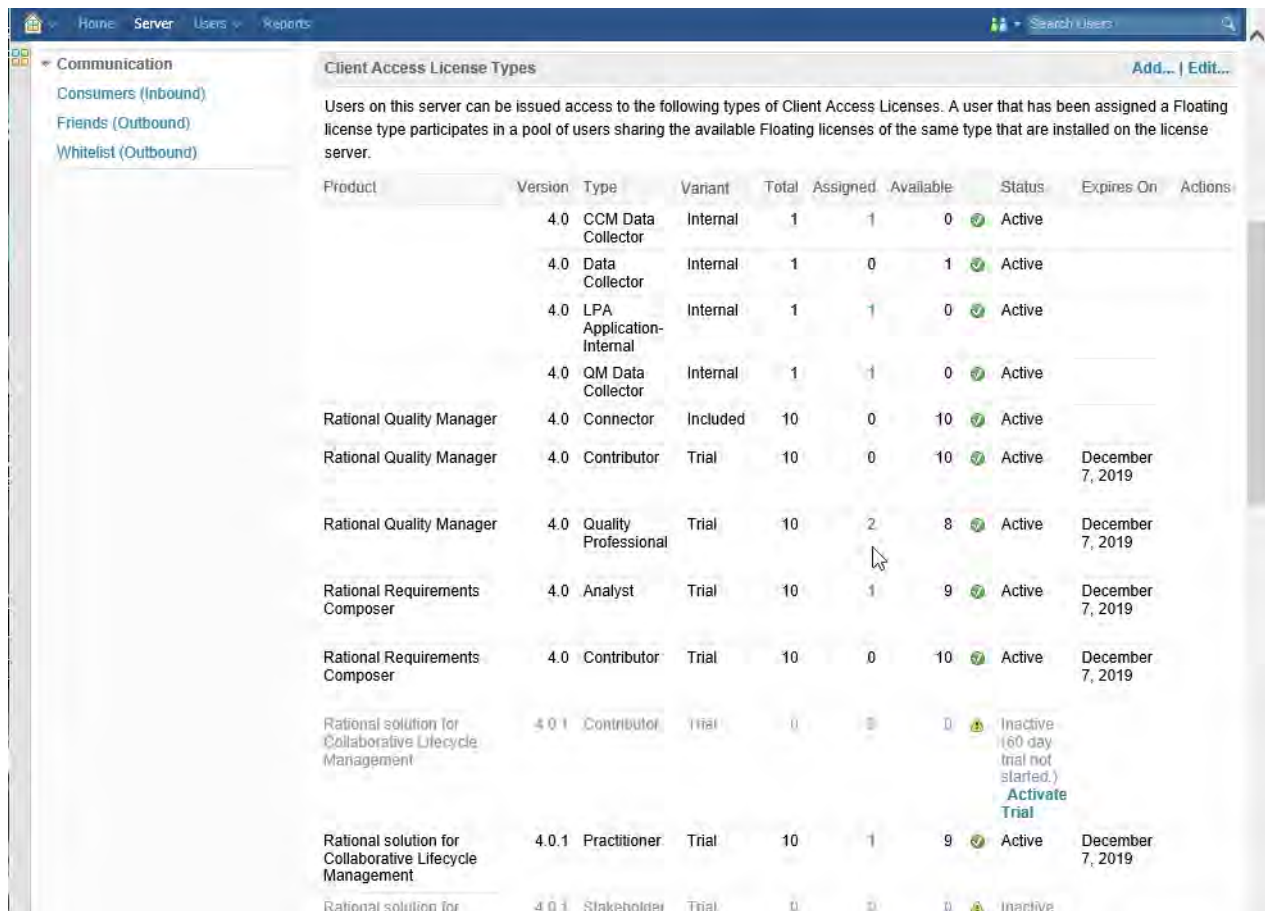
### 3.2.3 CLM Licences

Each user may be assigned one or more licences for each RTC application. The version we are using has a set of trial licences that we will use to assign licences to our team members. To see the licences that are available, go the to server administration area (the "server" option on the menu at the top left) and select the license key management option.

The available licences for the products are shown. Some have not been activated for this class, but you can activate them if you want. Some of the licences do not have an expiration – these are "single use" licenses which are assigned to specific system users like rm\_user to allow them to do their background work.

The two licences that are of interest to us are listed below with the RTC description of what ten enable.:

1. **Quality Professional:** This Quality Professional license is designed for quality professionals actively participating in the project. A user that has been assigned a Quality Professional Client Access License has full read and write access to Change Management, Report Customization, Planning, Test management and Requirements management. This license also provides read access to Software Configuration Management, Automation (Build System) and Design Management capabilities unless otherwise restricted by role-based process permissions
2. **Quality Contributor:** This Contributor license is designed for professional, non-development, team-members actively participating in the project. A user that has been assigned a Contributor Client Access License has full read and write access to Change Management, Report Customization, and Planning. This license also provides read access to Software Configuration Management, Automation (Build System), Requirements Management, Test Management and Design Management capabilities unless otherwise restricted by role-based process permissions.



Product	Version	Type	Variant	Total	Assigned	Available	Status	Expires On	Actions
	4.0	CCM Data Collector	Internal	1	1	0	Active		
	4.0	Data Collector	Internal	1	0	1	Active		
	4.0	LPA Application-Internal	Internal	1	1	0	Active		
	4.0	QM Data Collector	Internal	1	1	0	Active		
Rational Quality Manager	4.0	Connector	Included	10	0	10	Active		
Rational Quality Manager	4.0	Contributor	Trial	10	0	10	Active	December 7, 2019	
Rational Quality Manager	4.0	Quality Professional	Trial	10	2	8	Active	December 7, 2019	
Rational Requirements Composer	4.0	Analyst	Trial	10	1	9	Active	December 7, 2019	
Rational Requirements Composer	4.0	Contributor	Trial	10	0	10	Active	December 7, 2019	
Rational solution for Collaborative Lifecycle Management	4.0.1	Contributor	Trial	0	0	0	Inactive (60 day trial not started.)		Activate Trial
Rational solution for Collaborative Lifecycle Management	4.0.1	Practitioner	Trial	10	1	9	Active	December 7, 2019	
Rational solution for	4.0.1	Stakeholder	Trial	0	0	0	Inactive		

The role of each user is now determined by their repository security role and the licences that have been assigned to them by an administrator. The licenses provide a more fine grained control over what users may and may not do. For example, if we create a user "Sharma" as a normal JazzUser then based on the security role, Sharma could potentially modify any artifact in any application. For security reasons, the JazzUser security role is really what that a user could potentially do, not actually do.



For example, unless Sharma is assigned an RQM licence, then Sharma is unable to perform any actions in the RQM application. If Sharma is assigned a Quality Professional license, they can work in RQM, they are still unable to work on artifacts in the Requirements Composer application.

*Because there are so many levels of security, access problems can often occur but can usually be traced back to either a missing permission or a conflict in the permissions.*

### 3.3 Creating a User

In this section we create a user named "frodo" who will experiment with. From the administration screen as Protech, who is a JazzAdmin, we use the create user option to get to this screen.

We will leave the default security role as JazzUser and we will give him a Quality Professional licence that will enable him to use the RQM application.

One of the Client Access Licenses expires in 56 days

Server Administration

Protech

Home Server Users Reports

New users will be created with a default password equal to their User ID in your directory service.

Frodo Baggins \*

Save

#### Overview

##### Details

User ID (case sensitive): \* frodo

E-mail Address: \* frodo@shire.net

##### Repository Permissions

- ☐ JazzAdmins
- ☐ JazzDWAdmins
- ☐ JazzGuests
- ☒ JazzUsers
- ☐ JazzProjectAdmins

##### Photo

Browse...

Will be scaled to 100 x 100 pixels

##### Client Access Licenses

- ☐ CCM Data Collector  
0 available - Internal
- ☐ Data Collector  
1 available - Internal
- ☐ LPA Application-Internal  
0 available - Internal
- ☐ QM Data Collector  
0 available - Internal
- ☐ Rational Quality Manager - Connector  
10 available - Included
- ☐ Rational Quality Manager - Contributor  
10 available - Trial (Expires On: December 7, 2019)
- ☒ Rational Quality Manager - Quality Professional  
7 available - Trial (Expires On: December 7, 2019)
- ☐ Rational Requirements Composer - Product

By default, frodo is a JazzUser and we have assigned him a Quality Professional licence. We don't enter a password for the user, users are created with a default password that is the same as their userid. Once the user has been created, we would normally have the user reset the password via email, but since we do not have a email system set up on the virtual machines, we will just use the default password.



If we navigate as administrator to a display of the users, we see something like this:

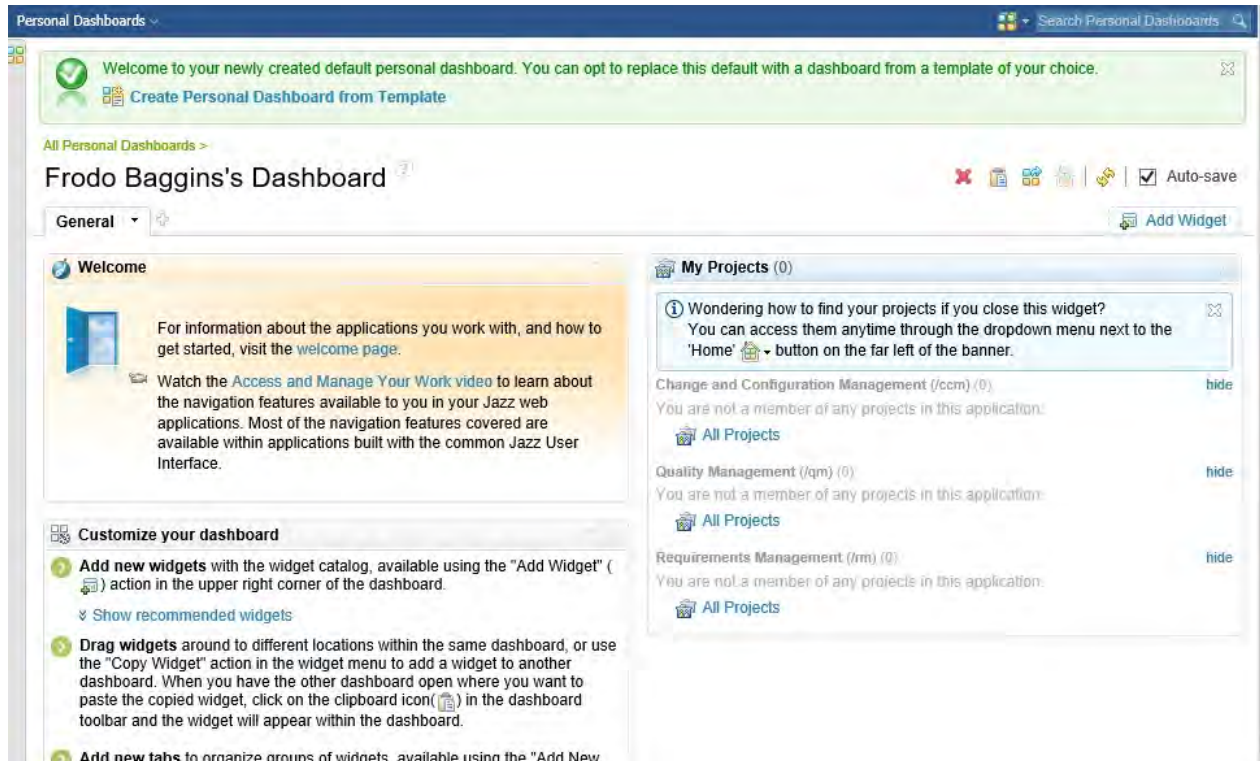
Name	User ID	E-Mail
Al	al	al@jkebanking.net
Bob	bob	bob@jkebanking.net
Build	build	build@jkebanking.net
Curtis	curtis	curtis@jkebanking.net
Dave	dave	dave@jkebanking.net
Deb	deb	deb@jkebanking.net
Frodo Baggins	frodo	frodo@shire.com
Marc	marc	marc@jkebanking.net

We also cannot delete users for a variety of reasons, however one of the most important reasons is that project meta-data is automatically created and associated with the various artifacts that provides audit trails of who did what to which artifact and when they did it. If we could delete a user, we could lose valuable meta-data. Instead we archive users so they are no longer active but their information still exists in the Jazz system.

### 3.3.1 Logging Into a User Area

Now that the frodo user exists, we can log in as frodo. Using the URL

`https://localhost:9443/jts`



When we do login as frodo, we are taken to his personal dashboard, which is like his personal work area. However, frodo is not part of any projects so there is nothing for him to do. For example, he can go to the RQM application by using the URL below. The "qm" is used for the RQM application to stand for "Quality Management".

`https://localhost:9443/jts/qm`

This brings up frodo's work area in RQM which shows the Money that Matters project. Frodo can see the project because the project has had its permissions set so that everyone can read it. However, Frodo is not a member of the project team for does he have permissions to work on the project.





Quality Management (/qm) One of the Client Access Licenses expires in 56 days

JKE Banking (Quality Management) Frodo Baggins

Project Dashboards Requirements Planning Construction Lab Management Builds Execution Reports Search QM Resources

All JKE Banking (Quality Management) Dashboards >

### JKE Banking (Quality Management)

General Planning Execution Development Defects

**Money that Matters Sample**

This sample provides an in-flight perspective of the collaborations that occur on a typical Agile team aligning requirements, development and quality. Walk through an iteration with the team as they deliver a new feature to support an important corporate initiative called Money that Matters. For getting started see the [Money that Matters sample scenario](#).

**My Tasks**

No tasks found.

**Quality Manager Event Log**

- [21] Membership changed: JKE Banking (Quality Management) 3 days ago
- [3] Review TestPlan: User Acceptance (7) 3 days ago
- [3] Review TestPlan: System Verification (6) 3 days ago
- [5] Review TestPlan: Master Test Strategy (5) 3 days ago
- [5] Review TestSuite: SVT Test - Money that Matters (4) 3 days ago
- [5] Review TestCase: (QM Tutorial) Allocate Dividends to Multiple Causes (3) 3 days ago
- [5] Review TestCase: Allocate Dividends to a Single Cause (2) 3 days ago
- [3] Review TestCase: Donor dividend allocation conforms to stated criteria (1) 3 days ago
- [3] Membership changed: JKE Testing - Money that Matters 3 days ago
- [3] Membership changed: JKE Testing 3 days ago

**PROJECT DETAILS**

**Timelines**

Development

- First Project Phase Ends Nov 7, 2019
- Release 1 Testing Ends Nov 13, 2019
- Release 1 Development Ends Nov 3, 2019

[Show More](#)

[View All Timelines >](#)

**Members (11)**

Showing 11 of 11 members.

See the [Project Area](#) to view all members.

Related Sites

- IBM Rational
- IBM Rational Quality Mgmt

If frodo tries to create a project to work on, RTC rejects the attempt because frodo is not a JazzProjectAdmin. Before frodo can do any work, he has to be added to an existing project by a JazzAdmin or have his security role changed so that he is a JazzAdmin and create his own project.

Quality Management (/qm) One of the Client Access Licenses expires in 56 days

Application Administration - Quality Management (/qm) Frodo Baggins

Project Areas Templates Access Groups

Active Project Areas >

**Project area cannot be saved. show details**

**frodos project** \*

**Save**

**Details** [Preview](#)

Summary:

Description:

**Process**





# Test Management with Rational Quality Manager 6.0

## Module Four

---

### *Test Plans*

*The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair.*

Douglas Adams

*Testing is the process of comparing the invisible to the ambiguous, so as to avoid the unthinkable happening to the anonymous.*

James Bach

*The real complexity in our jobs is that all planning is done under conditions of uncertainty and ignorance. The code isn't the only thing that changes. Schedules slip. New milestones are added for new features. Features are cut from the release. During development, everyone - marketers, developers and testers - comes to understand better what the product is really for.*

Brian Marick



## 4.1 Introduction

The simplest way to think of a test plan is that testing is a project, and the test plan is the corresponding project plan. There are a number of topics about test planning – developing time lines and effort estimation for example – that are way out of scope for what we want to cover in this course. The purpose of this module is just to review some of the basics of test plans so that we can see how to use RQM to automate our test planning.

There are a number of reasons to prepare test plans, just three will be mentioned here, but the same reasons for preparing a project plan at the start of any project also apply for test plans.

First, the process of preparing a test plan forces us to think through the testing process. This allows us to identify items that we don't understand, assess the difficulty of the testing tasks, identifies challenges and potential pitfalls in advance so that we can develop contingencies or alternatives.

Just as writing tests provides us with a number of insights into the code to be written, the writing of a test plan gives us insights into the testing to be done. It is not unusual to discover problematic issues that might come up during testing that were not apparent on a first inspection but were uncovered during the process of developing a test plan.

Second, the test plan is a living document. It is used as the basis for making decisions and communicating those decisions to team members. It also documents the decisions made and compromises reached and the reasons behind them. This provides not just a working document but also a historical record so that in the future we can evaluate the testing effort and do any forensic sort of analysis that needs to be done.

Third, the test plan helps us manage complexity and change. A testing project could be quite complex – consider the testing for the on-board computer systems for the space shuttle – and if we are in an Agile environment, we may find the project goals and design evolving because of the Agile process thriving on the “churn of changing requirements.” Having a documented plan helps us organize our testing when it becomes very complex as well as providing us with an efficient way to implement changes instead of doing things on an ad hoc manner on the fly.

In the following sections, we will look at the basics of the IEEE 829 test plan and then look at the facilities for using test plans in RQM.

## 4.2 IEEE 829 Master Test Plan (MTP)

One of the important points to keep in mind is that these standards are used by test organizations that often deal with software where the failure to test correctly may have catastrophic results – for example the software for a jumbo jet autopilot, or software that processes financial transactions at a government agency or software that controls a self driving car. Because of this, the standard is very detailed and comprehensive, but does that mean that your test plans need to follow the standard to the letter?

Generally the answer is no. The IEEE suggests that the standard should not necessarily be used “out of the box” because, just looking through this section, you can see that in a small testing project, a detailed IEEE 829 complaint test plan is overkill. Instead, it is suggested that the standard is often better used as a starting point for your own standard, adopting the portions that are appropriate for your testing organization, or be used to fill in the gaps of any existing standards you may have developed.

RQM does not implement this standard slavishly – it does allow you to implement a lot of what is in the standard, but it also adds in the CLM sorts of linkages from the RQM module to the Requirements Composer and other modules as well.

The rest of this section is summarized from the IEEE 829-2008 Standard as a short reference. In a few places, the text of the standard has been rewritten to make it easier to follow.

The MTP describes the overall test planning and test management for multiple levels of testing for either one project or across multiple projects. The MTP takes into account the software requirements and the quality management decisions for the project in order to meet the following objectives:

1. Defining the different testing activities (parts) for the project.
2. Setting the quality control criteria and testing objectives for each activity.
3. Allotting the division of labor (time, resources) to each activity.
4. Defining the interrelationships between the activities.
5. Identify the risks, assumptions, and standards of workmanship required for each activity.
6. Defining how to control each activity.
7. Confirm that the activity quality control objectives are consistent with the overall quality planning.
8. Describes the integrity level schema and the integrity level selected.
9. Defines the number of levels of test, such as system testing, security testing, acceptance testing, milestone testing, configuration testing and so on.



10. Defines the overall tasks to be performed, and the documentation and reporting requirements.

## ***4.2.1 Master Test Plan Outline***

### ***1. Introduction***

- 1.1. Document identifier
- 1.2. Scope
- 1.3. References
- 1.4. System overview and key features
- 1.5. Test overview
  - 1.5.1 Organization
  - 1.5.2 Master test schedule
  - 1.5.3 Integrity level schema
  - 1.5.4 Resources summary
  - 1.5.5 Responsibilities
  - 1.5.6 Tools, techniques, methods, and metrics

### ***2. Details of the Master Test Plan***

- 2.1. Test processes including definition of test levels
  - 2.1.1 Process: Management
    - 2.1.1.1 Activity: Management of test effort
  - 2.1.2 Process: Acquisition
    - 2.1.2.1: Activity: Acquisition support test
  - 2.1.3 Process: Supply
    - 2.1.3.1 Activity: Planning test
  - 2.1.4 Process: Development
    - 2.1.4.1 Activity: Concept
    - 2.1.4.2 Activity: Requirements
    - 2.1.4.3 Activity: Design
    - 2.1.4.4 Activity: Implementation
    - 2.1.4.5 Activity: Test

2.1.4.6 Activity: Installation/checkout

2.1.5 Process: Operation

2.1.5.1 Activity: Operational test

2.1.6 Process: Maintenance

2.1.6.1 Activity: Maintenance test

2.2. Test documentation requirements

2.3. Test administration requirements

2.3.1 Anomaly resolution and reporting

2.4.2 Task iteration policy

2.3.3 Deviation policy

2.3.4 Control procedures

2.3.5 Standards, practices, and conventions

2.4. Test reporting requirements

### **3. General**

3.1. Glossary

4.2. Document change procedures and history

## **4.2.2 Detailed Descriptions**

### **1. Introduction**

Describes the document and its place in context of the project-specific lifecycle. This is essentially an executive summary or abstract section that provides an overview of the entire test effort including the test organization, the test schedule, and the integrity schema. A summary of required resources, responsibilities, and tools and techniques may also be included in this section.

#### **1.1. Document identifier**

Uniquely identifies a version of the document by including information such as the date of issue, the issuing organization, the author(s), the approval signatures (possibly electronic), and the status/version (e.g., draft, reviewed, corrected, or final). Identifying information may also include the reviewers and pertinent managers.



## 1.2. Scope

This section describes the purpose, goals, and scope of the test plan including a description of any customization to the test planning process. The project(s) for which the plan is being written and the specific processes and products covered by the test effort.

The scope also describes the inclusions, exclusions, and assumptions/limitations that clearly define the limits of the test effort described in the plan. This is most efficiently done by specifying what is being included and equally important, what is being excluded from the test effort.

For example, only the current new version of a product might be included and prior versions might be excluded from a specific test effort.

The scope also identifies any grey areas for the test effort (assumptions and/or limitations) where management discretion or technical assumptions are being used to direct or influence the test effort. For example, system sub-components purchased from other suppliers might be assumed to have been tested by their originators so that they would not be re-tested individually but only tested in some integration test effort.

The testing effort described in the scope should reflect the overall test approach and the development methodology. For example, if a “waterfall” methodology is being used then each level of the test will be executed only one time. However, an iterative or Agile methodology is being used, then there will be multiple iterations of each level of test.

The test approach identifies what will be tested and in what the test levels will be done. The test approach identifies the rationale for testing or not testing, and it identifies the rationale for the selected order of testing. The test approach describes the relationship to the development methodology. The test approach may identify the types of testing done at the different levels. For example, “thread testing” may be executed at a system level, whereas “requirements testing” may take place at the component integration as well as at a systems integration level.

## 1.3. References

List all of the applicable reference documents. The references are separated into “external” references that are imposed external to the project and “internal” references that are imposed from within to the project. This may also be at the end of the document.

External references are all of the relevant policies or laws that give rise to the need for this plan, e.g.:

- a) Laws
- b) Government regulations
- c) Standards (e.g., governmental and/or consensus)
- d) Policies



Internal references refer to documents such as other plans or task descriptions that supplement this plan, e.g.:

- a) Project authorization
- b) Project plan (or project management plan)
- c) Quality assurance plan
- d) Configuration management plan

### **1.4. System overview and key features**

This section describes the mission or business purpose of the system or software product under test or provides a reference to where this information can be accessed, for example in a system definition document, such as a Concept of Operations). It also describes the key features of the system or software under test or reference where that information can be found.

### **1.5. Test overview**

This section describes the test organization, test schedule, integrity level scheme, test resources, responsibilities, tools, techniques, and methods necessary to perform the testing.

#### **1.5.1 Organization**

Describes the relationship of the test processes to other processes such as development, project management, quality assurance, and configuration management. It includes the lines of communication within the testing organization(s), the authority for resolving issues raised by the testing tasks, and the authority for approving test products and processes. This may include (but should not be limited to) a visual representation, e.g., an organization chart.

#### **1.5.2 Master test schedule**

Describes the test activities within the project life cycle and milestones and summarizes the overall schedule of the testing tasks, identifying where task results feed back to the development, organizational, and supporting processes (e.g., quality assurance and configuration management). It also describes the task iteration policy for the re-execution of test tasks and any dependencies.

#### **1.5.3 Integrity level schema**

Describes the identified integrity level scheme for the software-based system or software product, and the mapping of the selected scheme to the integrity level scheme used in the IEEE standard. The MTP documents the assignment of integrity levels to individual components (e.g., requirements, functions, software modules, subsystems, non-functional characteristics, or other partitions), where there are differing integrity levels assigned within the system. At the beginning of each process, the assignment of integrity levels is reassessed with respect to changes that may need to be made in the integrity levels as a result of architecture selection, design choices, code construction, or other development activities.



#### 1.5.4 Resources summary

Summarizes the test resources, including staffing, facilities, tools, and special procedural requirements (e.g., security, access rights, and documentation control).

#### 1.5.5 Responsibilities

Provides an overview of the test organization and responsibilities for testing tasks and identifies organizational roles including test leads and other testers and their responsibilities.

#### 1.5.6 Tools, techniques, methods, and metrics

Describes the documents, hardware and software, test tools, techniques, methods, and test environment to be used in the test process, and the techniques that will be used to identify and capture reusable testware. Included is information regarding acquisition, training, support, and qualification for each tool, technology, and method.

This section also documents the metrics to be used by the test effort, and describe how these metrics support the test objectives. Metrics that are appropriate a specific level of testing such as component, component integration, system, and acceptance testing may be included in the specific level test plan documents

## 2. Details of the Master Test Plan

This section describes the test processes, test documentation requirements, and test reporting requirements for the entire test effort.

### 2.1. Test processes including definition of test levels

This section identifies test activities and tasks to be performed for each of the test processes used in the the testing effort and documents those test activities and tasks. This section provides an overview of the test activities and tasks for all development life cycle processes including identifying the number and sequence of levels of test. There may be a different number of levels than the example used in this standard (component, component integration, system, and acceptance). For example, integration is often accomplished through a series of test levels, for both component integration and systems integration. Examples of possible additional test levels include security, usability, performance, stress, recovery, and regression. Small systems may have fewer levels of test, for example by combining system and acceptance. If the test processes are already defined by an organization's standards, a reference to those standards would suffice here.

#### 2.1.1 Process: Management

This section describes how all requirements of the standard are satisfied if the life cycle used in the MTP differs from the life cycle model in the standard. Testing requires advance planning that spans several development activities.

This section addresses the following eight topics for each test activity:

- a) Test tasks: Identify the test tasks to be performed with minimum test tasks, task criteria, and required inputs and outputs. Optional test tasks may also be performed to augment the test effort to satisfy project needs. The degree of intensity and rigor in performing and documenting the task should be commensurate with the integrity level. As the integrity level increases or decreases, so do the required scope, intensity, and degree of rigor associated with the test task.
- b) Methods: Describe the methods and procedures for each test task, including tools. Define the criteria for evaluating the test task results.
- c) Inputs: Identify the required inputs for the test task. Specify the source of each input. For any test activity and task, any of the inputs or outputs of the preceding activities and tasks may be used.
- d) Outputs: Identify the required outputs from the test task. The outputs of the management of test and of the test tasks will become inputs to subsequent processes and activities, as appropriate.
- e) Schedule: Describe the schedule for the test tasks. Establish specific milestones for initiating and completing each task, for the receipt of each input, and for the delivery of each output.
- f) Resources: Identify the resources for the performance of the test tasks. Specify resources by category (e.g., staffing, tools, equipment, facilities, travel budget, and training).
- g) Risks and Assumptions: Identify the risk(s) (e.g., schedule, resources, technical approach, or for going into production) and assumptions associated with the test tasks. Provide recommendations to eliminate, reduce, or mitigate risk(s).
- h) Roles and responsibilities: Identify for each test task the organizational elements that have the primary and secondary responsibilities for the execution of the task, and the nature of the roles they will play.

## **2.2. Test documentation requirements**

This section defines the purpose, format, and content of all other testing documents that are to be used (in addition to those that are defined in MTP Section 2.4). If the test effort uses test documentation or test levels different from those in this standard (i.e., component, component integration, system, and acceptance), this section needs to map the documentation and process requirements to the test documentation contents defined in this standard.

## **2.3 Test administration requirements**

This section describes the anomaly resolution and reporting processes, task iteration policy, deviation policy, control procedures and standards, practices, and conventions. These activities are needed to administer the tests during execution.



### 2.3.1 Anomaly resolution and reporting

Describes the method of reporting and resolving anomalies, including the standards for reporting an anomaly, the Anomaly Report distribution list, and the authority and time line for resolving anomalies. This section of the plan defines the anomaly criticality levels.

### 2.4.2 Task iteration policy

Describes the criteria used to determine the extent to which a testing task is repeated when its input is changed or task procedure is changed (e.g., reexecuting tests after anomalies have been fixed). These criteria may include assessments of change, integrity level, and effects on budget, schedule, and quality.

### 2.3.3 Deviation policy

Describes the procedures and criteria used to deviate from the MTP and level test documentation after they are developed. The information required for deviations includes task identification, rationale, and effect on system/software quality. It also identifies the authorities responsible for approving deviations.

### 2.3.4 Control procedures

Identifies control procedures applied to the test activities. These procedures describe how the software-based system and software products and test results will be configured, protected, and stored. These procedures may describe quality assurance, configuration management, data management, or other activities if they are not addressed by other efforts. Describes how the test activities comply with existing security provisions and how the test results are to be protected from unauthorized alterations.

### 2.3.5 Standards, practices, and conventions

Identifies the standards, practices, and conventions that govern the performance of testing tasks including, but not limited to, internal organizational standards, practices, and policies.

## 2.4. Test reporting requirements

Specifies the purpose, content, format, recipients, and timing of all test reports. Test reporting consists of Test Logs, Anomaly Reports, Level Interim Test Status Report(s), Level Test Report(s), and the Master Test Report. Test reporting may also include optional reports defined by the user of this standard. The format and grouping of the optional reports are user defined and will vary according to subject matter

## 3. General

Introduces the following subordinate sections. This section includes the glossary of terms and acronyms. It also describes the frequency and the process by which the MTP is changed and baselined. It may also contain a change-page containing the history of the changes (date, reason for change, and who initiated the change).

### **3.1. Glossary**

Provides an alphabetical list of terms that may require definition for the users of the MTP with their corresponding definitions. This includes acronyms. There may also be a reference to a project glossary, possibly posted online.

### **4.2. Document change procedures and history**

This section specifies the means for identifying, approving, implementing, and recording changes to the MTP. This may be recorded in an overall configuration management system that is documented in a Configuration Management Plan that is referenced here. The change procedures need to include a log of all of the changes that have occurred since the inception of the MTP. This may include a Document ID (every testing document should have a unique ID connected to the system project), version number (sequential starting with first approved version), description of document changes, reason for changes (e.g., audit comments, team review, system changes), name of person making changes, and role of person to document (e.g., document author, project manager, system owner). This information is commonly put on an early page in the document (after the title page and before Section 1). Some organizations put this information at the end of the document.



## 4.3 IEEE 829 Level Test Plan (LTP)

Again the same caveats apply to this section as to the previous section. Specifically, the level test plan described in the 829 standard is a complex and fine grained plan that does not have to be followed “out of the box” but should be used to guide the development of your own level test plans.

The level test plan is what you will often see referred to as just the “test plan” in many references since up until the 2008 standard, it was the only definition of a test plan. While the MTP is very broad in scope, it is in the level test plan that we get down to a lot of the nitty gritty details of test planning. The main reason for this is that a lot of the detailed planning is dependent on the type of testing we are doing (functional testing as opposed to performance testing for example.).

The rest of this section is summarized from the IEEE 829-2008 Standard as a short reference. In a few places, the text of the standard has been rewritten to make it easier to follow.

Each level test plan describes the the scope, approach, resources, and schedule of the testing activities for a specified level of testing. It identifies the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the associated risk(s). In the title of the plan, the word “Level” is replaced by the organization’s name for the particular level being documented by the plan (e.g., Component Test Plan, Component Integration Test Plan, System Test Plan, and Acceptance Test Plan).

In most projects, there are different test levels requiring different resources, methods, and environments. As a result, each level is best described in a separate plan. Different level test plans may require different usage of the documentation content topics listed below. Some examples of test levels are follows:

1. Each basic software component (unit testing) and database.
2. Integrated components for integration testing.
3. Tests for each software requirement.
4. Software qualification testing for all requirements.
5. Systems integration: aggregates of other software configuration items, hardware, manual operations, and other systems. It is not unusual for large systems to have multiple levels of integration testing.
6. System qualification testing for system requirements.

Other possible examples of levels include operations, installation, maintenance, regression, and nonfunctional levels such as security, usability, performance, stress, and recovery. Any one of the example levels may be more than one level for an organization; e.g. acceptance testing may be two levels: Supplier's System and User's Acceptance test level

### ***4.3.1 Level Test Plan Outline***

#### ***1. Introduction***

- 1.1. Document identifier
- 1.2. Scope
- 1.3. References
- 1.4. Level in the overall sequence
- 1.5. Test classes and overall test conditions

#### ***2. Details for this level of test plan***

- 2.1. Test items and their identifiers
- 2.2. Test Traceability Matrix
- 2.3. Features to be tested
- 2.4. Features not to be tested
- 2.5. Approach
- 2.6. Item pass/fail criteria
- 2.7. Suspension criteria and resumption requirements
- 2.8. Test deliverables

#### ***3. Test management***

- 3.1. Planned activities and tasks; test progression
- 3.2. Environment/infrastructure
- 3.3. Responsibilities and authority
- 3.4. Interfaces among the parties involved
- 3.5. Resources and their allocation
- 3.6. Training





3.7. Schedules, estimates, and costs

3.8. Risk(s) and contingency(s)

#### **4. General**

4.1. Quality assurance procedures

4.2. Metrics

4.3. Test coverage

4.4. Glossary

4.5. Document change procedures and history

### **4.3.2 Detailed Descriptions**

#### **1. Introduction**

Introduces the following subordinate sections. This section identifies the document and puts it in context of the test effort and the project-specific lifecycle. This section also identifies the types of tests and test conditions for the specific level of testing.

##### **1.1. Document identifier**

Same as for the MTP

##### **1.2. Scope**

The test approach identifies what will be tested and in what the test levels will be done. The test approach identifies the rationale for testing or not testing, and it identifies the rationale for the selected order of testing. The test approach describes the relationship to the development methodology. The test approach may identify the types of testing done at the different levels. For example, “thread testing” may be executed at a system level, whereas “requirements testing” may take place at the component integration as well as at a systems integration level.

##### **1.3. References**

Same as for the MTP

##### **1.4. Level in the overall sequence**

This section describes this level's context in the overall test hierarchy or sequence. This is best supported by an illustration. It may be combined with Section 1.2 above on Scope.

### **1.5. Test classes and overall test conditions**

This section summarizes the unique nature of this particular level of test. This is additional detail within the basic scope defined in 1.2. For example, it provides descriptions for the particular level such as:

- a) Component test would focus on the desired attributes (e.g., logic) of each component individually or in groups or as a single set.
- b) Each level of integration test would have an inventory of interfaces to exercise
- c) System tests would focus on meeting the system's requirements
- d) Acceptance tests would focus on the attributes of fitness for use

Some examples of possible classes within one or more levels are as follows:

- Positive (or valid) testing of input values that should be processed successfully
- Negative (or invalid) values that should NOT process but do provide an appropriate error processing, like an error notification message to a user
- All boundary values, including those just above, just below, and just on each limit
- Normal values based on usage profiles
- Exploratory tests based on compatibility requirements with prior version

## **2. Details for this level of test plan**

This section introduces the following subordinate sections. This section describes the specific items to be tested at the designated level and provides a Test Traceability Matrix that links the items to be tested with the requirements. It is in this section that the approach is described along with the pass/fail criteria and suspension/resumption criteria, and test deliverables are identified.

### **2.1. Test items and their identifiers**

This section identifies the test items (software or system) that are the object of testing, e.g., specific attributes of the software, the installation instructions, the user instructions, interfacing hardware, database conversion software that is not a part of the operational system) including their version/revision level. Also identified are any procedures for their transfer from other environments to the test environment.

References are supplied to the test item documentation relevant to an individual level of test, if it exists, such as:

- Requirements
- Design
- User's guide



- Operations guide
- Installation guide

As well as any Anomaly Reports relating to the test items and the identification of any items that are to be specifically excluded from testing.

## **2.2. Test Traceability Matrix**

Provides a list of the requirements (software and/or system; may be a table or a database) that are being exercised by this level of test and shows the corresponding test cases or procedures. The requirements may be software product or software-based system functions or nonfunctional requirements for the higher levels of test, or design or coding standards for the lower levels of test. This matrix may be part of a larger Requirements Traceability Matrix (RTM) referenced by this plan that includes requirements for all levels of test and traces to multiple levels of life cycle documentation products. It may include both forward and backward tracing.

## **2.3 Features to be tested**

This section identifies all software product or software-based system features and combinations of software or system features to be tested.

## **2.4. Features not to be tested**

This section identifies all features and known significant combinations of features that will not be tested and the rationale for exclusion.

## **2.5. Approach**

Describes the overall approach for the level of testing. For each major feature or group of features, specify the approach that will ensure that they are adequately tested. The approach may be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to do each one.

Features to be tested, features not to be tested, and approaches (LTP Section 2.5) are commonly combined in a table called a Test Matrix. It contains a unique identifier for each requirement for the test (e.g., system and/or software requirements, design, or code), an indication of the source of the requirement (e.g., a paragraph number in the source document), and a summary of the requirement and an identification of one or more generic method(s) of test. Some examples of possible methods are:

- a) Black box: The test inputs can be generated and the outputs captured and completely evaluated from the outside of a test item; i.e., test cases are developed from the test item specification, only without looking at the code or design.
- b) : Considers the internal structure of the software (e.g., attempts to reach all of the code). Commonly requires some kind of test support software.
- c) Analysis: Just viewing the outputs cannot confirm that the test executed successfully; some kind of additional computations, simulations, studies, and so on will be required.

d) Inspection: This is a static test; the code or documentation is read and examined without being executed.

The Test Matrix may be combined with the Test Traceability Matrix. The Test Traceability Matrix links each requirement with one or more test cases. The test coverage requirements in part 4.4 may reference or be combined with this section.

## **2.6. Item pass/fail criteria**

Specifies the criteria to be used to determine whether each test item has passed or failed testing. This is commonly based on the number of anomalies found in specific severity category(s). For example, require that there are no category 1 or 2 anomalies remaining.

## **2.7. Suspension criteria and resumption requirements**

Specifies the criteria used to suspend all or a portion of the testing activity on the test items associated with this plan. Specify the testing activities that must be repeated when testing is resumed.

## **2.8. Test deliverables**

Specifies all information that is to be delivered by the test activity (documents, data, etc.). The following documents may be included:

- a) Level Test Plan(s)
- b) Level Test Design(s)
- c) Level Test Cases
- d) Level Test Procedures
- e) Level Test Logs
- f) Anomaly Reports
- g) Level Interim Test Status Report(s)
- h) Level Test Report(s)
- i) Master Test Report

Test input data and test output data may be identified as deliverables. Test tools may also be included. If documents have been combined or eliminated, then this list will be modified accordingly. It also describes the process of delivering the completed information to the individuals (preferably by position, not name) and organizational entities that will need it. This may be a reference to a Configuration Management Plan. This delivery process description is not required if it is covered by the MTP and there are no changes.

# **3. Test management**



Introduces the following subordinate sections. This section describes the test activities and tasks for the specified level and the progression of these. It is here that the infrastructure, responsibilities and authority, organizational interfaces, resources, training, schedules, and risk(s) are identified if they are not identified or described in a higher level document such as the MTP.

### ***3.1. Planned activities and tasks; test progression***

Identifies the set of tasks necessary to prepare for and perform testing. Identify all inter-task dependencies. Identifies any significant constraints such as test item availability, testing resource availability, and deadlines. It may be desirable to combine all of the documentation content topics about resources (Sections 3.1 through 3.8) into one section. This content topic and the next one could be combined in a chart showing entry criteria, person responsible, task, and exit criteria

### ***3.2. Environment/infrastructure***

This section specifies both the necessary and the desired properties of the test environment and any relevant test data. This may include the physical characteristics of the facilities, including the hardware, the off-the-shelf software, the test support tools and databases, personnel (identifying their organizations as appropriate), and anything else needed to support the test. It includes the environment for setup before the testing, execution during the testing (including data capture), and any post-testing activities (e.g., data reduction and analysis). Also specify the level of security that must be provided for, and any safety issues related to, the testing facilities, software, and any proprietary components. It may include externally provided content topics (possibly provided by third parties) including systems and/or subsystems and identifies the source(s) for all of these needs.

### ***3.3. Responsibilities and authority***

This section identifies the individuals or groups responsible for managing, designing, preparing, executing, witnessing, and checking results of this level of testing, and for resolving the anomalies found. In addition, it identifies those persons responsible for providing the test items identified in Section 2 and the environmental needs identified in Section 3.2.

The responsible parties may include the developers, testers, operations staff, user representatives, technical support staff, data administration staff, and quality support staff. They may be participating either full or part time. They may have primary or secondary responsibilities.

### ***3.4. Interfaces among the parties involved***

Describes the means and the contents of communication between the individuals and groups identified in Section 3.5. A figure that illustrates the flow of information and data may be included.

### ***3.5. Interfaces among the parties involved***

Describes the means and the contents of communication between the individuals and groups identified in Section 3.5. A figure that illustrates the flow of information and data may be included.

### **3.6. Resources and their allocation**

Delineates any additional required resources that are not already documented by other parts of the plan (test environment needs are in Section 3.2, personnel resources are in Section 3.5, and schedule needs are in Section 3.7). This includes both internal and external resources (such as outside test resources, e.g., test labs, outsourcing, etc.).

### **3.7. Schedules, estimates, and costs**

Includes test milestones identified in the software or system project schedule as well as all test item transmittal events and defines any additional test milestones needed. Estimate the times required to do each testing task. specifies the schedule for each testing task and test milestone. For each testing resource (i.e., facilities, tools, and staff), specifies its periods of use.

### **3.8. Risk(s) and contingency(s)**

Identifies the risk issues that may adversely impact successful completion of the planned level testing activities. Specifies potential impact(s) of each risk, with contingency plan(s) for mitigating or avoiding the risk. The risk(s) and contingency(s) that are current at the time of signoff of the first document release may change as the project continues, and then the risk(s) and contingency(s) can be tracked in a separate document (risk register) that is not under signoff control.

This section may be a reference to a master project plan, or it may supplement it with a greater level of detail. This section 3.8 may be combined with Section 3.1, planned activities, and tasks.

## **4. General**

### **4.1. Quality assurance procedures**

Identifies the means by which the quality of testing processes and products will be assured. Includes or references anomaly tracking and resolution procedures. The quality assurance information may be described in a Quality Assurance Plan or Standard Procedure that can be referenced.



#### **4.2. Metrics**

Identifies the specific measures that will be collected, analyzed, and reported. The metrics specified here are those that only apply to this particular test level (the global metrics are described in MTP Section 1.5.6). This may be a reference to where it is documented in a Quality Assurance Plan or as a part of documentation in an overall measurement program.

#### **4.3. Test coverage**

Specifies the requirement(s) for test coverage. Test coverage is an indication of the degree to which the test item has been reached or “covered” by the test cases, including both the breadth and depth. The type of coverage that is relevant varies by the level of test. For example, unit test coverage is often expressed in terms of percentage of code tested, and software or system validation test coverage can be a percentage of requirements tested. There is a need for specification of coverage or some other method for ensuring sufficiency of testing.

#### **4.4. Glossary**

Same as for the MTP

#### **4.5. Document change procedures and history**

Same as for the MTP



## 4.4 Software and System Integrity Levels

*From the IEEE 829-2008*

In addition to a similar role in all other life cycle processes, a scheme of integrity levels provides the structured means for setting the breadth and depth of testing. A high integrity level requires additional test tasks than does a low integrity level. A high integrity level requires testing that is more in-depth than does a low integrity level. Without a requirement for a certain integrity level, the tester will not know which functions, requirements, or products require only a cursory effort and which require an intense effort.

The 829-2008 standard uses integrity levels to determine the testing tasks to be performed. Integrity levels may be applied to requirements, functions, groups of functions, components, and subsystems. Some of these may not require the assignment of an integrity level because their failure would impart no negative consequence on the intended system operation. The integrity scheme may be based on functionality, performance, security, or some other system or software characteristic.

This standard provides as an example the four-level software integrity level scheme shown below. The categories of the software integrity level scheme shown below are defined based on the seriousness of the consequence(s) (of incorrect behavior during execution) and on the potential for mitigation (taking steps to lessen risk by lowering the probability of a risk event's occurrence or reducing its effect should it occur). Each level may have an associated descriptive word, e.g.:

4. **Catastrophic:** Software must execute correctly or grave consequences (loss of life, loss of system, environmental damage, economic or social loss) will occur. No mitigation is possible.
3. **Critical:** Software must execute correctly or the intended use (mission) of system/ software will not be realized causing serious consequences (permanent injury, major system degradation, environmental damage, economic or social impact). Partial-to-complete mitigation is possible.
2. **Marginal:** Software must execute correctly or an intended function will not be realized causing minor consequences. Complete mitigation possible.
1. **Negligible:** Software must execute correctly or intended function will not be realized causing negligible consequences. Mitigation not required.



## 4.5 Test Plans in Rational Quality Manager

The implementation of test plans in the RQM is more or less along the lines of the IEEE standard but there is more variability in the sort of content that RQM permits. Basically, the underlying motivation is that while all projects should have a test plan, to go the level of rigour and detail needed for compliance to the 829 standard may be onerous, excessive and way beyond the quality needs of a particular project.

The following is the from the IBM documentation

### 4.5.1 Test Plan Overview

#### Test Plan Sections

A test plan includes several predefined sections. Each section includes its own editor. Some sections, such as the Business Objectives and Test Objectives sections, consist of a rich-text editor for text input. These editors provide common formatting features such as table support, font support, bullets, and numbered lists.

Other test plan sections, such as the Requirements and Tests Cases sections, provide links to these additional test artifacts. Still other sections include tables that establish and measure against criteria such as Exit Criteria, Entry Criteria, Quality Objectives, and Test Schedules. You can add your own sections and remove sections that you do not need using the Manage Sections feature.

#### Test Plan Templates

A test plan is based on a test plan template. When you create a new test plan, you choose a template to base it on. You can create a new test plan from one of the predefined templates, or from a test plan template that you create. You can also designate which test plan templates should appear by default within a new test plan.

A test plan template is a collection of test plan sections. You create a template by adding and removing existing sections or creating new sections. If the section names do not match what you are accustomed to, create new sections and add them to a template.

Each test organization can design their own test templates. This flexibility makes the test plan suitable for both agile and formal test teams and for teams that perform different kinds of testing, such as functional regression testing, performance testing, system verification testing, globalization testing, and so on.

## Test Plan Categories

Test plan categories are used in the Summary section of a test plan. You can use test plan categories to help organize your test plans into logical groups. Later, when you view a list all of your test plans, you can sort, filter, and group the test plans in the list by using the categories.

By default, three test plan categories are provided: Product, Release, and Iteration. You can add more choices for products, releases, and iterations. You can also create your own categories, such as Test Type, Component, or Division.

## Test Plan Custom Attributes

If an administrator defined custom attributes for test plans, they display in the Summary section of a test plan under Attributes. Like categories, custom attributes can be used to help organize your test artifacts. With categories, the field values are restricted to text format; however, with custom attributes you can also use integer and date formats. Custom attribute values are free-form and not restricted to a predefined set, allowing any data to be entered and used when working with test artifacts. When you view a list all of your test plans, you can filter the list by using the custom attributes.

## Test Plan Work Items

A work item is a way of keeping track of the tasks and issues that your team needs to address. The status and number of work items are indicators of the health of your project. If you have configured an integration with a defect provider (such as the Change and Configuration Management application or IBM Rational ClearQuest), you can create work items for team members directly from sections of the test plan.

For example, to assign a section of a test plan to another team member, click Work Item: Create in the test plan section and specify a summary description, owner, and due date for the work item. This results in the creation of a new work item.

## Test Plan Relationship with Other Test Artifacts

From a test plan you can manage other test artifacts, such as test cases and requirements. However, it is also possible to use the product without a test plan.

## Test Execution Status

From the test plan overview you can track the test execution status of test cases and test suites. You can view the relationship between test estimates and actual time spent, test execution progress in relation to the assigned weight (or point) values, and test execution status in relation to the number of tests attempted.



## Test Plan Tasks

Here are some of the many tasks that you can perform with test plans:

1. Set up a formal review process that is visible at all times to all members of the larger project team
2. View and work with sets of requirements that are linked from a requirements management application to test plans
3. Create a test plan snapshot and use it as the basis for a new test plan
4. Create test cases and associate them with the test plan
5. List platform coverage and the various test environments supported and tested by the test plan
6. Estimate the overall test planning and test execution effort
7. Define schedules for each test iteration.
8. Define business objectives, test objectives, quality goals, and entrance and exit criteria

### 4.5.2 Test Plan Development Checklist

	Test plan task	Task details
<input type="checkbox"/>	Before you create the initial test plans in an iteration, review preparatory tasks, considerations, and practices.	<a href="#">Preparing to test</a>
<input type="checkbox"/>	Create a test plan.	<a href="#">Creating test plans</a>
<input type="checkbox"/>	Link the test plan to the project requirement collections to ensure that your testing effort covers the requirements.	<a href="#">Linking to requirement collections</a>
<input type="checkbox"/>	Create a risk assessment in the test plan to support risk-based testing practices. A risk assessment is useful for prioritizing what needs to be tested based on the risk that is later associated with your test artifacts.	<a href="#">Creating the initial risk assessment</a>
<input type="checkbox"/>	Establish a test schedule by associating the test plan with iterations from a project area or team area timeline.	<a href="#">Creating test schedules</a>
<input type="checkbox"/>	Add predefined or user-defined quality objectives to define this information: <ul style="list-style-type: none"> <li>• The overall quality goals for the release or iteration that the test plan covers</li> <li>• The required entry and exit criteria for testing</li> </ul>	<a href="#">Adding quality objectives to a test plan</a>
<input type="checkbox"/>	If needed, assign sections in the test plan to be completed by other team members.	<a href="#">Managing the team with work items</a>
<input type="checkbox"/>	Submit the completed test plan for review and incorporate any review feedback.	<a href="#">Assigning reviewers and approvers</a> <a href="#">Tracking reviews</a>

### 4.5.3 Creating Test Plans

The test plan captures the test requirements, test cases, test execution criteria, and other information for a particular test cycle. A test plan is based on a test plan template. When you create a test plan, you choose a template to base it on.

A test plan consists of several sections that are defined in the template. You can add your own sections to the test plan and you can remove sections that are supplied with the template.

#### Procedure

1. In the main menu, click Planning > Create > Create Test Plan.

The new test plan opens, with a table of contents on the left and an editor on the right.

2. At the top of the new test plan, enter a unique test plan name.
3. Select a test plan template from the list, if available.
4. If team areas are enabled for the project area, select a team area from the list. (Note You must explicitly enable team areas in Manage Project Properties.)
5. Select a test plan template from the list.

For information about the templates provided with the application, see the Test plan template reference.

6. Select an owner and priority for the test plan.
7. Add a description of the test plan.
8. Complete the test plan Summary section or assign the Summary section to another team member to complete.
9. To complete the test plan Summary yourself:

- a) Define categories.

You can use categories to define a hierarchical organization of test plans, test cases, test scripts, and other test artifacts. You can also define subcategories and other category relationships. You can use the default categories that are provided or create your own. To define categories, click the Manage Test Plan Categories icon (Manage category).

- b) Assign values to the custom attribute fields.

If an administrator defined any custom attributes for test plans, they display in the Attributes section of the Summary. With categories, the field values are restricted to text format; however, with custom attributes you can also use integer and date formats. Custom attribute values are free-form and not restricted to a predefined set, allowing any data to be entered and used when working with test artifacts. When you view a list all of your test plans, you can filter the list by using the custom attributes.



10. To assign the Summary section, or any section of the test plan, to another team member to complete, click Work Item: Create and complete the fields in the form. For more information about using work items, see Managing the team with work items.
11. Complete the other test plan sections as needed or assign them to other team members to complete.
12. To open all test plan sections in the editor at once, select Show All Sections, located on the left, just under the list of sections.
13. To add your own, customized test plan sections or to remove sections that you do not need, click Manage Sections.
14. Click Save to save the new test plan.
15. Optional: Click the Duplicate icon (Copy ) in the upper right corner of the test plan to make a copy of the test plan.
16. Optional: Click the Print View icon (Print view ) in the upper right corner of the test plan to display a printable version of the test plan.
17. Optional: Click the Set to Lock/Unlock icon (Change lock) in the upper right corner of the test plan to lock or unlock the test plan. Depending on how your project is configured, you might be required to sign this action electronically by providing your password and a comment. When the test plan is locked, the editable fields are disabled for editing. (Note Permission to lock a test plan depends on your role-based permissions.)  
  
To reverse the locking attribute, click the icon again.
18. Optional: Click the down arrow next to the Print PDF icon (Export to PDF ) in the upper right corner of the test plan to save the test plan as a PDF.

## Creating Test Plans From a Template

To create a test plan from a template:

1. In the main menu, click Planning > Create Test Plan.
2. In the Template field, select an available template.  
  
Note If only one template is available for a test plan, the Template field is not available. The new test plan will use the default template.
3. Save the test plan. After you save the test plan, you cannot change it to use a new template. You can modify the sections that are included in the test plan by clicking the Manage Sections link below the table of contents.

## 4.5.4 Managing Test Artifact Sections

You can add and remove sections of individual test plans, test cases, test suites, and keywords. You can also configure the sections that are included in test plan, test case, and test suite templates.

### Procedure

1. Open a test plan, test case, test suite, or keyword.
2. From the list of test artifact sections, click Manage Sections to open the Manage Sections window.
3. Select Show all sections to display all the available sections.

Note You can set a user preference to display all sections of an artifact by default. To set the preference, click the User Profile (User Profile menu) icon in the upper-right portion of the banner, click My Preferences for Quality Management > Editor Table of Contents and then select Show all sections in the editor. With this preference set, you see all sections when you open an artifact. The sections are displayed in a closed state and you must double-click the sections to open them.

4. To add an established section, select the section from the Available Sections list, and move it to the Selected Sections list by clicking the Move right (Move right) icon.

Tip: Press Ctrl and click items to make multiple selections. You can also move all of the sections at once by clicking the Move all right (Move all right) icon.

5. To remove a section, move the section from the Selected Sections list to the Available Sections list by clicking the Move left (Move left) icon.
6. To reorder sections, use the Move up (Move up) and Move down (Move down) icons.
7. To add a customized section, click the Create Section (Create section) icon. The New Custom Section window opens.
  - a) In the Section Type field, select the type of content for the new template section, Rich Text Editor or Grid.
  - b) If you select Grid, click the Add (Add) icon to add a column and then type the column name.
  - c) If you select Rich Text Editor, you can enter boilerplate content into the rich text field so that it is included in all templates that use the section.
  - d) Click OK to close the New Custom Section window.

Note To delete a custom section, select the section, and click the Delete (Delete) icon. You cannot delete standard sections of a template.

8. Click OK to close the Manage Sections window.
9. Click Save to save the test plan, test case, or test suite.

### What to do next





You can further customize your test artifacts by editing the name and description of sections. For example, you can change the name of the Summary section in a test plan to Overview, or change both the name and description of a custom section. The ability to edit section names and descriptions is controlled by the Edit Section Header permission. To change the description of a standard section (for example, to provide more detailed information about the categories and attributes in the Summary section), you must modify or create a template.

### ***4.5.5 Creating Master and Child Test Plans***

You create a master test plan by adding one or several child test plans to a test plan. You can use existing test plans as children or create new plans. A child test plan cannot have child test plans of its own.

#### **Procedure**

1. Open a test plan or create a new one.
2. Add the Child Test Plans section to the test plan.
  - a) In the area above the table of contents, click Manage Sections.
  - b) Select Show all sections to display all of the available sections.
  - c) In the Available Sections list, select Child Test Plans and move the item to the Selected Sections list by clicking the Add (Add) icon.
  - d) Optional: Move the child test plan to the top of the list by using the Up (Move up) icon.
  - e) Click OK to close the Manage Sections window.
3. In the table of contents, click the Child Test Plans section.
4. To add a child test plan to the master, click the Add Existing Test Plan icon (Add).
5. Select one or more test plans from the list and click OK.
 

Note If you have a long list of child test plans, you can sort the list by clicking on the column headers. You can also manage the list of test plans by using filters and queries.
6. To create a test plan and add it as a child, click the Add New Child Plan icon (Create test plan) and provide a name and description. Because the new child inherits the section information from the master, be sure to configure the master before creating the child. .
7. Optional: Click Roll Up Objectives to include child test plans in the calculation of quality objectives.

Quality objectives include the overall quality goals for a release, or the entrance or exit criteria. There are several automatically calculated quality objectives, such as Execution Record Pass Rate and the Number of Open Sev1 Defects allowed. Roll up means that when the child objectives are added to the master, the children are included in the calculation. For example, assume that the exit criteria for the master specifies that there must be fewer than five severity-one defects. If the master already has two severity-one defects, and a child has four, then the exit criteria for the master cannot be met if you include the child quality objectives.

8. Click Save to save the new master test plan.

### ***4.5.6 Linking to Requirement Collections***

You can ensure that your testing effort adequately covers the necessary requirements by linking requirement collections to your test plans. Later, you can assign each requirement to a test case and link to the test case from the test plan to ensure that the requirements have complete testing coverage.

#### **About this task**

You must configure the integration between the test project area and the requirements project area before you can create the link to the requirements collection.

Limitation: When you create a link to external artifacts in a CLM workbench source application, a link is automatically created back to the source artifact in the target application, which is called a back link. However, a state of synchronization between bi-directional links and the traceability between source and target artifacts are not guaranteed.

#### **Procedure**

1. Open a test plan or create a new one.
2. Add the Requirement Collection Links section to the test plan if it is not already included.
  - a) In the area above the table of contents, click Manage Sections.
  - b) Select Show all sections to display all of the available sections.
  - c) In the Available Sections list, select Requirement Collection Links and move the item to the Selected Sections list by clicking the Add (Add) icon.
  - d) Click OK to close the Manage Sections window.
3. Open the Requirement Collection Links section in a test plan.
4. Click the Add new links icon (Add).



5. Select a requirement collection from the list or type some words from the requirement collection name and click OK.

A link to the requirement collection is displayed in the Summary column.

6. Point the cursor over one of the links.

Information about the requirement collection is displayed in rich hover text.

7. Click Show More to see additional information about the requirement collection.
8. Close the rich hover text.
9. Click the link itself to open the requirement collection in the Requirement Management application.
10. After the requirement collection opens, you can review the requirements in the collection and return to the test plan by clicking the link in the Links tab.

## Bidirectional Links

When a source artifact and target artifact link to each other, this is known as bidirectional linking. The source and target applications store their links in separate databases, and the source application does not have access to the database in the target application. Certain transactional operations, such as when artifacts are copied or deleted, can occur in the source application server. In these cases, corresponding back links in artifacts in the target application are not automatically added or removed.

There can also be instances where back links are intentionally removed in the target application without removing the corresponding link in the source application. As a result, if bidirectional links are not in a state of synchronization, traceability between the source and target artifacts can appear differently based upon your starting view point.

### ***4.5.7 Associating Test Plans with Requirement Collections***

When the applications of the Rational solution for Collaborative Lifecycle Management (CLM) are integrated, you can link test plans to requirement collections, modules, and saved module views in the Requirements Management application. Later, you can assign each requirement in the collection to a test case and link to the test case from the test plan to ensure that the requirements have complete testing coverage.

**Limitation** When you create a link to external artifacts in a CLM workbench source application, a link is automatically created back to the source artifact in the target application, which is called a back link. However, a state of synchronization between bidirectional links and the traceability between source and target artifacts are not guaranteed.

## Procedure

1. Open or create a test plan, and in the table of contents, click Requirement Collection Links.
2. If there is more than one requirements provider application, click the Requirement Collections tab.
3. Click the Add new links icon (Add). The Requirement Links window opens.
4. The requirement projects are displayed on the right. Expand the project that contains the requirement collections, and then open the requirements folder that contains the collection you want to associate with your test plan. Requirement collections that are available for association with test plans are displayed on the left.
5. Select the requirement collection to associate with the test plan. To select more than one requirement collection, press and hold the Ctrl key and select the collections.
6. Scroll to the bottom of the window, and click OK. The requirement collections are displayed in the Requirement Collection Links section of the test plan.
7. Click Save to save the changes that you made to the test plan.

## What to do next

You can also create links directly from the Requirements Management application artifacts to test plans. As you link collections, modules, and module views to test plans, the link types define the traceability relationships. After creating links, you can display a summary of the linked artifact or navigate to the artifact. You can also add a widget to your dashboard to monitor the status of linked artifacts.

You can reconcile the test plan with the requirements in the associated requirement collections to add test coverage for the requirements. During a reconcile operation, requirements without test coverage in the test plan are identified and you can choose to associate the requirements with existing test cases or to generate test cases for the requirements.

### ***4.5.8 Reconciling Test Plans with Requirements***

After you have associated requirement collections with test plans, the requirements in the Requirements Management application might have changed. Requirements might have been added, deleted, or modified. You can reconcile test plan coverage with the requirements in the Requirements Management application.



## About this task

When you reconcile requirement collections in a test plan, new requirements or requirements that have changed are displayed. You can update the requirements in the test plan and mark the corresponding test cases as suspect. You can also choose not to update the requirement link. If new requirements have been added to the requirement collection, you can generate new test cases or add the requirements to existing test cases to add test coverage for the requirements.

## Procedure

1. Open the test plan with which you associated the requirement collections, and go to the Requirement Collection Links section.
2. Click the Reconcile Requirements in Collections icon Reconcile Requirements in Collections and Reconcile Requirement icon. The Reconcile Requirements wizard opens and displays the requirements that were updated in the collection since the last reconciliation. If there are new requirements in the collection, requirements that do not have test coverage in the current test plan are displayed.
3. In the Pending Actions section, the Reconcile Test Coverage action is displayed. To add test coverage for new requirements, complete the following steps:
  - a) Select the check box next to one or more requirements; then in the action column, click the Expand/Collapse icon (Expand/Collapse). The Action menu opens.
  - b) In the Action menu, do one of the following:
 

If the selected requirements already have been linked to existing test cases, to add the test cases to the current test plan, click Add Test Case to Test Plan.

To generate new test cases for each selected requirement, from the Action menu, click Generate [n] Test Cases, where n is the number of test cases to generate; then complete the fields in the New Test Case window and click OK.

To generate a single new test case for all selected requirements, from the Action menu, click Generate Test Case; then complete the fields in the New Test Case window and click OK.

- a) Click Next.

If there are additional pending actions, the Reconcile Requirements window displays the actions and the list of requirements associated with each action: Reconcile Removed Items, Reconcile Deleted Items, and Reconcile Updated Items.

4. Optional: Reconcile requirements that were removed from collections or deleted from the project since the last reconciliation. Those requirements that have been moved out of the associated requirement collections or those requirements do not belong to any requirement collections, but are associated to the test cases in the current test plan, are listed for the Reconcile Unplanned Items action.

- a) In the Pending Actions section, select Reconcile Unplanned Items or Reconcile Deleted Items.
- b) Select the check box next to one or more requirements to reconcile unplanned or removed items, then do one of the following:

To ignore the changes for this operation, click the Ignore icon Ignore.

To accept changes, click the Mark Suspect icon Mark Suspect.

The operation is added to the Action column for the selected items.

5. Optional: Change the suspect status of test cases that are linked to modified requirements.

- a) In the Pending Actions section, select Reconcile Updated Items.
- b) Select the check box next to one or more requirements to reconcile modified items, then do one of the following:

To ignore the changes for this operation, click the Ignore icon Ignore.

To accept changes, click the Mark Suspect icon Mark Suspect.

To clear suspicion from the test cases, click the Clear Suspicion icon.

The operation is added to the Action column of the selected requirements.

6. For any modified requirements, to update the associated test case, select Create New Quality Task. You will be prompted to create a quality task for each test case that is associated with the selected requirements when you click Finish.

Note You can create a new quality task only for the requirements for which you chose to apply Mark Suspect. The Create a new Quality Task option is disabled when Clear Suspicion or Ignore is applied.

7. Optional: If reconciliation fails due to errors, such as from server connection issues, complete the following steps:

- a) In the Pending Actions section, select Reconcile Failed Items.
- b) Select the check box next to one or more requirements with failed items.
- c) To ignore the changes for this operation, click the Ignore icon (Ignore).

The operation is added to the Action column of the selected requirements.

8. Click Finish. Actions that you specified in the wizard are implemented.



9. If you chose to create a new quality task for modified test cases, a window opens prompting you to create the quality tasks.
  - a) Click the Create New Quality Task button.
  - b) Select the task type.
  - c) In the Summary field, enter a brief headline that identifies the work item.
  - d) In the Filed Against field, select a category that identifies the component or functional area that the quality task belongs to.
  - e) Assign an owner to the task.
  - f) Set a due date for the task to be completed.
  - g) Enter a description of the quality task and click OK.

A quality task is created in the Change and Configuration Management application. The task appears in the My Tasks widget of the personal dashboard for the owner that you assigned the quality task to. The Links section of the quality task shows a related test case link for the test case associated with this task.

10. Click Save.

## What to do next

To view changes to test cases in the test plan, in the table of contents, click Test Cases. The test cases whose suspect status you changed are displayed with the Suspect icon



## 4.5.9 Linking to Development Plans

You can improve overall collaboration by linking test plans to the various development plans used by the development team. You must configure the integration between the test project and the development project before you can create the link to the development plan.

Limitation: When you create a link to external artifacts in a CLM workbench source application, a link is automatically created back to the source artifact in the target application, which is called a back link. However, a state of synchronization between bidirectional links and the traceability between source and target artifacts are not guaranteed.

### Procedure

1. Open a test plan or create a new one.
2. Add the Development Plan Links section to the test plan if it is not already included.
  - a) In the area above the table of contents, click Manage Sections.
  - b) Select Show all sections to display all of the available sections.
  - c) In the Available Sections list, select Development Plan Links and move the item to the Selected Sections list by clicking the Add (Add) icon.
  - d) Click OK to close the Manage Sections window.
3. Open the Development Plan Links section in a test plan.
4. Click the Add new links icon (Add).
5. Select a plan from the list or type some words from the test plan name and click OK.

A link to the development plan is displayed under the Summary column.
6. Point the cursor over one of the links.

Information about the development plan is displayed in rich hover text.
7. Click Show More to see additional information about the development plan.
8. Close the rich hover text.
9. Click the link itself to open the development plan in the Change Management application.
10. After the development plan opens, you can review the work items in the plan and return to the test plan by clicking the link in the Links tab.



### 4.5.10 Creating Test Schedules

You can establish a test schedule by associating a test plan with iterations from a project area or team area timeline. You can also list key project dates, such as the date of the final release, code freeze, UI freeze, beta entry, or beta exit.

Test plan schedules are defined as iterations and timelines in the Jazz process, and as a result, test plan authors must have permission to create and modify iterations. This permission is applied at the Project Configuration level. You can find this permission in the Process section of Permitted Actions. The specific permission that is required is named "Modify the iteration structure". Optionally, you can create a test plan author role and assign this permission to that role. Otherwise, all of the Process permissions are enabled for the Test Team Member role by default.

#### About this task

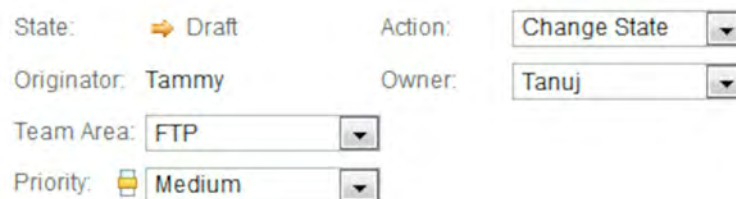
If a team area is associated with a particular timeline, you can select the team area in the test plan and associate the test plan schedule with the iterations in that timeline. For more information, see *Timelines and Iterations®* in *Rational® Quality Manager*.

**Note** In the Quality Management application, you must explicitly enable the use of team areas and multiple timelines.

#### Procedure

1. Open a test plan.

If team areas are enabled and a team area is associated with a timeline, you can associate the test plan with a team area by selecting a team area from the list.





The image shows a configuration form for a test plan. It includes the following fields and values:

- State:** Draft (with a yellow flag icon)
- Originator:** Tammy
- Team Area:** FTP (selected from a dropdown menu)
- Priority:** Medium (selected from a dropdown menu with a yellow flag icon)
- Action:** Change State (with a dropdown arrow)
- Owner:** Tanuj (selected from a dropdown menu)



2. Open the Test Schedules section of the test plan.
3. Click Browse.
4. In the Browse window, select an iteration and click OK.

The test schedule is populated with the selected iteration and its child iterations as shown in the following image:

**Test Schedules**  Define the test schedule for this Test Plan.


Show All  Items per page

Test Schedule: Iteration: [Browse](#) [Clear](#)


Name	Description	Points	Planned Defects	Planned Start D:	Planned End Da	Planned Duratio	Actions
4.0 [2/6/2012 ...]		0	0	Feb 6, 2012	Jun 1, 2012	116 d	 
M1 [2/13/2012 ...]		0	0	Feb 13, 2012	Feb 24, 2012	11 d	
M2 [2/27/2012 ...]		0	0	Feb 27, 2012	Mar 16, 2012	18 d	
M3 [3/12/2012 ...]		0	0	Mar 12, 2012	Mar 23, 2012	11 d	
M4 [3/26/2012 ...]		0	0	Mar 26, 2012	Apr 6, 2012	11 d	

Tip: To get multiple iterations in the test schedule, you must define a hierarchy of iterations. In the previous example, the top-level iteration (4.0) represents a release and the child iterations (M1, M2, and so on) represent iterations or milestones within the release.

- When the iterations are displayed in the test schedule, select an iteration, and in the Actions column, click the Edit (Edit) icon. The Edit Properties window opens as shown in the following image:

**Edit Properties** 

M1 [2/13/2012 - 2/24/2012]

Description:	<input type="text"/>	Defect Validity Rate	<input type="text" value="0"/>
Points	<input type="text" value="0"/>	Planned Start Date	<input type="text" value="Feb 13, 2012"/> 
Planned Defects	<input type="text" value="0"/>	Planned End Date	<input type="text" value="Feb 24, 2012"/> 

Note The planned start and end dates are inherited from the dates in the timeline, as are the actual dates that are shown in brackets next to the iteration name. Initially, the planned dates and actual dates are identical. To update the actual dates, use the timeline editor. When you make updates in the timeline editor, the new dates are reflected in the iteration name. The planned dates do not change. If you need to change the planned dates, make that change in the test schedule, not the timeline editor.

Name	Planned Start D:	Planned End D:	Planned Duration
4.0 [2/6/2012 - 6/1/2012]	Feb 6, 2012	Jun 1, 2012	116 d
M1 [2/13/2012 - 2/24/2012]	Feb 13, 2012	Feb 24, 2012	11 d
M2 [2/27/2012 - 3/16/2012]	Feb 27, 2012	Mar 16, 2012	18 d
M3 [3/12/2012 - 3/23/2012]	Mar 12, 2012	Mar 23, 2012	11 d
M4 [3/26/2012 - 4/6/2012]	Mar 26, 2012	Apr 6, 2012	11 d



6. Provide the detailed iteration properties.
  - a) In the Description field, type a description for the iteration.
  - b) In the Points field, type a points value, which is an estimate of the execution effort for the iteration. Alternatively, you can provide more detailed information about the distribution of these points over time in the Plan Details window. See step 7.
  - c) In the Planned Defects field, type the number of projected defects.
  - d) In the Defect Validity Rate field, type the number of defects that are determined valid.
  - e) If necessary, click the planned start date and use the calendar to select the date. Note If you did not initially supply a planned end date, supply one now.
  - f) Click OK to save the iteration properties.
  - g) Repeat for each iteration in the schedule.
  - h) Click Save to save the schedule.
7. Optional: You can click the Plan Details (Plan Details) icon to estimate the work to be done and distribute points over the iteration. The plan details are reflected in the Planned Attempted and Planned Complete points on the Execution Trend report. Point estimates that you enter in this window will override the points value that you previously provided for the iteration.
  - a) To automatically distribute points over an iteration, click the Auto Generation icon.
  - b) To manually distribute points over an iteration, Click the Add Row (Add table row) icon, and type the estimated values for the date, the points to be attempted by that date, and the points to be completed by that date. Add new rows for each date, and then click OK.

**Execution Progress Planning**

**Plan Details**

Date	Points attempted	Points completed	Comment
Wed Sep 01 2010 00:00:00 GMT-0400 (Eastern Daylight Time)	10	0	
Wed Sep 08 2010 00:00:00 GMT-0400 (Eastern Daylight Time)	25	10	
Wed Sep 15 2010 00:00:00 GMT-0400 (Eastern Daylight Time)	50	25	
Wed Sep 22 2010 00:00:00 GMT-0400 (Eastern Daylight Time)	80	60	
Thu Sep 30 2010 00:00:00 GMT-0400 (Eastern Daylight Time)	100	75	

OK Cancel

8. List key project dates:
  - a) Click the Add Key Dates icon (Add) to add a date that you want to track in the test plan.
  - b) Repeat the previous step for each date to include.
  - c) Click Save to save the schedule.
  - d) Update the dates as needed as the project evolves.

### ***4.5.11 Estimating the Overall Size of the Test Effort***

In the early stages of a test project, you can provide high-level estimates of the time required to complete your test planning activities and the time or effort required to run all of your tests. These estimates are often based on what is known about the project requirements.

#### **Procedure**

1. Open the Test Estimation section of a test plan.
2. Select a unit of measurement for the Planning Effort from the list.

The following units are available:

  - a) H: person hours
  - b) PD: person days
  - c) PM: person months
  - d) PY: person years
3. Provide an estimate for Planning Effort using the unit of measurement chosen in the previous step.
4. Select a unit of measurement for Execution Effort from the list.
5. Provide an estimate for Execution Effort using the unit of measurement chosen previously.
6. Click Save to save the estimate.

#### **What to do next**

As the project develops, individual testers can make more detailed estimates of the required effort to run individual test cases. Then you can use the test plan, the list views, and test execution records to measure your progress and provides input to several reports.



## 4.6 Planning Test Environments

Planning your test environment includes defining your platform coverage, setting up test plan environment types, generating new test environments, and adding existing test environments to a test plan.

### 4.6.1 Platform Coverage and Test Environments

In the Test Environments section of a test plan, you can list the software and hardware platforms that you hope to cover and use that list to generate the actual test environments that you need to run your tests.

The Test Environments section of the test plan includes two tabs:

- Platform Coverage
- Test Environments

Use the Platform Coverage tab to create a non-binding list of platforms that you hope to cover. This is where you specify the required operating systems, browsers, supported hardware platforms, CPU-types, and so on. Typically, you might use Platform Coverage as a high-level planning tool. For example, your Platform Coverage list might appear as follows:

- Operating System: Windows XP Professional, SUSE Linux, AIX, Solaris, Mac OS X
- Database: DB2, Oracle
- Application Server: Tomcat, WAS
- CPU: x86
- Browser: Firefox, Internet Explorer

Use the Test Environments tab to list the actual combinations of environmental attributes that are available for testing on test lab machines. The Test Environments list might appear as follows:

- Test Environment 1 = Windows XP, Oracle
- Test Environment 2 = Mac OS, Firefox
- Test Environment 3 = Windows XP, DB2

It is not a requirement that you specify platform coverage or test environments in a test plan. Nor is a test plan required for test execution. However, if you specify test environments in your test plan, the test execution wizard will use those attributes when creating the test case execution records for the test cases. If you choose not to define test environments in your test plan, or you do not have a test plan, a complete list of attributes will be available during test execution.

## 4.6.2 Defining Platform Coverage

In the Test Environments section of the test plan, you can define a list of the software and hardware platforms that you plan to cover.

### About this task

In the Test Environments section of the test plan, use the Platform Coverage tab to create a non-binding list of platforms that you plan to cover. You can then use that list to generate the actual test environments that you need to run your tests.

### Procedure

To list the platforms that you plan to cover:

1. Open the Test Environments section of a test plan.
2. Click the Platform Coverage tab.
3. Click the Manage the platforms to be covered icon (Add platforms to be covered icon) to open the Available Environment Options window.
4. Select the environment type from the list.

The default environment types include Application Server, Browsers, Database, Management Agent, Test Adapter, Operating System, and CPU.

5. Move the specific application servers, browsers, CPUs, and so on from the Available column to the Selected column.

Note The options that are available in the dialog are defined as part of a test plan environment type. For more information about creating or modifying test plan environment types, see Test plan environment types.

6. Click OK to add the selected platforms to the test plan.
7. Click Save to save the test plan.

### Results

The list of covered platforms can be viewed in the test plan and are used as input if you choose to generate test environments automatically.



### 4.6.3 Generating New Test Environments

In the Test Environments section of a test plan, you can generate new test environments automatically and add them to the test plan. You can also add existing test environments to the plan.

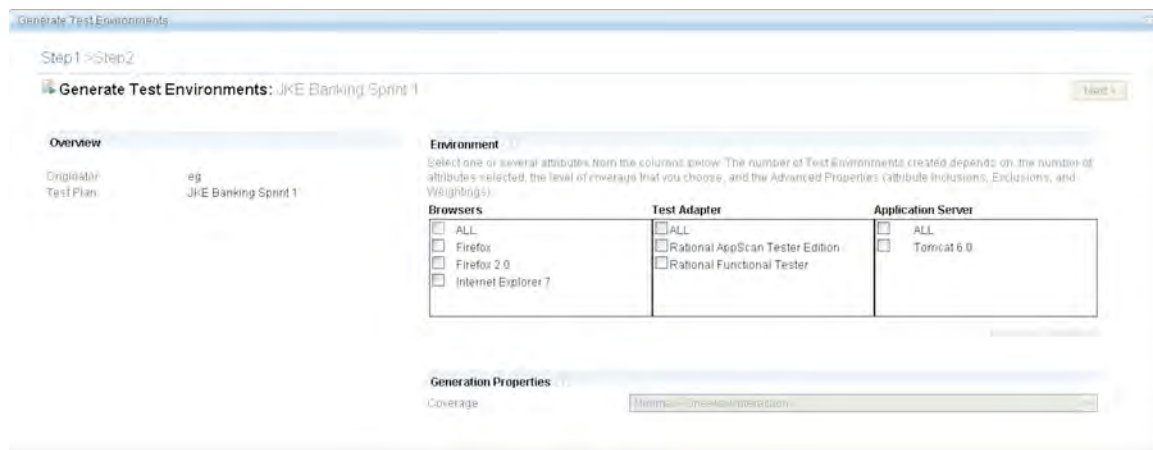
#### Before you begin

Before you generate test environments, you must define your platform coverage. The platforms that you have selected to be covered are used as input to the Generate Test Environments wizard.

#### Procedure

To generate new test environments in a test plan:

1. Open the Test Environments section of a test plan.
2. Click the Test Environment tab.
3. Click the Generate New Test Environments icon (Generate new test environments) to start the Generate Test Environments wizard.
4. Select one or several environment attributes from each column.



The number of test environments created depends on the number of attributes selected, the level of coverage that you choose, and the Advanced Properties (Inclusions, Exclusions, and Weightings) that you select.

Note Advanced Properties may not be visible until you select at least two attributes from one of the columns.

5. Select the level of coverage that you want. Use this setting and the Advanced Properties setting to fine-tune the environments that will be generated.



- a) Choose Minimal to ensure that each selected attribute is covered at least once, with no attempt to cover specific combinations of attributes. For example, if you select one attribute from three of the columns, three environments are created, ensuring that each selected attribute is covered at least once.
- b) Choose Medium - pair-wise interaction to ensure that each combination of paired attributes is covered at least once.
- c) Choose Large - three-way interaction to ensure that each three-way combination of attributes is covered at least once.
- d) Choose All - all permutations to ensure that all combinations of attributes are covered at least once.

Note The coverage options only become available when you start selecting attributes. The greater the coverage that you want, the more attributes and columns you must select.

6. Click Advanced Properties to display a window with three tabs: Inclusions, Exclusions, and Weightings.

- a) Click Inclusions to specify the attribute combinations to always include, for example Internet Explorer 7.x running on Windows XP.
- b) Click Exclusions to specify the attribute combinations to explicitly exclude, for example Safari browser running on Windows XP.
- c) Click Weightings to set the weight or importance of each attribute relative to the other values for that attribute.

For example, you can assign greater weight to Windows XP than SUSE Linux to increase the likelihood that Windows XP will be included in the list of generated test environments.

To add an Inclusion or Exclusion, click the Add Inclusion/Add Exclusion icon (Add Inclusions/Add Exclusions, select the attribute and click OK.

To adjust a Weighting, simply move the slider.

7. Click Next. The wizard creates a preview of the Generated Test Environments.
8. If necessary, select any test environments that you want to remove and click Finish. The wizard generates the test environments according to the criteria you have selected.
9. Optionally, decide how you want to group the generated test environments by selecting one of the choices in the Group By list.
10. Click Save to save the newly-generated test environments in the test plan.

#### ***4.6.4 Adding Existing Test Environments to a Test Plan***

In the Test Environments section of a test plan you can add test environments that have already been created to the plan. The list of test environments shows the combinations of hardware and software platforms to be used in testing.



## Procedure

1. In an open test plan, click Test Environments.
2. Click the Test Environment tab.
3. Click the Add Existing Test Environments (Add) icon.
4. In the window that opens, select one or several test environments and click OK.

## Results

The environments are added to the test plan.

### 4.6.5 Test Environment Types

A test environment is the configuration of environment types for a particular test. For example, a test environment that uses Firefox, Windows XP, and Apache Tomcat is a configuration of these environment types: browser, operating system, and application server.

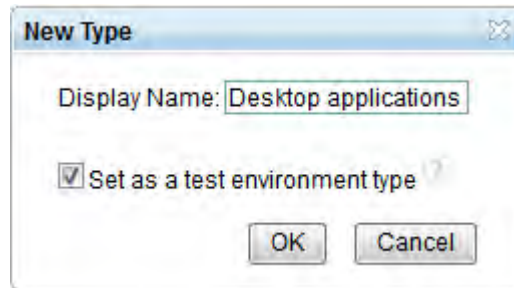
Each environment type can have one or more values. For example, the browser environment type might have Firefox, Internet Explorer, and Safari as its values. A test plan platform is the set of all of the environment types and their values for a test plan. The platform defines what the product supports and what the test team is testing.

Test environment types are shown in these areas:

- In the Create Test Environment editor, in the lab resource description
- In the Create Request editor, in the lab resource description
- In the Test Plan editor, in the Test Environments section
- In the Test Case editor, in the Test Case Execution Records section, in the Generate Test Case Execution Records wizard
- In the Test Suite editor, in the Test Suite Execution Records section, in the Generate Test Suite Execution Records wizard

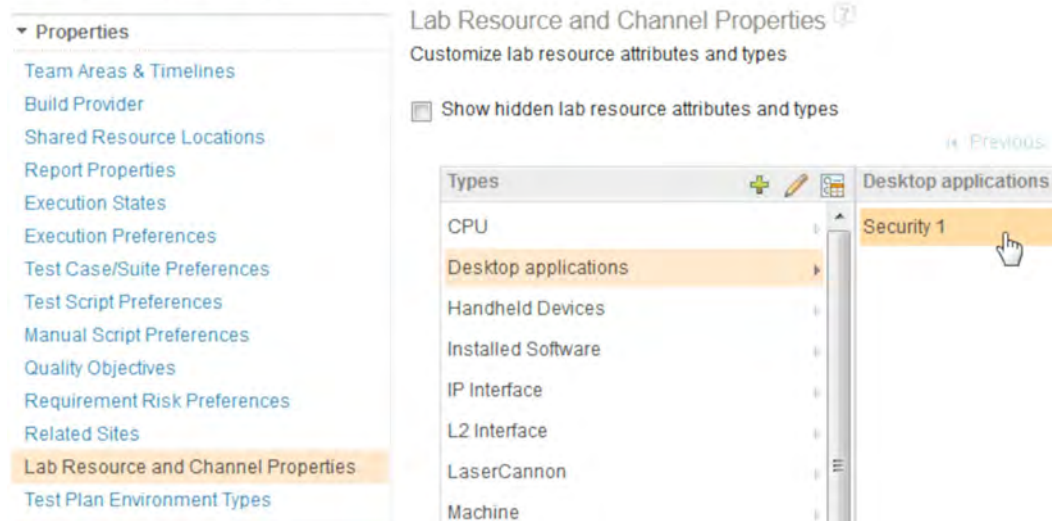
### Example: Defining an environment type and an environment type value

Security 1 is an in-house security application that your company provides to its employees as a desktop application. You want to define Security 1 as an environment type value, so you define "Desktop applications" as a type and Security 1 as a value of that type. When you create the type, you select Set as test environment type so that the new type is shown as an option whenever you define platform coverage and create test environments.



In this case, "Desktop applications" is shown as the heading of a group of test environment types, and Security 1 is shown as a value that you can select.

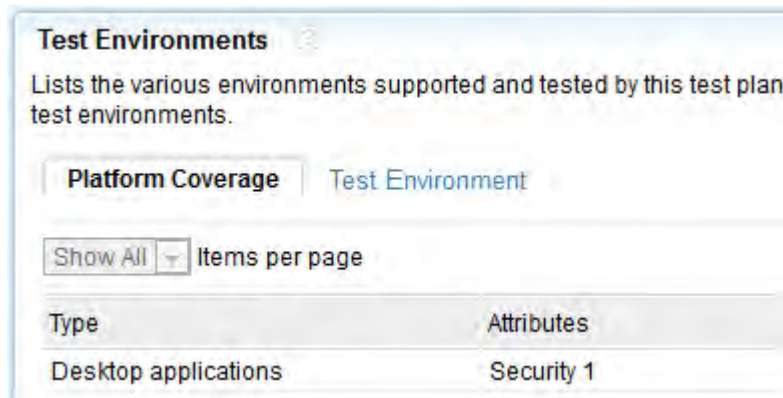
### Manage Project Properties



In your lab management test environment, you can also specify the new type and value as criteria of your machine.



Then, in the Test Environment section of your test plan, on the Platform Coverage tab, you can use "Desktop applications" as an environment type and Security 1 as an environment type value.



### Methods to create test environment types

Several environment types are included by default. You can also create test environment types:

- To create a test environment type that is associated with machines, while you create the type, select the Set as a test environment type check box. Selecting this option makes the new type available when you are defining platform coverage and creating test environments. The test environment types that you create are associated with the Machine type.
- To create a test environment type that is associated with any lab resource type, leave the Set as a test environment type check box clear. The new type is displayed wherever the type you associated it with is displayed. The new type is also displayed as the title of a set of test environment types, with its subtypes displayed as test environment types that you can select.

#### 4.6.6 Environment Values: Types of Machines

You can customize the values of test environment types so that they are associated with types of machines. When you create a test environment type, you can associate it with machines. Then, wherever the machine field is displayed, the environment type is displayed as a value that you can select.

### About this task

When you create a lab resource type, if you do not set it as a test environment type, you can later. The definition of the test environment type is based on the attributes of the machine type. Therefore, you must add a new attribute to the machine type to reference the

test environment type. To set a lab resource type as a test environment type, complete steps 7 - 12 in the following procedure.

## Procedure

1. Log in as a user who has permission to save and modify a project area. These users have that permission:
  - a) A user who has permission to save and modify a project area in the process configuration
  - b) A user who is an administrator of the project area
  - c) A user who has either the JazzProjectAdmins or JazzAdmins repository group permissions
2. In the upper-right portion of the banner, click the Admin menu Administration and click Manage Project Properties.
3. Click Lab Resource and Channel Properties. In the Types pane, click the Add New Type (Add) icon.
4. In the window that opens, type a name for the new lab resource type, and make sure that the Set as a test environment type check box is selected. When that check box is selected, the new type is available when you are defining platform coverage and creating test environments. Click OK. The new lab resource type is displayed in the list of types.
5. Define at least one value for the type:
  - a) Select the new lab resource type. A new column is displayed to the right of the type.
  - b) Click the Add New Type icon. A default value is displayed. The value has the same name as the type, with 1 appended to it.
  - c) Click the value name and type a descriptive name.
  - d) If needed, create one or more other values. When you are finished, click Save.
6. Log out, and then log in again. The test environment type that you created is displayed in these places:
  - a) Test environment sections in test plans
  - b) The wizard for creating test case execution records in test cases
  - c) The wizard for creating test suite execution records in test suites
  - d) The Create Test Environments lab management editor
7. Open the Manage Project Properties page and click Lab Resource and Channel Properties.
8. In the list of types, hover over Machine, and from the menu that is displayed, click Edit type attributes.



9. Click the Add icon. In the new row, enter an attribute name. For the data type, select Type. The Value column shows the type that you are configuring. Click OK, and save and reload the page.
10. Verify that you defined at least one value for the new type. If you do not define a value, the type is not displayed when you are defining platform coverage and creating test environments.
11. On the Manage Project Properties page, click Test Plan Environment Types.
12. In the list of types, click the new type, and then click the Set type icon. The type is established as a test plan environment type. Click OK, and save and reload the page. The type is now available as an environment type in test plans and when you generate test case execution records.

## What to do next

If you need to remove a test environment type, open the Test Plan Environment Types section. Hover over the type name, and from the menu that is displayed, click Remove Type. If you need to restore a removed test environment type, use the same menu that you used to remove the type, and click Set Type.

### 4.6.7 Test Environment Values: Any Lab Resource Type

You can customize the values of test environment types so that they are associated with any lab resource types. When you create a test environment type, you can associate it with a lab resource type. Then, wherever the field of the lab resource type is displayed, the environment type is also displayed as a value that you can select.

## About this task

For example, if you created an environment type and associated it with the Operating System type, the new environment type and its values would be displayed wherever the operating system is displayed.

## Procedure

1. Log in as a user who has permission to save and modify a project area. These users have that permission:
  - a) A user who has permission to save and modify a project area in the process configuration
  - b) A user who is an administrator of the project area
  - c) A user who has either the JazzProjectAdmins or JazzAdmins repository group permissions

2. In the upper-right portion of the banner, click the Admin menu Administration and click Manage Project Properties.
3. Click Lab Resource and Channel Properties. In the Types pane, click the Add New Type (Add) icon.
4. In the window that opens, type a name for the new lab resource type, and make sure that the Set as a test environment type check box is selected. When that check box is selected, the new type is available when you are defining platform coverage and creating test environments. Click OK. The new lab resource type is displayed in the list of types.
5. Define at least one value for the type:
  - a) Select the new lab resource type. A new column is displayed to the right of the type.
  - b) Click the Add New Type icon. A default value is displayed. The value has the same name as the type, with 1 appended to it.
  - c) Click the value name and type a descriptive name.
  - d) If needed, create one or more other values. When you are finished, click Save.
6. Log out, and then log in again.
7. Return to the Manage Project Properties page, and then click Test Plan's Environment Types.
8. In the list of types, click the type that you created, and then click the Set Type icon Set type.
9. In the Set Type Identifier window, rename the type if needed, and then click OK. Click Save.
10. Log out and log back in again. The test environment type that you created is displayed in these places:
  - a) Wherever the type that you associated it with is displayed
  - b) Test environment sections in test plans and test cases
  - c) The wizard for creating test case execution records in test cases
  - d) The wizard for creating test suite execution records in test suites
  - e) The Create Test Environments lab management editor

Remember When you create a lab resource type, if you do not set it as a test environment type, you can later. The test environment type's definition is based on the machine type's attributes. A new attribute must be added to the machine type to reference the new type.

- a) Click Admin > Manage Project Properties, and then click Lab Resource and Channel Properties.
- b) In the list of types, hover over the lab resource type that you want to associate the new type with, and from the menu that is displayed, click Edit type attributes.



- c) Click the Add icon. In the new row, enter an attribute name. For the data type, select Type. The Value column shows the type that you are configuring. Click OK, and then save and reload the page.
- d) Verify that you defined at least one value for the new type. If you do not define a value, the type is not displayed when you are defining platform coverage and creating test environments.
- e) On the Manage Project Properties page, select Test Plan Environment Types.
- f) In the list of types, click the new type, and then click the Set type icon. The type is established as a test plan environment type. Click OK, and save and reload the page. The type is now available as an environment type in test plans and when you generate test case execution records.

## What to do next

If you need to remove a test environment type, open the Test Plan Environment Types section. Hover over the type name, and from the menu that is displayed, click Remove Type. If you need to restore a removed test environment type, use the same menu that you used to remove the type, and click Set Type.



## 4.6.8 Adding Quality Objectives

You can add any predefined or user-defined quality objectives to an individual test plan and adjust the objectives as needed during the testing cycle.

### About this task

By default, an approved test plan is no longer open for editing. If a user modifies and attempts to save an approved test plan, the following message is displayed: Do not allow saving of an "Approved" or "Retired" artifact. To change this behaviour, see [Allowing users to edit approved artifacts](#).

### Procedure

To add a quality objective to a test plan and track it during the testing cycle:

1. Add a quality objective to a test plan.
  - a) In an open test plan, click either Quality Objectives, Entry Criteria, or Exit Criteria.
  - b) Click the Add Row icon (Add table row).
  - c) Select a row from the table.
  - d) Click OK.
  - e) Click Save to save the test plan.
2. Track the quality objective at a later time.
  - a) Open either the Quality Objectives, Entry Criteria, or Exit Criteria sections of a test plan.
  - b) Refresh the predefined objectives with the most recent values by clicking the Evaluate Quality Objectives icon (Evaluate quality objectives). This causes Quality Management to gather the actual data associated with the objective and measure it against the expected value.
  - c) Refresh the user-defined objectives by typing a value.

Note You can also type values for predefined objectives, but anything that you type will be recalculated by the software.
  - d) Double-click the Status value in the objective's row to choose a status.
  - e) Optionally, enter a comment by double-clicking on the objective's Comment cell.
  - f) Click Save to save the test plan.

## 4.6.9 Using Shared Resources

Before you can browse to external test scripts that reside on shared network resources, an administrator must make the resources available. After the resources are available, you can associate the test plan with these shared network resources.



## Procedure

To associate the test plan with shared network resources:

1. Click the Resources section of the test plan.
3. Click the Add Resource Location icon (Add).
4. Select the script type.
5. Select the shared location.
6. Optionally, select Include Plan and/or Include Test Iteration.
7. Click OK.

Note When you choose to include the plan and iteration, you append variables to the project path. For example, the path might be D:\RFT workspace\Project1\(\$Test-Plan)\(\$Test-Iteration). During a test run, the variables, (\$Test-Plan) and (\$Test-Iteration), are replaced with the names of the test plan and iteration in which the test case execution record or test suite execution record is associated with, allowing versioning support. For example, if you use the same script for a different iteration, you can edit the test case execution record or test suite execution record with the new iteration name. When you run the script, it will be added to the new iteration folder.

### 4.6.10 Copying Portions of Existing Documents

If you have content in external documents that you would like to reuse, you can paste the content into the appropriate test plan, test case, or test suite section.

## Procedure

To copy and paste existing content into a test plan, test case, test suite, or any rich-text editor:

1. Open the external document that contains the text you want to reuse.
2. Select and copy the text.
3. Open any test plan or test case section that includes a rich-text editor.
4. Click Edit to open the rich-text editor.
5. Using your keyboard, press Ctrl+V to paste the copied text into the section.
6. Click the Validate Content icon (Rich Text Editor cleanup) to validate and clean up the text. Then, click OK to accept the cleanup or Cancel to reject.
7. Use the rich-text editor to modify the text. For example, you can:
  - a) Change the appearance of the text by adding bold, italics, underline, and strikethrough.
  - b) Change the size, foreground color, and background color of text.

- c) Add numbered and bulleted lists.
  - d) Add tables.
  - e) Add indents.
  - f) Justify the text.
  - g) Insert images.
8. Click Save.

## ***4.6.11 Reviewing Test Plans***

### **Assigning Reviewers and Approvers**

As part of a formal test plan, test case, or test suite review process, you must assign team members to be reviewers or approvers.

#### **Before you begin**

Only users who have been granted permission and are a member of the related project can add reviewers and approvers. Permissions are controlled in Jazz™ Project Administration. In addition, the users whom you assign as reviewers and approvers must be members of the project to which the test plan, test case, or test suite belongs.

#### **Procedure**

To assign reviewers and approvers:

1. Open a test plan, test case, or test suite.
2. In the sections table of contents, click the Formal Review section.
3. In the New drop-down list, select a review type: Review or Approval.
4. Set a completion due date and a description.
5. Add one or more approvers.
  - a) Click Add approver.
  - b) Type a name or the first few characters of a name to load the complete list of users in the project. You can use \* and ? as wild card characters.
  - c) In the Matching users field, click the user name of the person to assign as a reviewer or approver.
  - d) To add that single user, click Add & Close; to add multiple users, click Add after each addition.



6. When the test plan, test case, or test suite is ready to be reviewed, on the overview page in the Action list, select Ready for review. This changes the artifact state from Draft to Under Review and generates email notifications to the reviewers and approvers.
7. Click Save.

## Results

Each team member who is identified receives a Task-Review work item to review or approve the test plan, test case, or test suite. The status of each person's review is listed as Pending. After each member completes the work item, the owner can click History to track the review history.

## Reviewing Test Artifacts

When you are assigned to review or approve a test artifact, such as a test plan, test case, or test suite, you receive a notification in the My Reviews widget on your Home page. You can open the artifact and then review, approve, or reject the artifact. You can also review the artifact while browsing a list of test artifacts.

## Procedure

1. Go to your Home page and look for the new task in the My Reviews widget.
2. Open the test artifact to review by clicking the name in the My Reviews widget. For an example of how to add the My Reviews widget, see the "Monitor review tasks" section in the tutorial, Lesson 2: Add a new requirement to the test plan.
3. When you finish your review, click the Formal Review section in the test plan, test case, or test suite.
4. If you are a reviewer, do the following:
  - a) Find your name in the list of reviewers and approvers.
  - b) Change the Status from Pending to Reviewed or Rejected.
  - c) Type your review comments.
  - d) Click Save.
  - e) If your role requires that you authenticate with the Jazz™ server for reviews and approvals of this artifact type, you will be required to provide your password.
5. If you are an approver, do the following:
  - a) Find your name in the list of reviewers and approvers.
  - b) Change the Status from Pending to Approved or Rejected.
  - c) Note When you reject the test plan or test case, it is expected that the requester will update the content and request another approval.

- d) Click Save.
  - e) If your role requires that you authenticate with the Jazz server for reviews and approvals of this artifact type, you will be required to E-Sign this action by providing your password.
6. Return to the My Reviews widget of your home page. If all reviews and approvals are complete for the artifact under review, the task is removed from your My Reviews widget.
7. To review test artifacts while browsing a list of test artifacts:
- a) From a list of test artifacts, select one or more test artifacts.
  - b) Click the Action Menu (Edit) for any of the selected test artifacts and click Review Test artifact type to mark the selected test cases as reviewed or rejected

## Tracking Reviews

You can manage the review process by keeping track of the completed and outstanding reviews and approvals.

### Procedure

To track reviews:

1. Open a test artifact that is under review.
2. Open the Formal Review section to display the result of each review, reviewer comments, and outstanding review requests.
3. Click History to see the review history of the test artifact.



## Module Five

---

### *Test Artifacts*

*It is not a test that finds a bug but it is a human that finds a bug  
and a test plays a role in helping the human find it.*  
Pradeep Soundararajan

*Correctness is clearly the prime quality. If a system does not do  
what it is supposed to do, then everything else about it matters  
little.*  
Bertrand Meyer

*We are stuck with technology when all we want is stuff that works.*  
Douglas Adams

*Testing is a skill. While this may come as a surprise to some  
people it is a simple fact.*  
Graham Fewster



## **5.1 Introduction.**

This section consists of more of the detailed instructions on some of the topics covered in class. Since the RQM product is vast and complex, we will not be covering every operational detail in class. However, part of what I have tried to do is supplement the overview from class with some additional documentation.

Almost all of the material in this section is from the RQM documentation which may no longer be available except at some archive sites.



## 5.2 Developing Test Cases

A test case answers the question: "What am I going to test?" You develop test cases to define the things that you must validate to ensure that the system is working correctly and is built with a high level of quality. A test suite is a collection of test cases that are grouped for test execution purposes.

### ***Test cases***

A typical use of test case might be to use the same test script for testing multiple configurations. For instance, if you want to test a login script on three different browsers, such as Firefox, Internet Explorer, and Safari, you can create three different test case execution records in that test case. In a test case that is called Test Browsers you might include three testing scenarios:

1. Test case execution record 1: Firefox and log-in test script
2. Test case execution record 2: Internet Explorer and log-in test script
3. Test case execution record 3: Safari and log-in test script

### ***Test suites***

If each test case represents a piece of a scenario, such as the elements that simulate a completing a transaction, use a test suite. For instance, a test suite might contain four test cases, each with a separate test script:

1. Test case 1: Login
2. Test case 2: Add New Products
3. Test case 3: Checkout
4. Test case 4: Logout

Test suites can identify gaps in a testing effort where the successful completion of one test case must occur before you begin the next test case. For instance, you cannot add new products to a shopping cart before you successfully log in to the application. When you run a test suite in sequential mode, you can choose to stop the suite execution if a single test case does not pass. Stopping the execution is useful if running a test case in a test suite depends on the success of previous test cases.

Test suites are also useful for the following types of tests:

1. **Build verification tests:** A collection of test cases that perform a basic validation of most the functional areas in the product. The tests are executed after each product build and before the build is promoted for use by a larger audience.



2. **Smoke tests:** A collection of test cases that ensure basic product functionality. Typically, smoke tests are the first level of testing that is performed after changes are made to the system under test.
3. **End-to-End integration tests:** A collection of test cases that cross product boundaries and ensure that the integration points between products are exercised and validated.
4. **Functional verification tests:** A collection of test cases that focus on a specific product function. Executing this type of test with a test suite ensures that several aspects of a specific feature are tested.
5. **Regression tests:** A collection of test cases that are used to make a regression pass over functional product areas.

### ***Test scripts***

Each test case is typically associated with a test script, although you can run a test with no associated test script.

A test script is a manual or automated script that contains the instructions for implementing a test case. You can write manual test scripts to be run by a human tester, or you can automate some or all of the instructions in the test script. You can also associate automated functional test scripts, performance test scripts, and security test scripts with a test case.

### ***Test cases with multiple test scripts***

You can also associate more than one test script with a single test case, but you should do this only if each test script associated with the test case provides a different way to test the scenario in the test case. For example, a test case might require several scripts to test the scenario in different test environments. Or, you might have both manual and automated scripts that can be used to exercise the test case.

Do not use test cases to group test scripts that exercise different aspects of a common function or you might see incorrect information about test results and attempted points. For example, if you run a test case for a specific test plan, iteration, and test environment and specify a different functional script each time, the last result of the test case execution record only captures the results of the last script run.

In some cases, you may also want to use test suites to group related test cases. For example, you can use a suite to run a set of automated regression test cases together or to sequentially run a set of test cases that comprise an end-to-end scenario. With a test suite, you can plan, initiate, and track the execution of the related test cases that make up the larger scenario that is tested by the test suite.

### ***Test case and test suite sections***

The list below identifies the default sections that are included in test cases and test suites.

Test case and test suite default sections

<b>Test Case</b>	<b>Test Suite</b>
Summary	Summary
Test Case Design	Test Suite Design
Formal Review	Formal Review
Development Items	Pre-Condition
Requirements Links	Post-Condition
Risk Assessment	Risk Assessment
Pre-Condition	Expected Results
Post-Condition	Test Cases
Expected Results	Test Suite Execution Records
Test Scripts	Attachments
Test Case Execution Records	Execution Variables
Attachments	
Execution Variables	

Each section includes its own editor. Some sections, consist of a rich-text editor with common formatting for features such as tables, fonts, bullets, and numbered lists.

Other test case sections provide links to these other test artifacts.

You can add your own sections and remove unneeded sections by using the Manage Sections feature.

### ***Weight and categories of test cases and test suites***

Test case and test suite categories are used in the Summary section of a test case or test suite.

You can use test case and test suite categories to help group your test cases or test suites logically. Later, when you can browse to list all your test cases or test suites, you can sort the list by using the categories.



Four default categories are provided: category, function, iteration, and theme. You can add your own categories, functions, iterations, and themes and types of categories.

Weight is a measurement of results not a measurement of time. You can use it flexibly, but for time measurements use the estimate and time spent.

### ***Custom attributes for test case and test suites***

If an administrator defined custom attributes for test cases or test suites, they are listed in the summary.

Like categories, custom attributes can be used as a grouping, sorting, and filtering mechanism to organize a test project into logical hierarchies of artifacts in your test project. Categories are restricted to text format; however, using custom attributes can include text, integer, and date formats. Custom attribute values are free-form so you can use any data that you need for test artifacts.

### ***Templates***

Test cases and test suites are based on templates. When you create a test case or test suite, you base it on a template. You can use the default templates, create new templates, and designate default templates.

Templates contain a collection of sections. You create a template by adding and removing provided sections or creating new sections. If the section names do not match what you are accustomed to, create new sections and add them to a template.

Each test organization can design their own custom test templates.

### ***Preconditions and postconditions***

Conditions provide information for the person who runs a test. Use preconditions to communicate requirements that must be met before the tester can run the test case or test suite. Use postconditions to communicate requirements that must be met after the tester runs the test case or test suite.

### ***Test cases, test suites, and test execution***

Test cases are integral to the test execution process. To run a test, you can do any of these things:

1. Run a test case.
2. Combine several test cases into a test suite, and run the test suite.
3. Generate test case execution records or test suite execution records to define the environmental attributes for running the test, and then run them.

## 5.2.1 Creating Test Cases

You can create a test case to describe what you are going to test, define preconditions and post-conditions, assign test steps, and record the results of running the test case.

### About this task

Typically, one or more test scripts and test case execution records are associated with a test case. Test case execution records are often used to run a test.

### Procedure

1. In the main menu, click Construction; then in the Create section, click Test Case. The new test case opens, with a table of contents on the left and an editor on the right.
2. At the top of the new test case window, enter a name for the new test case.
3. Select a test case template from the list, if available.
4. If team areas are enabled for the project area, select a team area from the list. Note: You must explicitly enable team areas in Manage Project Properties.
5. Select an owner and a priority for the test case.
6. Add a description of the test case.
7. Complete the test case Summary section or assign the Summary to another team member to complete.
8. To complete the test case Summary yourself:

- a) Define categories.

You can use categories to define a hierarchical organization of test plans, test cases, test suites, test scripts, and other test artifacts. You can also define sub-categories and other category relationships. You can use the default categories that are provided or create your own. To define categories, click the Manage Test Case Categories icon (Manage category).

- b) Type a number to estimate the amount of time required to complete the test case and select a time interval for the estimate. For example, if you type 1.5 in the Estimate field and select days for the interval, the estimate is represented as "1 day 4 hours."

Later on, you can open a test plan to view test execution status and see how closely the actual time spent matches the estimate.

- c) Assign a numerical value for the relative weight of the test case.

You can use weight as a means of tracking the relative difficulty or importance of the test case. It is provided as a measurement of results not a measurement of time. There is flexibility in how the weight value can be used, but for time measurements using the Estimate and Time Spent fields is recommended. When a test case is executed, the weight is calculated based on the success of the steps included in the test script. The calculated weight distribution for a result



can be adjusted to more accurately record the true outcome. Weight is used later on in one of the status bars that show execution progress in the test plan.

- d) Assign values to the custom attribute fields.

If an administrator defined any custom attributes for test cases, they display in the Attributes section of the Summary. The field values for custom attributes can be in the format of text, integer, or date. You can sort or filter your test cases using custom attributes in the same way that you use categories.

9. To assign the Summary section, or any section of the test case, to another team member to complete, click Work Item: Create and complete the fields in the form.
10. Optional: Click Manage Sections to add your own, customized test case sections or to remove unrequired sections.
11. Complete the remaining test case sections as needed:
  - a) Provide additional detail about the test case in the Test Case Design section.
  - b) Set up a formal review process for the test case.
  - c) Associate requirements with the test case.
  - d) Define test case preconditions and postconditions.
  - e) Define the expected test case results.
  - f) Identify risks associated with the test cases.
  - g) Associate a test script to the test case or create a manual test script.
  - h) Generate or create test case execution records for the test case.
  - i) Attach related documents to the test case.
  - j) Add execution variables to the test case.
12. Click Save.
13. At any point after you create the test case, you can add it to one or more test plans.
14. Optional: Click the Duplicate icon (Copy ) in the upper right corner of the test case to make a copy of the test case.
15. Optional: Click the Print View icon (Print view ) in the upper right corner of the test case to display a printable version of the test case.
16. Optional: Click the Set to Lock/Unlock icon (Change lock) in the upper right corner of the test case to lock or unlock the test case. Depending on how your project is configured, you might be required to sign this action electronically by providing your password and a comment. When the test case is locked, the editable fields are disabled for editing.

Note Permission to lock a test case depends on your role-based permissions.

To reverse the locking attribute, click the icon again.

17. Optional: Click the Print PDF icon (Export to PDF ) in the upper right corner of the test case to save the test case as a PDF.

## 5.2.2 Managing Test Artifact Sections

You can add and remove sections of individual test plans, test cases, test suites, and keywords. You can also configure the sections that are included in test plan, test case, and test suite templates.

### Procedure

1. Open a test plan, test case, test suite, or keyword.
2. From the list of test artifact sections, click Manage Sections to open the Manage Sections window.
3. Select Show all sections to display all the available sections.

Note You can set a user preference to display all sections of an artifact by default. To set the preference, click the User Profile (User Profile menu) icon in the upper-right portion of the banner, click My Preferences for Quality Management > Editor Table of Contents and then select Show all sections in the editor. With this preference set, you see all sections when you open an artifact. The sections are displayed in a closed state and you must double-click the sections to open them.

4. To add an established section, select the section from the Available Sections list, and move it to the Selected Sections list by clicking the Move right (Move right) icon.

Tip: Press Ctrl and click items to make multiple selections. You can also move all of the sections at once by clicking the Move all right (Move all right) icon.

5. To remove a section, move the section from the Selected Sections list to the Available Sections list by clicking the Move left (Move left) icon.
6. To reorder sections, use the Move up (Move up) and Move down (Move down) icons.
7. To add a customized section, click the Create Section (Create section) icon. The New Custom Section window opens.
  - a) In the Section Type field, select the type of content for the new template section, Rich Text Editor or Grid.
  - b) If you select Grid, click the Add (Add) icon to add a column and then type the column name.
  - c) If you select Rich Text Editor, you can enter boilerplate content into the rich text field so that it is included in all templates that use the section.
  - d) Click OK to close the New Custom Section window.

Note To delete a custom section, select the section, and click the Delete (Delete) icon. You cannot delete standard sections of a template.

8. Click OK to close the Manage Sections window.
9. Click Save to save the test plan, test case, or test suite.

### What to do next



You can further customize your test artifacts by editing the name and description of sections. For example, you can change the name of the Summary section in a test plan to Overview, or change both the name and description of a custom section. The ability to edit section names and descriptions is controlled by the Edit Section Header permission. To change the description of a standard section (for example, to provide more detailed information about the categories and attributes in the Summary section), you must modify or create a template.

### 5.2.3 Importing Test Cases

You can import test cases that are saved as local XML files directly into your repository, where they are then available on the Browse Test Cases page.

#### Before you begin

Ensure that the local XML file to import adheres to the XML schema documentation.

The following example of a basic test-case XML file that adheres to the schema:

```
<?xml version="1.0"?>
<testcase xmlns="http://jazz.net/xmlns/alm/qm/v0.1/">
  <title xmlns="http://purl.org/dc/elements/1.1/">New Customer Order Test Case</title>
  <description xmlns="http://purl.org/dc/elements/1.1/">Test the ordering
  functionality of the Classics Java application.</description>
  <updated xmlns="http://jazz.net/xmlns/alm/v0.1/">2008-08-23T01:00:21.437Z</updated>
  <state xmlns="http://jazz.net/xmlns/alm/v0.1/">com.ibm.rqm.planning.common.new</state>
  <creator xmlns="http://purl.org/dc/elements/1.1/">mary</creator>
  <owner xmlns="http://jazz.net/xmlns/alm/v0.1/">unassigned</owner>
  <regressionTest>false</regressionTest>
  <weight>20</weight>
  <category term="Category" value="Web UI"/>
  <category term="Function" value="Execution"/>
  <category term="Theme" value="Functionality"/>
</testcase>
```

#### Procedure

1. In the main menu, click Construction > Import > Test Cases.
2. Select Import from a local XML file, and click Browse.
3. Locate the file, and click Open.
4. Click Import to import the test case into the repository.



## 5.2.4 Adding New Test Cases

As an alternative to creating independent test cases, you can create new test cases in the context of an existing test plan. A test plan describes the scope of the overall test effort and provides a record of the test planning process. Test plans also allow you to manage other test artifacts, such as test cases and requirements.

### Procedure

1. Open an existing test plan or create a test plan.
2. From the test plan table of contents, click Test Cases.
3. Click the Add New Test Case icon (Add new test case) to open the New Test Case window.
4. Type a name for the new test case.
5. Type a description of the test case.
6. Select a test case template.
7. Type a number for the estimate of the time required to complete the test case and select a time interval for the estimate. For example, if you type 1.5 in the Estimate field and select days for the interval, the estimate is represented as "1 day 4 hours."
8. Assign a numerical value for the relative weight of the test case.

How you assign the weight value can be determined using different criteria, such as importance to test effort or the relative complexity of the test case. For example, a test case that is assigned a weight of 100, could be twice as important or complex as a test case that has an assigned weight of 50. The weight should not be used as a measurement of time. If you are intending to measure the time that is assumed to be required for the test case, then use the Estimate field. When a test case is executed, the weight (points) is calculated based on the success of the steps included in the test script. The points for a result can be adjusted to more accurately record the true outcome. Weight is used later on in one of the status bars that show execution progress in the test plan.

9. Select an owner and a priority for the new test case.
10. Select a Category, Function, Iteration, and Theme.
11. You can use these attributes to organize your test cases into various groupings that can later be sorted. Your team is free to define these attributes any way that makes sense.
12. Click OK.



## Results

The new test case is created and added to the test plan.

## What to do next

Now, you can edit the test case to modify the priority, state, and other attributes of the test case. Before you can run the new test case, you must add a test script to the test case. Open the test case and add a script in the Test Scripts section.

### 5.2.5 Adding New Test Cases

As an alternative to creating independent test cases, you can create new test cases in the context of an existing test plan. A test plan describes the scope of the overall test effort and provides a record of the test planning process. Test plans also allow you to manage other test artifacts, such as test cases and requirements.

## Procedure

1. Open an existing test plan or create a test plan.
2. From the test plan table of contents, click Test Cases.
3. Click the Add New Test Case icon (Add new test case) to open the New Test Case window.
4. Type a name for the new test case.
5. Type a description of the test case.
6. Select a test case template.
7. Type a number for the estimate of the time required to complete the test case and select a time interval for the estimate. For example, if you type 1.5 in the Estimate field and select days for the interval, the estimate is represented as "1 day 4 hours."
8. Assign a numerical value for the relative weight of the test case.

How you assign the weight value can be determined using different criteria, such as importance to test effort or the relative complexity of the test case. For example, a test case that is assigned a weight of 100, could be twice as important or complex as a test case that has an assigned weight of 50. The weight should not be used as a measurement of time. If you are intending to measure the time that is assumed to be required for the test case, then use the Estimate field. When a test case is executed, the weight (points) is calculated based on the success of the steps included in the test script. The points for a result can be adjusted to more accurately record the true outcome. Weight is used later on in one of the status bars that show execution progress in the test plan.

9. Select an owner and a priority for the new test case.

10. Select a Category, Function, Iteration, and Theme.
11. You can use these attributes to organize your test cases into various groupings that can later be sorted. Your team is free to define these attributes any way that makes sense.
12. Click OK.

## Results

The new test case is created and added to the test plan.

## What to do next

Now, you can edit the test case to modify the priority, state, and other attributes of the test case.

Before you can run the new test case, you must add a test script to the test case. Open the test case and add a script in the Test Scripts section.

## 5.2.6 Adding Existing Test Cases

You can add any existing test cases to a test plan.

## Procedure

1. Open an existing test plan or create a new one.
2. From the test plan table of contents, click Test Cases.
3. Click the Add Existing Test Cases icon (Add) to open the Add Test Cases window.
4. Select the test cases that you want to add to the test plan. If you have a large number of test cases, you can use the filtering capabilities to narrow down the list.
5. Click OK.
6. Save the changes to the test plan.

## Results

The test cases are added to the test plan.



### 5.2.7 Adding Existing Test Scripts to a Test Case

When you add an existing test script to a test case, it is listed in the Test Scripts section of the test case.

#### Procedure

1. From an open test case, click the Test Scripts section, and then click the Add Existing Test Scripts icon (Add). The Add Test Scripts window opens.
2. From the list of test scripts, select the test scripts to add, and then click OK.

To show a preview of individual test script details, hover your cursor over the test script ID or name. If you have a large number of test scripts, you can use the filtering capabilities to narrow down the list.

3. Save the changes to the test case.

#### Results

The test scripts are added to the Test Scripts section of the test case.

### 5.2.8 Adding a New Test Script to a Test Case

In the Test Scripts section of a test case, you can create a new test script and associate it with the test case.

#### Procedure

1. From an open test case, click the Test Scripts section, and then click the Add New Test Script icon (Add). The New Test Script window opens.
2. In the Name field, type a name for the new test script.
3. Optionally, enter a description of the new test script in the Description field.
4. Click OK. The new test scripts is added to the Test Scripts section of the test case.
5. Save the changes to the test case.

#### Results

You can click the new test script name to open the script in the Manual Test Editor.

## ***5.2.9 Estimating the Weight of Individual Test Cases***

You can provide a more detailed estimate of the test execution effort by adding a weight value to each test case. Test case execution records inherit the weight of their associated test case.

### **About this task**

Each test case can be assigned a weight value. How you assign the weight value can be determined using different criteria, such as importance to test effort or the relative complexity of the test case. For example, a test case that is assigned a weight of 100, could be twice as important or complex as a test case that has an assigned weight of 50. The weight should not be used as a measurement of time. If you are intending to measure the time that is assumed to be required for the test case, then use the Estimate field.

This detailed sizing information is used as input to several execution status reports, such as the TER Status Counts and Execution Status Using Weight reports. By assigning various weights to each test case, you can run accurate reports that consider both the absolute number of test cases that are run and an accurate measurement of the progress made against each test case.

### **Procedure**

To add weight to a test case:

1. Open the Summary section of a test case for editing.
2. Type a value for the relative weight of the test case as compared with other test cases.
3. Click Save.



## 5.3 Linking and Managing Requirements

In the Quality Management application, you can associate test cases with requirements in the Requirements Management application. Then you can include the test cases in test plans to ensure thorough test coverage of product requirements.

### About this task

Limitation: When you create a link to external artifacts in a CLM workbench source application, a link is automatically created back to the source artifact in the target application, which is called a back link. However, a state of synchronization between bidirectional links and the traceability between source and target artifacts are not guaranteed.

### 5.3.1 Associating Test Cases with Requirements

#### Procedure

1. Open or create a test case, and in the table of contents, click Requirement Links.
2. Click the Add new links icon (Add). The Requirement links page opens.
3. The requirement projects are displayed on the right. Select the requirement to associate with the test case. To select more than one requirement, press and hold the Ctrl key and select the requirements.
4. Click OK. The requirements are displayed in the Requirement Links section of the test case.
5. Click Save to save the changes that you made to the test case.

### 5.3.2 Generating Test Cases

Generating test cases for requirements in requirement collections in the Requirement Management Application

After you have associated requirement collections with test plans, you can generate test cases for the requirements in the collections. This ensures that you have thorough testing coverage and traceability for each requirement.

### About this task

Before you can generate test cases for requirements in requirement collections, you must associate requirement collections to test plans.

#### Procedure

1. Open the test plan with which you associated the requirements collection, and go to the Requirement Collection Links section.
2. Select the check box next to the requirement collection for which to generate test cases.
3. Click the Reconcile Requirements in Collections icon (Reconcile Requirements in Collections). The Reconcile Requirements wizard opens. All requirements that the requirement collection contains and do not have test coverage in the current test plan are displayed.
4. In the Pending Actions section, the Reconcile Test Coverage action is displayed. To add test coverage for new requirements, complete the following steps:
  - a) the Expand/Collapse icon (Expand/Collapse). The Action menu opens.
  - b) In the Action menu, do one of the following:

If the selected requirements already have been linked to existing test cases, to add the test cases to the current test plan, click Add Test Case to Test Plan.

To generate new test cases for each selected requirement, from the Action menu, click Generate [n] Test Cases, where n is the number of test cases to generate; then complete the fields in the New Test Case window and click OK.

To generate a single new test case for all selected requirements, from the Action menu, click Generate Test Case; then complete the fields in the New Test Case window and click OK.
  - c) Click Next.

If there are additional pending actions, the Reconcile Requirements window displays the actions and the list of requirements associated with each action: Reconcile Removed Items, Reconcile Deleted Items, and Reconcile Updated Items.
5. Click Finish, and then save the test plan.
6. In the table of contents, click Test Cases. In the Test Cases section of the test plan, all the test cases that were created for the Requirements Management requirements collection are displayed.



### 5.3.3 Reconciling Test Cases and Requirements

When requirements are changed or are deleted in the requirements source application, you can reconcile the associated test cases with the requirements to show the latest requirement status. You can then identify as suspect test cases that contain requirements that are changed or are deleted. Identifying suspect artifacts helps you update test plans and test cases quickly and accurately by responding to requirement changes. When a test case is marked as suspect, you can open a quality task to update the test case.

#### Procedure

1. Open a test case that has one or more requirements associated with it.
2. Open the Requirement Links section.
3. Select one or more requirements; then click the Reconcile Requirement icon Reconcile Requirements in Collections and Reconcile Requirement icon. The Reconcile Requirement window opens. In the Actions section, one or more of the following actions are displayed: Reconcile Updated Items or Reconcile Deleted Items.
4. To reconcile test cases with updated requirements, complete the following steps:
  - a) In the Actions section, select Reconcile Updated Items. The requirements associated with the action are displayed in the table.
  - b) Review the changes to the listed requirements.
  - c) Select one or more requirements to apply an action to associated test cases.
  - d) Click an action icon.

Ignore: No action will be applied to associated test cases for this reconcile operation. The requirement will be displayed as modified in future reconcile operations.

Clear Suspicion: If associated test cases are listed as suspect, their status will be updated to clear this state.

Mark Suspect: Associated test cases will be marked as suspect.

For selected requirements, the pending operation is displayed in the Action column. Repeat steps 4.c through 4.d until each requirement has a pending operation listed under Action.

5. For any modified requirements, to update the associated test case, select Create New Quality Task. You will be prompted to create a quality task for each test case that is associated with the selected requirements when you click Finish.

**Note** You can create a new quality task only for the requirements for which you chose to apply Mark Suspect. The Create a new Quality Task option is disabled when Clear Suspicion or Ignore is applied.

6. Click Finish.



7. If you chose to create a new quality task for modified test cases, a window opens prompting you to create the quality tasks.
  - a) Click the Create New Quality Task button.
  - b) Select the task type.
  - c) In the Summary field, enter a brief headline that identifies the work item.
  - d) In the Filed Against field, select a category that identifies the component or functional area that the quality task belongs to.
  - e) Assign an owner to the task.
  - f) Set a due date for the task to be completed.
  - g) Enter a description of the quality task and click OK.

A quality task is created in the Change and Configuration Management application. The task appears in the My Tasks widget of the personal dashboard for the owner that you assigned the quality task to. The Links section of the quality task shows a related test case link for the test case associated with this task.

8. If you receive a message prompting you to close and reopen the test case to show the latest changes, do so.

### What to do next

After the quality task is resolved, you can change the suspect status of the test case so that the test case is no longer marked as suspect.



### 5.3.4 Requirement Reconcile Operation Status Types

When you reconcile a requirement or requirement collection, requirements that have a changed status since the last reconcile operation are displayed with their new status. See the table below to understand the different status types.

Requirement status	Description	Available actions
Deleted	The requirement has been deleted from the project in the Requirement Management application.	<ul style="list-style-type: none"> <li>• Ignore</li> <li>• Mark Suspect</li> </ul>
Modified	The requirement has changed since the last reconciliation.	<ul style="list-style-type: none"> <li>• Ignore</li> <li>• Clear Suspicion</li> <li>• Mark Suspect</li> </ul>
New	The requirement has not been added to the Quality Management application.	You can select requirements to generate new test cases that add test coverage for the requirements.
Others	The requirement cannot be accessed, either due to a user permissions issue or because of a connectivity issue.	Ignore
Removed	The requirement has been removed from a collection, but has not been deleted from the project.	<ul style="list-style-type: none"> <li>• Ignore</li> <li>• Mark Suspect</li> </ul>

### Action descriptions

The following list defines the actions that you can apply to requirements.

#### Ignore

No action will be applied to associated test cases for this reconcile operation. The requirement will be displayed as modified in future reconcile operations.

#### Clear Suspicion

If associated test cases are listed as suspect, their status will be updated to clear this state.

#### Mark Suspect

Associated test cases will be marked as suspect.

### ***5.3.5 Configuring Data Record Selection Criteria***

Test scripts can have test data. You can define the selection criteria for data records that are provided by the test scripts associated with a test case. Selection criteria that you define at the test case level overrides the selection criteria which has been defined at the test script level.

#### **Procedure**

1. In the Test Scripts section of a test case, select one or more test scripts in the list; then click the Edit icon (Edit) next to a selected test script.
2. To override the test script-level data record selection criteria, complete the following steps:
  - a) In the pop-up menu, select Edit Data Record.
  - b) Under Record Selection Criteria, select a column name; then set a filter value.
  - c) To verify the selection, click the Show button; then click OK.
3. To reset the test case-level selection criteria, in the pop-up menu, select Reset Data Record to script level.



## 5.4 Developing Test Suites

You can create a test suite to group a collection of test cases for test execution purposes. You can associate new or existing test cases with the test suite.

### Procedure

1. In the main menu, click Construction > Create > Test Suite. The new test suite opens with a table of contents on the left side of the screen and an editor on the right side.
2. At the top of the new test suite window, enter a name for the new test suite.
3. Select a test suite template from the list, if available.
4. If team areas are enabled for the project area, select a team area from the list.

Note You must explicitly enable team areas in Manage Project Properties.

5. Select an owner and a priority for the test suite.
6. Add a description of the test suite.
7. Complete the test suite Summary section or assign the Summary to another team member to complete.
8. Optional: To complete the test suite Summary yourself:
  - a) Define categories. You can use categories to define a hierarchical organization of test plans, test cases, test suites, test scripts, and other test artifacts. You can also define subcategories and other category relationships. You can use the default categories that are provided or create your own. To define categories, click the Manage Test Suite Categories icon (Manage category).
  - b) Type a number for the estimate of the work effort that the test suite represents, and select a time interval for the estimate. For example, if you type 1.5 in the Estimate field and select days for the interval, the estimate is represented as 1 day 4 hours. Later, you can open a test plan to view test execution status and see how closely the actual time spent matches the estimate.
  - c) Assign a numerical value for the relative weight of the test suite.

You can use weight as a means of tracking the relative difficulty or importance of the test suite. It is provided as a measurement of results not a measurement of time. There is flexibility in how you determine the weight value for the test suite, but for time measurements use the Estimate and Time Spent fields. The results of the test suite will include weight (points) calculated based on the success of the test cases included in the test suite. The points for a result can be adjusted to more accurately record the true outcome. Weight is used later on in one of the status bars that show execution progress in the test plan.

The Calculate weight button is used to calculate the weight of the suite based on the test cases the test suite contains. After adding or removing test cases to the test suite, review the test suite weight and if necessary click the Calculate weight button to update the weight.

- d) Assign values to the custom attribute fields. If an administrator defined custom attributes for test suites, the attributes are displayed in the Attributes section of the Summary. The field values for custom attributes can be in the format of text, integer, or date. You can sort or filter your test suites using custom attributes in the same way that you use categories.
9. Optional: To assign completion of the Summary section, or any section of the test suite, to another team member, click Work Item: Create, and complete the fields in the form.
  10. Optional: Click Manage Sections to add your own, customized test suite sections or to remove unrequired sections.
  11. Complete the remaining test suite sections as needed:
    - a) Provide additional detail about the test suite in the Test Suite Design section.
    - b) Set up a formal review process for the test suite.
    - c) Define test suite preconditions and postconditions.
    - d) Define the expected test suite results.
    - e) Identify risks that are associated with the test suite.
    - f) Associate test cases with the test suite. Note Select a test case and click Move Up (Move test case up) or Move Down (Move test case down) in the action menu to reorder the list of test cases as required.
    - g) Associate a test cell with the test suite by selecting the Select Test Cell icon (Test cell) in the Test Cases section. This is only applicable to test suites containing automated test cases. Automated test cases contained within the suite run on the specified test cell.
    - h) Add execution variables to the test suite.
    - i) Generate or create test suite execution records for the test suite.
    - k) Attach related documents to the test suite.
  12. Click Save.
  13. At any point after you create the test suite, you can add it to one or more test plans.
  14. Optional: Click the Duplicate icon (Copy ) in the upper-right corner of the test suite to copy the test suite.
  15. Optional: Click the Print View icon (Print view ) in the upper-right corner of the test suite to display a printable version of the test suite.
  16. Optional: Click the Set to Lock/Unlock icon (Change lock) in the upper right corner of the test suite to lock or unlock the test suite. Depending on how your project is configured, you might be required to sign this action electronically by providing your password and a comment. When the test suite is locked, the editable fields are disabled for editing. Note Permission to lock a test suite depends on your role-based permissions.

To reverse the locking attribute, click the icon again.



17. Optional: Change the execution properties of a test case by selecting the test case and clicking the Edit icon (Edit) in the Test Script, Test Environment, or Execution Owner cells. To update the execution properties of multiple test cases, select the test cases and use the Change icon (Bulk change) in the column header.

18. To run test cases one after the other, select Run this suite in a sequence; to run the test cases concurrently, select Run this suite in parallel.

Note Most automated test adapters can run only one script at a time. If you select the same adapter for multiple test cases, the scripts on that adapter will run in sequence. For parallel execution, you must select a different adapter for each script.

19. For sequential mode only, choose any of these options:

- a) Select Pass execution variables between scripts if your test scripts use execution variables and you must pass the values between scripts during execution.
- b) Select Stop suite execution if any test does not pass to stop the suite execution if a test case does not pass.

20. Click Save to save the new test suite. Note You cannot run a test suite until it is saved.

21. Proceed to running a test suite.

### ***5.4.1 Adding Existing Test Suites to a Test Plan***

You can add any existing test suites to a test plan.

#### **Procedure**

1. Open an existing test plan or create a new one.
2. From the test plan table of contents, click Test Suites.
3. Click the Add Existing Test Suites icon (Add) to open the Add Test Suites window.
4. Select the test suites that you want to add to the test plan. If you have a large number of test suites, you can use the filtering capabilities to narrow down the list.
5. Click OK.
6. Save the changes to the test plan.

#### **Results**

The test suites are added to the test plan.



## 5.4.2 Adding a New Test Suite to a Test Plan

In the Test Suites section of a test plan, you can create a new test suite.

### Procedure

1. Open an existing test plan or create a new one.
2. From the test plan table of contents, click Test Suites.
3. Click the Add New Test Suites icon (Create test suite) to open the New Test Suites window.
4. Type a name for the new test suite.
5. Type a description of the test suite.
6. Assign a numerical value for the relative weight of the test case.

How you assign the weight value can be determined using different criteria, such as importance to test effort or the relative complexity of the test suite. For example, a test suite that is assigned a weight of 100, could be twice as important or complex as a test suite that has an assigned weight of 50. The weight should not be used as a measurement of time. If you are intending to measure the time that is assumed to be required for the test suite then use the Estimate field.

7. Select a test suite template.
8. Under Category, select a Function, Test Phase, Theme, and Type.

You can use these attributes to organize your test cases into various groups that can later be sorted. Your team is free to define these attributes any way that makes sense.

9. Click OK.

The new test suite is added to the test plan. Repeat these steps to create and add multiple test suites to the test plan.

10. Save the changes to the test plan.

### Results

The test suites are created and added to the test plan.



## 5.5 Test Execution Records

You have several options for creating and managing test case and test suite execution records, including creating them manually or automatically, creating them from a test plan, and editing existing test case and test suite execution records.

### 5.5.1 Single Test Case Execution Records

In addition to generating multiple test case execution records automatically, you can create a single test case execution record manually. Test case execution records are used to specify the execution environments for a specific instance of a test case, to run a test case instance, and to track the status of each test run.

#### Procedure

1. In the main menu, click Construction > Create > Test Case Execution Record. Tip: You can also generate test case and test suite execution records automatically in the Test Case Execution Records or Test Suite Execution Records section of a test case or test suite.
2. In the Create Test Case Execution Record page, type a name for the new test case execution record.
3. Select a team area and an owner from the lists. Typically, the owner is the person responsible for running the test. Note: You must explicitly enable team areas by going to Manage Project Properties.
4. Type a description of the test case execution record.
5. Complete the Overview section.
  - a) To set the associated test plan, click the Select Test Plan icon (Add) to apply the test case execution record to a particular test plan, or leave it unassigned. After you select a test plan, by default the Iteration field is populated with the value from the associated test plan.
  - b) Required: To set the associated test case, click the Select Test Case icon (Add) to select the test case to which the test case execution record should be assigned. Note: A test case execution record must be assigned to a test case. If you selected a test plan for the test case execution record, only test cases associated with that test plan are included in the list.

After you select a test case, the following fields are automatically populated with values from the associated test case: Default Test Script, Available Test Environments, Priority, Weight, and Estimate. You can edit these values if needed.
  - c) In Default Test Script, select the test script that is associated with the selected test case.



- d) In Iteration, select the test iteration to which the test case execution record should be assigned. Test iterations are defined in the Test Schedule section of the test plan and are only available here when you assign the test case execution record to a particular test plan.
- e) In Available Test Environments, select a test environment that defines the runtime environment. A test environment is a set of saved attributes, for example, a particular browser, operating system, hardware platform, and other attributes for a particular machine. If your project uses channels, available channels will appear once a test environment is selected.
- f) Define categories.  
You can use categories to define a hierarchical organization of test plans, test cases, test scripts, and other test artifacts. You can also define subcategories and other category relationships. You can use the default categories that are provided or create your own. To define categories, click the Manage Test Case Execution Record Categories icon (Manage category).
- g) Assign values to the custom attribute fields. If an administrator defined any custom attributes for test case execution records, they display in the Attributes section of the Summary. The field values for custom attributes can be in the format of text, integer, or date. You can sort or filter your test cases using custom attributes in the same way that you use categories.

6. Click Save.

## What to do next

After you run the test case, you can record the time that was required to complete the test case instance in the Time Spent field. This information can be used to improve estimates of the time required for future test runs and to track the time invested in testing efforts.

### 5.5.2 Test Suite Execution Records

You can manually create a test suite execution record. Test suite execution records specify the execution environments for an instance of a test suite, run a test suite instance, and track the status of each test run.

## Procedure

1. From the main menu, click Construction, and in the Create section, click Test Suite Execution Record. The Create Test Suite Execution Record page opens.
2. At the top of the page, type a name for the new test suite execution record.
3. If team areas are enabled for the project area, select a team area from the list.  
Remember Team areas can be enabled only from the Manage Project Properties page.
4. Select an owner from the list. Typically, the owner is the person who is responsible for running the test suite.

5. Type a description of the test suite execution record.
6. Complete the Overview section.
  - a) To associate a test plan with the test suite execution record, click the Select Test Plan icon (Add). If you select a test plan, the Iteration field is populated with information from the associated test plan.
  - b) Required: To assign the test suite execution record to a test suite, click the Select Test Suite icon Add.

Important A test suite execution record must be assigned to a test suite. If you selected a test plan for the test suite execution record, only the test suites that are associated with that test plan are included in the list.

After you select a test suite, these fields are automatically populated with values from the associated test suite: Available Test Environments, Priority, Weight, and Estimate. You can edit those values.
  - c) Select the iteration to assign the test suite execution record to. Test iterations are defined in the Test Schedule section of the test plan. If you assigned the test suite execution record to a test plan, the test iterations are available only in the Test Schedule section.
  - d) From the Available Test Environments list, select a test environment that defines the runtime environment. A test environment is a set of saved attributes; for example, a particular browser, operating system, hardware platform, and other attributes for a particular computer.

By default, the test environment that you specify at the test suite execution record level is used for all test cases that are run as part of the suite. To use a different test environment for a test case in a test suite, specify a new environment in the Test Cases section of the test suite.
  - e) Define categories.

You can use categories to define a hierarchical organization of test plans, test cases, test scripts, and other test artifacts. You can also define subcategories and other category relationships. You can use the default categories that are provided or create your own. To define categories, click the Manage Test Suite Execution Record Categories icon Manage category.
  - f) Assign values to the custom attribute fields. If an administrator defined any custom attributes for test suite execution records, the attributes are shown in the Attributes section of the Summary. The field values for custom attributes can be in text, integer, or date format. You can sort or filter your test cases by using custom attributes in the same way that you use categories.
7. Click Save. By default, when a test suite execution record is created, test case execution records are created. You can change that behavior in the Execution Preferences section of the Manage Project Properties page.



## What to do next

After you run the test suite, you can record the time that was required to complete it in the Time Spent field. You can use that information to improve your estimates of the time that is required for future test runs and to track the time that you invest in testing.

### 5.5.3 Automatic Generation of Test Execution Records

Use test case execution records and test suite execution records to specify the execution environments for each test case and test suite and to track the status of each test run. A wizard is provided for generating test case execution records and test suite execution records automatically.

## Before you begin

Before you can generate test case execution records or test suite execution records, you must have created at least one test case or one test suite.

## Procedure

To generate test case execution records or test suite execution records automatically:

1. Do one of the following:
  - a) From an open test case, select Test Case Execution Records.
  - b) From an open test suite, select Test Suite Execution Records.
2. Click the Generate New Test Case Execution Records or Generate New Test Suite Execution Records icon (Generate new test execution record).

This starts the wizard to generate test case execution records or test suite execution records. After the wizard starts, you can cancel by clicking the X in the upper right corner.

3. Select a Test Plan and Test Iteration from the lists.

Test iterations are the various phases or iterations in your test plan. Test iterations are only available when you select a particular test plan.

4. Optional: Select an Owner for the test case or test suite execution records.

Note You can choose to have an owner automatically assigned when the execution record is executed by setting the Automatically assign executer as owner to Test Case and Test Suite execution record if owner is not assigned execution preference. For information on how to set the preference, see Test execution preferences.

5. Optional: Select a Channel for the test case or test suite execution records. Choosing a channel guides test environment selection by restricting options for test environment reuse and suggesting attributes for test environment generation.

6. If you have previously defined your test environments, click Reuse Existing Test Environments, select the test environments to reuse, and click Next.
7. If you have not previously defined your test environments, click Generate Test Environments.

- a) Select one or several environment attributes from each column. The number of test case execution records or test suite execution records created depends on the number of attributes selected, the level of coverage that you choose, and the Advanced Properties (Inclusions, Exclusions, and Weightings) that you select.

Note Advanced Properties may not be visible until you select at least two attributes from one of the columns of attributes.

- b) In the Generation Properties section, select the level of coverage that you want.

Use this setting and the Advanced Properties setting to fine-tune the test case execution records or test suite execution records that will be generated.

Choose Minimal to ensure that each selected attribute is covered at least once, with no attempt to cover specific combinations of attributes. For example, if you select one attribute from three columns, three test case execution records or test suite execution records are created, ensuring that each selected attribute is covered at least once.

Choose Medium - pair-wise interaction to ensure that each combination of paired attributes is covered at least once.

Choose Large - three-way interaction to ensure that each three-way combination of attributes is covered at least once.

Choose All - all permutations to ensure that all combinations of attributes are covered at least once.

Note The coverage options only become available when you start selecting attributes. The greater the selected coverage, the more attributes and columns that you must select.

- c) Click Advanced Properties to display a window with three tabs: Inclusions, Exclusions, and Weightings.

Click Exclusions to specify the attribute combinations to explicitly exclude, for example Safari browser running on Windows XP.

Click Inclusions to specify the attribute combinations to always include, for example Internet Explorer 7.x running on Windows XP.

Click Weightings to set the weight or importance of each attribute relative to the other values for that attribute.

For example, you can assign greater weight to Windows XP than SUSE Linux to ensure that at the very least Windows XP will be tested.

- d) Click Next.

The wizard creates a preview of the Generated Test Environments from which the test case execution records or test suite execution records will be generated.



8. Select the test environments that you plan to commit to and click Next.

The wizard creates a preview of the generated test case execution records or test suite execution records.

Tip You can create unique names for each test case execution record or test suite execution record by selecting the To make the new test environment names unique, add a prefix to the generated name option.

9. Select the test case execution records or test suite execution records that you want to keep.
10. Optionally, decide how you want to group the generated test case execution records or test suite execution records by selecting one of the choices in the Group By list.
11. Click Finish.

The wizard generates the test case execution records or test suite execution records according to the criteria you have selected.

12. Click Save to add the test case execution records or test suite execution records to the test case or test suite.

## Results

After the test case execution records or test suite execution records are generated, you can select one and run it.

### ***Test Case Execution Records - Examples***

The following table shows four different sets of settings for generating test case execution records. In row 2, the Coverage is changed from Minimal to Medium. In row 3, three more modifications are made, and in row 4, only one more modification is made. Each change results in an increase in the number of test case execution records that are created.

### ***5.5.4 Generation of Test Execution Records from Test Plans***

You can create test case and test suite execution records directly from a test plan.

Table 1. Examples of test case execution records.

Example	Browsers selected	CPUs selected	OS selected	Coverage	Inclusions & Exclusions	Weighting	Result (TERs created)
1	Firefox, IE	Intel i386, PowerPC®	Windows XP, SUSE	Minimal	None	Priority to Firefox, Intel, & Windows XP	2
2				Medium			4
3	Firefox, IE, Safari		Windows XP, SUSE, Mac OS X			Safari and Mac OS X priority 3.	9
4				Large			18

## Procedure

1. Open an existing test plan or create a new one.
2. From the test plan table of contents, click either Test Cases or Test Suites.
3. Select the test cases or test suites that you want to create execution records for.
4. Click the Generate New Test Case Execution Records or Generate New Test Suite Execution Records icon (Generate new test execution record).
5. This starts the wizard to generate test case execution records or test suite execution records. After the wizard starts, you can cancel by clicking the X in the upper right corner.
6. Follow the steps in the wizard starting at Step 3.

### ***5.5.5 Editing Test Execution Records***

In test case execution records and test suite execution records, you can change the name, team area, owner, iteration, priority, estimate, weight, or any custom attribute. You can also update the default test script for test case execution records.

## Procedure

1. Click Execution, and in the Browse section, click Test Case Execution Records or Test Suite Execution Records.
2. Select the test case execution record or test suite execution record to edit.
3. If necessary, type a different name for the record.
4. To change the default test script for test case execution records, select a test script from the Default Test Script list.
5. Make other changes as needed. You can change the team area, iteration, owner, priority, estimate, weight, or custom attributes.
6. Click the Save icon Save. If your changes cause a test suite execution record to become out of sync with its test case execution records, you are prompted to synchronize them. Click the Synchronize Test Case Execution Records with Test Suite Execution Records icon to do so.



## 5.6 Test Scripts

You can use IBM Rational Quality Manager to manage your manual test scripts.

You can use the embedded manual test editor to create manual test scripts and accomplish these tasks:

- Attach images, text files, comments, and verification text to be viewed during a test run
- Display traceability from individual test script steps to requirements associated with the test case or overall project
- Use keywords to ensure reusability and add automation capabilities to groups of statements in your manual tests
- Supply your manual test scripts with realistic data values by inserting test data variables
- Submit defects during a test run

Test scripts are associated with test cases. When you run a test, you run the test case execution record that is associated with the test case. In a manual test, you perform each test instruction by stepping through the statements and verifying that certain conditions pass or fail. During execution you can capture the actual results and file defects, which can be viewed later in the execution results.

If you set up an integration with IBM Rational Functional Tester, then you can use the manual test script recorder option in the manual test editor to automatically capture a set of actions as steps in a manual test script. Using the recorder to capture script steps saves your team time and eliminates user errors that might occur when manually writing manual test script steps. The manual test script recorder also gives you the option to automatically save screen captures. You can also use the manual test script recorder for exploratory testing.

### 5.6.1 Execution Variables

To use specific values in your test scripts during a test run, define execution variables in test scripts or keywords. You can also pass execution variables from a test suite or a test case to associated test scripts or keywords. If you are an administrator or a test lead, you can set execution variables at the project level.

You can use execution variables with all manual scripts, command-line scripts, and JUnit Selenium scripts. If you use an adapter that supports execution variables, you can use them with automated test scripts. To determine whether an adapter supports execution variables, consult the adapter-specific documentation.



## Artifact- and project-level execution variables

Execution variables can be defined at project or artifact levels. If you are a project administrator and have permission to save project execution variables, you can define project execution variables. Because project-level execution variables are centrally defined, they can be consistently reused across a team. In addition, project execution variables have sets of values, called enumerated values, that are defined for use in test artifacts. The use of enumerated values safeguards testers from accidentally entering an invalid value at run time.

Test authors can define artifact execution variables in the context of test suites, test cases, and test scripts. Those test artifact execution variables can take any value.

- To use values that are present throughout a test suite or a test case, define artifact execution variables in the Execution Variables section of that test case or test suite.
- To use an execution variable in a manual test script or keyword, define the variable in the Execution Variables section of the Manual Test editor. After you define a variable, you can insert references to it in a manual test script or keyword step. During a test run, the current value of the execution variable replaces the reference in the step.
- To set the value of a particular execution variable, you can insert execution variable steps in a manual test or keyword. You can create and insert an execution variable step from the Execution Variables window. During a test run, you are prompted to assign a value to the execution variable. After you set the value, all references to that execution variable have the value that you provided.

At the artifact level, if values for the same variable are defined in multiple places, the values of the higher-level variable override the values of the lower-level variable. For example, a variable is defined in a test suite and in a test case that is within the test suite. The value of the variable that is defined in the test suite overrides the initial value of the variable that is defined in the test case. In another example, a variable is defined in a test case and in a test script that is within the test case. The value of the variable that is defined in the test case overrides the initial value of the variable that is defined in the test script.

- The values of test suite execution variables override the values of test case, test script, and keyword execution variables
- The values of test case execution variables override the values of test script and keyword execution variables
- The values of test script execution variables override the values of keyword execution variables

An artifact can incorporate both project and artifact execution variables. However, if two or more of those variables have the same name, a conflict occurs that prevents the artifact from being saved. You can change the name of a project-level execution variable to



match the name of an artifact-level execution variable. In that case, the value of the artifact-level variable overrides the value of the project-level variable when the test is run.

## Execution variables in sequential test-suite runs

If you run sequential test suites, you can pass execution variables from one test script to another. For example, the first step in a test suite runs a test script that creates a user and sets an execution variable that is named `userID`. Subsequent steps in the test suite run test scripts that reference the value of the `userID` execution variable, update the user information, and verify that the user information is correct. You can enable this behavior in the Test Case section of a test suite or in the Run Test Suite window.

## Execution variables in command-line scripts

When you run a command-line test that includes execution variables, the values of the execution variables are provided as environment variables to the process that runs the command-line test. The names of those environment variables start with "RQM\_" to distinguish them from other environment variables.

For example, if a test suite contains an execution variable that is named `myVar`, an environment variable that is named `RQM_myVar` is created.

Remember Ideally, execution variable names include only valid Java identifiers: A-Z, a-z, 0-9, `_` and `$`. If an execution variable contains an identifier that is not in Java, an underscore (`_`) replaces that identifier in the new environment variable name. For example, if you have an execution variable that is named `user.Name`, the resulting environment variable is `RQM_user_Name`.

In addition, an environment variable that is named

`RQM_ExecutionVariablesFile`

is created, and its value is an absolute path name to a text file; for example,

`RQM_ExecutionVariablesFile=C:\Users\tneal.DEV\AppData-Data\Local\Temp\QMAadapter1220694107401772431.vars.`

That file contains name and value pairs: one for each execution variable that is defined when the command-line script runs. For example:

```
#Execution Variables
#Wed Mar 02 15:48:54 EST 2011
testcase2Var=set statically in testcase2
testcase1Var=set statically in testcase1
testsuite1Var=set statically in testsuite1
```

When the command-line test is complete, the text file is read. If the test suite allows execution variables to be passed, the contents are used to pass the values of execution variables to the next test script. You can modify the text file to create execution variables or to change the value of execution variables that are passed to other test scripts.

## Built-in execution variables

In certain cases, your test might need to retrieve more information about the test execution environment that it is running in. For example, the test might need such the name of the test suite or the test case that is being run. Built-in execution variables provide environment information. To enable built-in variables, select the Include built-in variables check box in the Execution Variables section of the test script editor.

When that check box is selected, these extra execution variables are created:

```
RQM_TESTSCRIPT_NAME
RQM_TESTSCRIPT_WEBID
RQM_TESTCASE_NAME
RQM_TESTCASE_WEBID
RQM_TESTCASE_RESULT_WEBID
RQM_TESTCASE_RESULT_NAME
RQM_TESTSUITE_NAME
RQM_TESTSUITE_WEBID
RQM_TESTSUITE_RESULT_WEBID
RQM_TESTSUITE_RESULT_NAME
RQM_TESTPLAN_NAME
RQM_TESTPLAN_WEBID
RQM_TESTPHASE_NAME
RQM_TESTPHASE_WEBID
RQM_TESTCASE_EXECUTIONRECORD_NAME
RQM_TESTCASE_EXECUTIONRECORD_WEBID
RQM_SUITEEXECUTIONRECORD_NAME
RQM_SUITEEXECUTIONRECORD_WEBID
RQM_PROJECT_NAME
RQM_PROJECT_ALIAS
RQM_BUILDRECORD_NAME
RQM_BUILDRECORD_WEBID
RQM_BUILDDEFINITION_NAME
RQM_BUILDDEFINITION_WEBID
RQM_CONFIGURATION_NAME
RQM_CONFIGURATION_CPU
RQM_CONFIGURATION_BROWSER
```



RQM\_CONFIGURATION\_ADAPTER  
RQM\_CONFIGURATION\_DATABASE  
RQM\_CONFIGURATION\_APPLICATIONSERVER  
RQM\_CONFIGURATION\_MANAGEMENTAGENT  
RQM\_CONFIGURATION\_OS  
RQM\_CREATEDBY\_USERID  
RQM\_CREATEDBY\_NAME  
RQM\_LABRESOURCE\_NAME  
RQM\_LABRESOURCE\_WEBID

## 5.6.2 Manual Test Scripts and Statements

Manual testing is a type of testing that includes human involvement and no automation.

In a typical test process, you first create test cases, and then you create the test scripts. The next step is to associate the test scripts with the test cases. During test execution, the script opens and guides the tester through the test step-by-step.

The way you divide the creation of test artifacts among your group depends on the workflow of your organization. In larger organizations, several people might be involved in developing test artifacts. It might make sense for one team member to create the scripts, while another team member creates the test cases. In smaller organizations, one person might fill several roles, so that person might create the script at test-creation time.

When constructing a manual test script, use the test editor for creating and editing test scripts. Type statements into the editor and assign a type to each step. You can also create manual test scripts automatically in the Test Case Design section of a test case.

If you have set up an integration with IBM Rational Functional Tester, you can use the manual test script recorder option in the manual test editor to automatically capture a set of actions as steps in a manual test script. (This function is also referred to as "assisted manual testing.") Using the recorder to capture script steps saves your team time and eliminates user errors that might occur when manually writing manual test script steps. The manual test script recorder also gives you the option to automatically save screen captures. You can also use the manual test script recorder for exploratory testing.

You can create two types of manual testing statements:

- Execution Steps (Step) tell the testers what actions to perform when they run the script. For example, "Start the application." represents an execution step.
- Reporting Steps (Reporting step) are higher-level checkpoints. Reporting steps might summarize the result of several execution steps; for example, a reporting step might ask, "Were you able to log in?"

You can attach images, files, comments, and verification text to a statement. Attachments are accessible during the test run. In addition, you can insert keywords and test data.

### ***5.6.3 Using the Manual Test Script Recorder***

The manual test recorder integrates with IBM Rational Functional Tester so that you can automate the creation and editing of manual test scripts. You record user gestures in your application, which are then converted into manual script statements in English.

#### **Manual Test Script Recorder Overview**

You can use the manual test script recorder in IBM Rational Quality Manager to simplify the creation of manual test scripts. Manually recording script steps saves your team time and eliminates user errors that might occur when manually writing the steps. You can also use the recorder for exploratory testing.

**Note** The application that you are testing must be an HTML application, Java™ application or an applications with Dojo controls.

The manual test recorder uses an integration with IBM Rational Functional Tester to automate the creation and editing of manual test scripts. You use a manual test script recorder to record user gestures in your application, which are then converted into manual script statements in English. The manual test script recorder also automatically captures screenshots of the application where the gestures were made.

#### ***Basic workflow for configuring and using the manual test recorder***

1. Install IBM Rational Functional Tester.
2. Configure the Rational Functional Tester adapter for use with IBM Rational Quality Manager.
2. Optional: Prepare the test environment in Rational Functional Tester by enabling the browsers, Java run time environments (JREs), and Eclipse platforms. By default, the test environment is automatically enabled by Rational Functional Tester. But in certain scenarios you must manually enable the required components. For more enablement information, see *Automatically enabled environment for recording functional test scripts*.
3. Optional: If you want to open your application under test (AUT) directly from the manual test script recorder, configure the AUT in the Application Configuration Tool in Rational Functional Tester.
4. Start the Rational Functional Tester adapter.
5. Optional: Modify user preferences for the manual test script recorder.
6. Create or modify a manual test script by using the manual test script recorder.



## Enabling the Manual Test Script Recorder

Enable the manual test script recorder option in the manual test editor so that you can automatically capture a set of actions as steps in a manual test script.

### ***Before you begin***

IBM Rational Functional Tester version 8.2.2 is required to use the manual test script recorder. For a summary of the setup that is required, see [Setting up Rational Functional Tester](#).

You must meet these prerequisites to use the manual test script recorder:

- Rational Functional Tester version 8.2.2 must be installed on the workstation where you record the script. For more information, see the installation section.
- The Rational Functional Tester adapter must be configured for use with IBM Rational Quality Manager. For more information, see [Configuring and running the IBM Rational Functional Tester adapter for IBM Rational Quality Manager](#).
- The test environment in Rational Functional Tester must be enabled. Although the test environment is typically enabled by default, sometimes you must manually enable the required components. For more information, see [Automatically enabled environment for recording functional test scripts](#).
- If you want to open the application for testing directly from the manual test script recorder, you must configure the application in the Application Configuration Tool in Rational Functional Tester. For more information, see the [Configuring applications for testing](#) topic.
- To use the manual test script recorder, a user must be assigned a project area role that has the Save Adapter Request permission.

### ***About this task***

After you integrate with Rational Functional Tester, ensure that the manual test script recorder option is enabled so that you can automatically capture a set of actions as steps in a manual test script.

Capturing script steps by using the manual test script recorder saves your team time and eliminates errors that might occur when steps are manually written. The recorder can automatically save screen captures. You can also use the manual test script recorder for exploratory testing.

By default, the manual test recorder icon Record is available to all users in the manual test script editor. You can restrict the use of the recorder in specific projects by setting the Disable recorder option for creating manual test script preference.

## Procedure

To enable or disable the manual test script recorder in the manual test script editor in Rational Quality Manager:

1. In the banner, click the Admin Administration menu, and then click Manage Project Properties.
2. In the Properties section, click Manual Script Preferences.
3. Select or clear the Disable recorder option for creating manual test script check box. The check box is cleared by default, so the recorder option is usually enabled.

## What to do next

Start the Rational Functional Tester adapter and then create or modify a manual test script by using the recorder.

## Setting User Preferences for the Manual Script Recorder

You can optimize settings for the manual test script recorder in the Preferences window.

## About this task

Use the options in the Recorder Preferences section to set user preferences for the manual test script recorder in the manual test script editor.

## Procedure

1. Click User Profile (User Profile menu) in the banner, and then click My Preferences for Quality Management.
2. Click the Recorder Preferences section, indicate your preferences, and click OK.

Property	Description
Minimize the browser window when using the recorder option for creating manual script	If the browser window should minimize automatically when you start the recording. This is not supported on all browsers.
Display images captured during recording	Specify the default location to store the images that are captured during the recording process.
Show As	Specify the default format for images that are stored in the Description or Expected Results column.



## **Results**

Preferences are saved to the user profile. Most settings are not active until the next time you log in. Your default values are shown in the Record Manual Script dialog box that opens when you start a recording.

## **Creating or Modifying Manual Test Scripts with the Recorder**

You can use the manual test script recorder in the manual test editor to complete a set of actions and capture them as steps in a manual test script. This approach saves your team time and eliminates user errors from manually writing manual test script steps. You can also save screen captures automatically or do exploratory testing.

### ***Before you begin***

To use the manual test script recorder in the manual test editor, you must meet the following prerequisites:

- The system that you are using to record the steps must have access to an IBM® Rational Functional Tester adapter that is enabled for recording. See *Configuring the manual test script recorder*.
- The Rational Functional Tester adapter that is enabled for recording must be running. See *Configuring and running the IBM Rational Functional Tester adapter for IBM Rational Quality Manager*.
- The application that you are testing must be an HTML application, Java™ application or applications with Dojo controls.
- If Rational Functional Tester does not support the language that you are using for Rational Quality Manager, English is used for content from Rational Functional Tester.

### ***About this task***

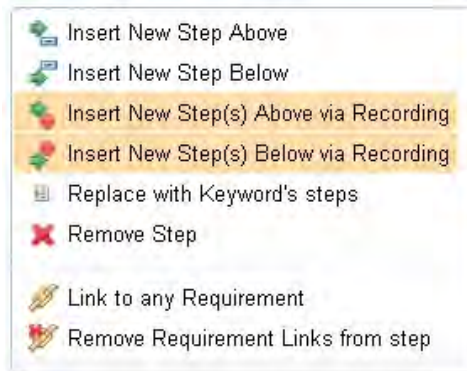
You can use the manual test script recorder to create a set of steps in a new manual test script or to add more steps to a saved manual test script. After you create or modify a manual test script with the recorder, you run it like any other manual test script.

### ***Procedure***

1. Start the process to create or modify a test script with the recorder by doing any one of these steps:
  - a) Create a manual test script.
  - b) Open a manual test script.
  - c) Open the Test Case design section of a test case.

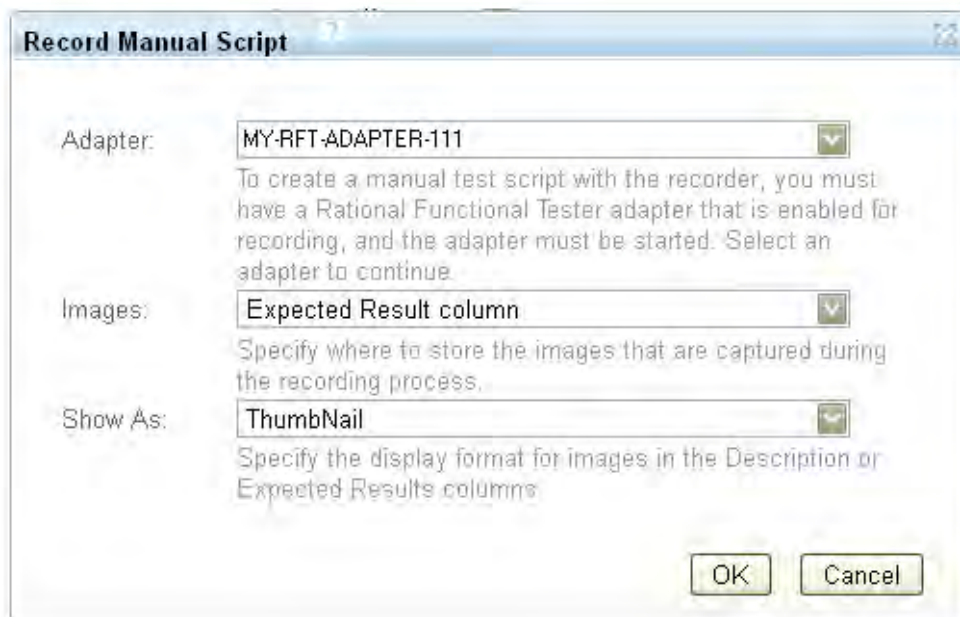


- d) Open the Test Scripts section of a test case.
2. Open the Record Manual Script window in one of these ways:
  - a) In a new manual test script or in the Test Case Design or Test Scripts section of a test case, click the Record Manual Script (Record) icon.



- b) In an existing manual test script, select a step where you can add new steps. Click the menu arrow in the step, and select Insert New Step(s) Above via Recording or Insert New Step(s) Below via Recording.

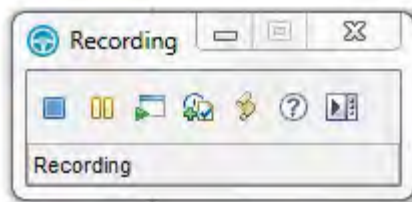
Note If you started the recording from the Test Case Design or Test Scripts section of a test case, the Record Manual Script dialog box includes fields for the name and description of the new manual test script.



3. Select an Adapter. The list includes only Rational Functional Tester adapters that are enabled for manual test script recording. If none is listed, no Rational Functional Tester adapters are configured. Contact an administrator.



4. Specify where to store the Images that are captured during the recording process. Your selection is stored and used as the default value for the next recording. Select one of these options:
  - a) Expected Result column (default)
  - b) Description column
  - c) Attachment
  - d) No Image
5. If you choose to store images in the Description or Expected Results column, click Show As to specify whether to display the images as thumbnails, full-size images, or as links to separate image files. Your selection is stored and used as the default value for the next recording.
6. Click OK to start recording. If you are using the preference to automatically minimize the browser window, then the Rational Quality Manager user interface is minimized, and you see a Rational Functional Tester recording toolbar on the desktop.



Note If you use Firefox version 7 or later, the browser does not automatically minimize when you start the recording, even if you selected the user preference to minimize the browser. You must manually minimize the browser window.

7. Complete the actions to record as manual test script steps. For example, if the first step in the manual test script is to open the application under test, then you open the application on your computer. For information about the icons that are displayed on the Rational Functional Tester recording toolbar, see the Recording toolbar.
8. When you finish capturing the test script actions, click Stop on the recording toolbar. The manual test script shows the steps that the recorder captured.

Tip: If your browser does not correctly perform the window focus request from Rational Quality Manager, restore the window manually. If Rational Quality Manager is on the machine that is running the recorder, you can disable the user preference to minimize the browser window and always manually minimize or restore when you record.

9. Review the new manual test script steps and update the steps as needed, and click Save.

### **What to do next**

Associate the manual test script with a test case, test case execution record, or test suite execution record, and run it.

### 5.6.4 Adding Existing Scripts to a Test Case

When you add an existing test script to a test case, it is listed in the Test Scripts section of the test case.

#### Procedure

1. From an open test case, click the Test Scripts section, and then click the Add Existing Test Scripts icon (Add). The Add Test Scripts window opens.
2. From the list of test scripts, select the test scripts to add, and then click OK.  

Tip To show a preview of individual test script details, hover your cursor over the test script ID or name.
3. If you have a large number of test scripts, you can use the filtering capabilities to narrow down the list.
4. Save the changes to the test case.

#### Results

The test scripts are added to the Test Scripts section of the test case.

### 5.6.5 Creating Manual Test Scripts

The Manual Test editor is a rich-text editor for constructing test scripts.

#### About this task

You can create a test script on the Construction page or create a script from within a test case.

#### Procedure

1. Use one of these procedures to create a test script.
  - a) To create a test script from the Construction page:
    - a. In the main menu, click Construction > Create > Test Script.
    - b. At the top of the new test script window, type a descriptive name that identifies the purpose of the script.
    - c. If team areas are enabled for the project area, select a team area from the list.  
Note You must explicitly enable team areas in Manage Project Properties.
    - d. Select an owner for the test script.
    - e. For a manual test script, make sure the script type is Manual.



- f. Type a description.
  - g. To assign the task of completing the test script to another team member, save the test script, and then click Work Item: Create. Note Creating a work item to complete the test script does not change the owner of the test script.
- b) To create a test script from within a test case:
  - a. Open the test case to which to add a test script.
  - b. From the Test Scripts section of a test case, click the Add New Test Script icon (Add new test script). The New Test Script dialog box opens.
  - c. In the Name field, type a descriptive name that identifies the purpose of the script.
  - d. Type a description.
  - e. The type is Manual by default.
  - f. Click OK.
  - g. Save the test case. This creates the test script shell.
  - h. Click the test script name to open it in the Manual Test editor.
  - i. Select an owner for the test script.
  - j. To assign the task of completing the test script to another team member, click Work Item: Create. Note Creating a work item to complete the test script does not change the owner of the test script.
- 2. Optional: Add test data.
 

You can use test data to supply realistic values to the variables in a manual test script. Test data is a collection of related data records. During execution, the variables in the manual test script are replaced with data from the test data. As a result, you can use a single script to test multiple data records.

  - a) Next to the Test Data field, click the Change Associated Test Data icon (Edit).
  - b) Select a test data artifact; then click OK.
  - c) Optional: To filter data records so that a subset are used when the test script is executed, complete the following steps:
    - a Next to the Test Data Usage field, click the Select Records from Test Data icon (Edit).
    - b Under Record Selection Criteria, select a column name; then set a filter value.
    - c To verify the selection, click the Show button. The data records are filtered according to the filter settings. Tip You can click the Clear icon to reset the selection criteria.
    - d Click OK.
- 3. Define categories.

You can use categories to define a hierarchical organization of test plans, test cases, test scripts, and other test artifacts. You can also define subcategories and other category relationships. You can use the default categories that are provided or create your own. To define categories, click the Manage Test Plan Categories icon (Manage category).

4. Assign values to the custom attribute fields.

If an administrator defined any custom attributes for test scripts, they display in the Attributes section of the Summary. The field values for custom attributes can be in the format of text, integer, or date. You can sort or filter your test scripts using custom attributes in the same way that you use categories.

5. To begin typing your statements, click Click to add step in the text editor. A new step is added with the default step type as Execution Step. A step is composed of three fields: Description, Expected Results, and Validates.

Note To change the step type to a reporting step, click the arrow next to the Execution Step (Step) icon, and then click Reporting Step (Reporting step).

6. In the Description field, describe the action to take. Each step is an individual action.
7. In the Expected Results field, specify the conditions that must be met for a step to be considered successful.

Tip Use Shift+Enter to increase the height of the editor so that you can enter multi-line text.

8. To continue adding steps, press Enter to move to the next line.

Note You can insert a step before or after the current step by using the Action Menu, and then clicking Insert New Step Above or Insert New Step Below. You can also use shortcut keys: Use Ctrl+Shift+Enter to insert a new step before the current step and Ctrl+Enter to insert a step after the current step.

9. The Validates column displays requirements that are associated with test script steps. You can add requirements traceability to individual steps by either of the following methods:

- a) To link steps to requirements that are associated with the test case, at the far right, in the Requirement View, scroll down to view associated requirements; then drag individual requirements to test script steps.
- b) To link steps to any requirements associated with the project, in the Action Menu, click Link to any requirement. Note For the Validates column to be available, the project administrator must have enabled linkage between requirements and test script steps via Manage Project Properties > Test Script Preferences.

10. Optional: To cut, copy, and paste test script steps, select the steps and use the Cut, Copy, and Paste options in the Action Menu. There is no "Delete" or "Remove" control for steps. Cut steps to remove them from a script.

11. Optional: To toggle between normal and compact views, click the Compact View (Compact View) icon. Only a portion of the steps are displayed when you compact the view, making it easier to view the entire script.

12. Optional: To change the width of each column, drag the column dividers.



13. Optional: To add comments in your steps, click the Show/Hide Comments Column icon (Show/Hide comments column) and select Show Comments Column.
14. When you have finished adding steps, click Save at the top of the page to save the test script.

Tip To quickly save a script, press Ctrl+S.

15. Optional: To make a copy of the test script, in the upper right corner of the test case, click the Duplicate icon (Copy ).
16. Optional: Click the Print View icon (Print view) in the upper, right corner of the test script to display a printable version of the test script.
17. Optional: Click the Set to Lock/Unlock icon (Change lock) in the upper right corner of the test script to lock or unlock the test script. Depending on how your project is configured, you might be required to sign this action electronically by providing your password and a comment. When the test script is locked, the editable fields are disabled for editing.

Note Permission to lock a test script depends on your role-based permissions.

To reverse the locking attribute, click the icon again.

18. Optional: Click the PDF icon (Export to PDF) to export the test script details to a PDF.

## What to do next

If your organization requires a review of the test script, you can use the Formal Review section of the test script to assign reviewers and approvers.

### ***5.6.6 Manual Test Scripts from Test Case Design***

You can define test script steps in the Test Case Design section of a test case and use the design to automatically generate a test script.

## Before you begin

Before you write test script steps, check the terms that are used to identify reporting steps by clicking the Admin icon (Administration) at the upper, right part of the banner, selecting Manage Project Properties, and then clicking Manual script dictionary. Statements that contain these terms are converted to reporting steps when manual test scripts are automatically created.

Note By default, an approved test case is no longer open for editing. If a user modifies and attempts to save an approved test artifact, the following message is displayed: Do not allow saving of an "Approved" or "Retired" artifact. To change this behavior, see Allowing users to edit approved artifacts.

## Procedure

1. Open the test case to add a test script to.
2. Click the Test Case Design section.
3. To begin typing statements, click Add Content. Enter each step on a new line, which is separated by a line break. Groups of steps that are not separated by line breaks are included in the same step. The text that you include in the test case design is only converted into step descriptions, not expected results. If you have more information about expected results, you can add it to the steps after the script is generated.

Note To identify reporting steps, use the terms in the Manual script dictionary. For example, if the term report is listed as a term that identifies reporting steps, the statement Report whether the credit card information was processed becomes a reporting step after automatically creating the manual test script.

4. Use the toolbar to format steps, attach and resize images, and attach documents. If you are pasting text from another word-processing application, click the Validate (Rich Text Editor cleanup) icon to clean up the text.

Tip Click Preview to view your test script before you create it.

5. When you finish typing the statements, click the Create manual test script from test case design icon (Generate new test script) to automatically create a test script from the statements in the text editor.

Tip You can also complete this step from the Test Scripts section or when you are browsing through a list of test cases. In the latter case, click Construction > Browse Test Cases, select a test case, and click Create manual test script from test case design.

6. In the window that opens, optionally edit the title or click Work item: Create to assign the test creation to another team member, and then click OK.
7. Click the Test Scripts section to see the generated test script, and then click Save to save the test case.

Tip You can also generate the test script from the Test Scripts section.

8. To open the test script in the Manual Test editor, click the test script name.
9. If needed, you can continue to edit and format the statements with the Manual Test editor.

## Results

The manual test script is automatically generated and associated with the test case.



## 5.7 Manual Test Scripts

You create and update scripts using the Manual Test editor. The editor supports standard editing commands and specific commands for scripts.

### 5.7.1 *Attaching a File to a Step*

Instead of including many details in a complex step, attach a file to a step and then refer to that file during a test run.

#### Procedure

1. In the Manual Steps section of a test script, click the Show/Hide Contents icon Show/Hide Contents and select Show Attachments/Links.
2. Select the step to add an attachment to. In the column beside the step, click the Attachment icon (Attachment), and then click Insert Attachment. The Attachments View window opens, in which you can add or remove attachments.
3. Select the files to attach, and then click Close. The steps that contain attachments are marked with the Attachment icon.

#### Results

The files are attached to the test statement. During a test run, when you reach a step that contains an attachment, you can open that attachment by clicking the file name link.

### 5.7.2 *Adding Comments*

You can add comments to steps. During a test run, testers can view the comments.

#### Procedure

1. In the Manual Steps section of a test script, click the Show/Hide Contents icon Show/Hide Contents and select the Show Attachments/Links check box.
2. Select the step to add a comment to. In the column beside the step, click the Comment icon, and then click Insert Comment. The Comment window opens.
3. Type your comments, and then click OK. The steps that contain a comment are marked with a Comment icon Comment.
4. To edit or remove a comment, open the Attachments/Links panel, and next to the step that contains the comment, click the Comment icon.



## Results

During a test run, you and other testers can view comments by hovering over steps that contain the Comment icon Comment.

### 5.7.3 Verifying Text

When you expect certain text to be in the test application, you can associate that text with a step in a manual test script. Then, during a test run, the expected text is compared with the information in the application. For example, you can compare long strings of numbers or blocks of text.

#### Procedure

1. In the Manual Steps section of a test script, click the Show/Hide Contents icon Show/Hide Contents and select the Show Attachments/Links check box.
2. Select the step to add verification text to.
3. In the column beside the step, click the Assisted Data Entry/Verification icon Assisted Data Entry/Verification.
4. In the window that opens, enter the verification text, and then click OK. The steps that contain comparison text are marked with an Assisted Data Entry/Verification Assisted Data Entry/Verification icon.

Tip To edit or remove comparison text, beside the step that contains the text, click Assisted Data Entry/Verification. Then, click either Edit Assisted Data Entry/Verification or Remove Assisted Data Entry/Verification.

5. Run a test. During the test run, when you reach a step that contains verification text, a window opens.
6. Copy and paste the text from the application that you are testing into the Paste Text to Compare field. The results indicate whether the comparison is a success or a failure.

### 5.7.4 Providing Assisted Data Entry

In a manual test script, you can provide data entry text for a tester to use during a test run. Provide text that might help testers reduce data entry errors. For example, you can provide long strings of numbers or blocks of text.

#### Procedure

1. In the Manual Steps section of a test script, click the Show/Hide Contents icon Show/Hide Contents and select the Show Attachments/Links check box.



2. Select the step to provide data entry text to.
3. In the column beside the step, click the Assisted Data Entry/Verification icon Assisted Data Entry/Verification.
4. In the field, type the data entry text.
5. Select Use Text for Assisted Data Entry, and then click OK. The steps that contain data entry text are marked with an Assisted Data Entry/Verification Assisted Data Entry/Verification icon.

Tip To edit or remove comparison text, beside the step that contains the text, click Assisted Data Entry/Verification. Then, click either Edit Assisted Data Entry/Verification or Remove Assisted Data Entry/Verification.

## Results

During a test run, when a tester reaches a step that contains data entry text, a window opens. The tester can copy the text from the Provided Text field and paste it into the application.

### 5.7.5 Importing Manual Test Scripts

You can import test scripts that are saved as local XML files directly into your repository, where they are then available on the Browse Test Scripts page.

#### Before you begin

Ensure that the local XML file to import adheres to the XML schema documentation. See the link to schema information in step 4.

#### Procedure

1. In the main menu, click Construction > Import > Test Scripts.
2. Select Manual, and click Browse.
3. Locate the file, and click Open.
4. Click Import to import the test script into the repository.

Tip For more information on the import format and content model elements for test case XML files, see the test script XML schema documentation.

See the following example for a basic test script XML file that adheres to the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<testscript xmlns="http://jazz.net/xmlns/alm/qm/v0.1/">
  <title xmlns="http://purl.org/dc/elements/1.1/">New Customer Order Test Script</title>
  <description xmlns="http://purl.org/dc/elements/1.1/">Steps to manually test
    the Customer Order functionality</description>
  <category xmlns="http://jazz.net/xmlns/alm/qm/v0.1/" value="Web UI" term="Function"/>
  <state xmlns="http://jazz.net/xmlns/alm/v0.1/">com.ibm.rqm.planning.common.new</state>
  <steps xmlns="http://jazz.net/xmlns/alm/qm/v0.1/">
    <step xmlns="http://jazz.net/xmlns/alm/qm/v0.1/testscript/v0.1/" type="execution">
      <description>
        <div xmlns="http://www.w3.org/1999/xhtml">Log in to the system</div>
      </description>
      <expectedResult>
        <div xmlns="http://www.w3.org/1999/xhtml">Login succeeds</div>
      </expectedResult>
    </step>
    <step type="execution">
      <description>
        <div xmlns="http://www.w3.org/1999/xhtml">Place an order</div>
      </description>
      <expectedResult>
        <div xmlns="http://www.w3.org/1999/xhtml">An order is created in the system</div>
      </expectedResult>
    </step>
  </steps>
</testscript>
```

### 5.7.6 Linking Test Script Steps to Requirements

You can ensure that your testing effort adequately covers the necessary requirements by linking requirements directly to your test script steps. When you save the test script, any new requirements that were not previously associated with the parent test case are automatically added to the test case.

#### About this task

When you remove requirements from a test script step, by default, the test script is synchronized with its parent test case. If the requirements that were removed from a single test script no longer link to any other test scripts of the parent test case, when you save



the test script, you can remove the requirement associations from the parent test case. You can configure the options for synchronizing test script requirements in the Preferences window.

## Linking Test Script Steps to Requirements

If you have test scripts that fulfill multiple requirements, you can link a test script step to one or more requirements.

### *Before you begin*

1. Link a Quality Management project to a Requirements Management project.
2. Add test scripts to a test case that has associated requirements.
3. By default, the Validates column in the test script editor which allows you to link requirements to test script steps is disabled. To turn on the Validates column in the Manual Steps table, see Setting test script preferences for requirements management. This action enables linking between test script steps and requirements.

**Limitation** When you create a link to external artifacts in a CLM workbench source application, a link is automatically created back to the source artifact in the target application, which is called a back link. However, a state of synchronization between bidirectional links and the traceability between source and target artifacts are not guaranteed.

### *Procedure*

1. Open an existing test script (Construction > Browse > Test Scripts) or create a new test script (Construction > Create > Test Script).
2. If the attachments and links column is not shown, click the Show/hide Contents icon; then select the Show Attachments/Links check box.
3. To link a single step to a requirement that is associated with the parent test case:
  - a) From the manual test editor toolbar, click the Show Requirement View icon (Show Requirement View) to open the Requirement View.
  - b) In the Requirement View, select one or more requirements; then click the Add Requirement Link icon (Add Requirement Link). The linked requirement is shown in the attachments and links column. Or, drag the requirement link from the panel and drop it in the attachments and links column.
4. To link a step to any requirement that is associated with the project:
  - a) In a step, click the Action Menu icon (Action menu drop down); then click Link to any Requirement.
  - b) In the Links window, select a requirement from the Requirements Management project that is linked to this project.

Tip If you want requirement links in a test script to be automatically copied to the Requirement Links section of a parent test case when you add an existing test script to the test case, you can enable the Copy Requirement links into Test Cases when adding new child Test Scripts option in Test Scripts Preferences.

5. Click Save.

## **Results**

Linked requirements are displayed in the Attachment/Links column of the test script.

The Attachment/Links column on the right side of the page shows a summary of all of the linked requirements for the test script. You can hover over the requirement link to see details of the requirement. You can also link directly from a requirement in the Requirements Management application to a test script step. The linked requirement in the Requirements Management application displays a Validated By link to the test script.

## **What to do next**

To remove all linked requirements from a test script step, click the Action Menu and select Remove Requirements Links from step.

## **Setting Test Script Preferences for Requirements**

You can set a preference on whether to allow the linking of test script steps to various requirements in the manual test script editor. When enabled, the Validates column is shown in the manual test script editor, listing any requirements that you have linked to steps. You can also enable a setting to automatically copy the associated requirement links from test scripts to their parent test cases when you add the test scripts.

## **Before you begin**

Link a Quality Management project to a Requirements Management project.

## **Procedure**

1. Click Administration > Manage Project Properties.
2. At left, in the Properties navigation menu, click Test Script Preferences.
3. By default, you can link single steps or multiple steps to requirements. Linked requirements are shown in the manual test script editor, in the Validates column.

To disable the Validates column and linking to requirements, clear Allow linking between Requirements and Test Script steps.

Linking from test script steps to requirements is disabled.



- When you add existing test scripts to a test case, the test scripts might have steps that are linked to requirements that they validate.

To enable a preference so that the requirement links from the test scripts are automatically copied to the parent test case, select Copy Requirement links into Test Cases when adding new child Test Scripts.

- Click Save.

### 5.7.7 Working with Test Data

Test data is a collection of related data records that supplies realistic data values to variables in a manual test script.

Because data is separate from the test script, test data provides flexibility:

- You can modify test data without affecting the test script.
- You can share test data with many test scripts and users.

	A	B	C	D	E	F
1	Composer:STRING	Item:STRING	Quantity:STRING	CardNumber:STRING	CardType:ENUMERATION	Total:String
2	Haydn	Violin Concertos	1	1234 1234 1234 1234	Visa	15.99
3	Haydn	Violin Concertos	2	1234 1234 1234 1234	Visa	30.98
4	Haydn	Violin Concertos	5	1234 1234 1234 1234	Visa	75.95
5	Haydn	Violin Concertos	10	1234 1234 1234 1234	Visa	150.9
6	Haydn	Violin Concertos	50	1234 1234 1234 1234	Visa	750.5

If you are using a text editor, your data should look similar to the following:

```
Composer:STRING,Item:STRING,Quantity:STRING,CardNumber:STRING,CardType:ENUMERATION,Total:String
Haydn,Violin Concertos,1,1234 1234 1234 1234,Visa,15.99
Haydn,Violin Concertos,2,1234 1234 1234 1234,Visa,30.98
Haydn,Violin Concertos,5,1234 1234 1234 1234,Visa,75.95
Haydn,Violin Concertos,10,1234 1234 1234 1234,Visa,150.9
Haydn,Violin Concertos,50,1234 1234 1234 1234,Visa,750.5
```

When you create test data, you import the data from a comma separated values (CSV) file. To use external data in a manual test, associate the test data with a manual test script. When you associate a manual test script with test data, you can insert variables into the test script, replacing literal values. During test execution, the variables in the manual test script are replaced with data from the imported CSV file. By replacing literal values with variable data, you can use a single script to test multiple data records. Test data can be shared with other team members and offers a powerful way to extend test coverage.

## Test data categories

You can create test data categories to organize test data artifacts. By assigning categories to your test data artifacts, you can easily set up a hierarchical directory structure that will help keep you organized and help you find individual artifacts among the many test data artifacts that you typically manage.

## Creating Test Data

When you create test data, you import data from a comma separated values (CSV) file.

### *Procedure*

1. Create a test data CSV file:
  - a) Open a spreadsheet or a text editor.
  - b) In a CSV file, the first row of data must be the column definitions. Follow this format: ColumnName:TYPE, where type can be STRING, NUMBER, BOOLEAN, or ENUMERATION. If you are using a text editor, type each row of data on a new line and include a comma between each column value.

Note Defining columns is required when manually creating test data in the preceding step. It is optional when migrating manual test scripts that include test data; however, do not include actual data in the first row of a CSV file.
  - b) When you are finished typing the data, save the file. Specify .csv as the file extension, or select CSV (Comma delimited) for the file type when you save.

Note To ensure that text with special characters will be imported correctly, use a text editor, such as Windows Notepad or Microsoft Excel, to convert the CSV file to UTF-8 format.

For Notepad, open the file in the text editor, and click Save as. In the Files of type field, select All Files; In the Encoding field, select UTF-8. Click Save.

For Microsoft Excel, click Save as. From the Action Bar at the top of the dialog, click Tools, and then select Web Options. In the Web Options dialog, click the Encoding tab. In the Save this document as field, select Unicode (UTF-8). Click OK and then Save.
2. Create the test data artifact in the Quality Management application. In the main menu, click Construction > Create > Test Data.
3. Type a name for the test data.
4. Optional: Type a description.



5. Optional: Create and assign categories to group the test data. Test data categories make it easier to find test data by allowing you to sort test data artifacts by the categories you apply.
  - a) To add, edit, or further define test data categories, click the Manage Test Data Categories icon ().
  - b) To set a test data category value, in the list next to a category type, select a value.
6. In the Data File section, browse to the CSV file, and click Open. The CSV file is imported and the data records are displayed in the Data Records section.
7. Save the test data artifact.

## Updating Test Data

You can edit existing test data with the Test Data editor.

### ***About this task***

You can edit existing test data by uploading a new comma separated values (CSV) file, or by opening and modifying the current CSV file. If your test data is used by test scripts, you should not modify or remove columns in the CSV file. You can add and remove rows from test data that is included in your test scripts and the scripts will be automatically updated to reflect the changes to the data.

### ***Procedure***

To edit test data:

1. In the main menu, click Construction > Browse > Test Data.
2. In the Name column, click the test data that you want to update. The Test Data editor opens.
3. Do any of the following things:
  - a) To upload a new CSV file, click Browse and select the file to upload.
  - b) To open and modify the original CSV file, click the name of the file to open the file in an editor of your choice. Save your changes to the file and then click Browse to upload the updated file.
  - c) To add, edit, or further define test data categories, click the Manage Test Data Categories icon ().
  - d) To set a test data category value, in the list next to a category type, select a value.
4. Click Save.



## ***Results***

The new records are displayed in the Data Records section.

## **Using Test Data in Manual Test Scripts**

To use external data in a manual test, you need to associate test data with the manual test script. After you establish this association, you can use the variables in the test data to replace literal values in the manual test script.

### ***Before you begin***

Before completing these steps, create test data and associate a test case with a manual test script.

### ***Procedure***

To use test data in a manual test script:

1. From the Test Scripts section of the test case, open the test script by clicking the name. The test script opens in the Manual Test editor.
2. From the Test Data list, select the test data that you want to link with the manual test script, and then click Save. This action associates the test data with the test script. When you have a large number of test data artifacts to select from, if you have created test data categories and applied them to your test data artifacts, you can group the results in the Test Data list by selecting a category type in the Group By list.
3. Insert the test data variable into the manual test script:
  - a) Click the step where you want to add a test data variable, and then move the cursor to the location for inserting the variable.
  - b) From the toolbar, click the Insert Test Data Column icon (Insert Test Data Column).
  - c) Select the variable, and click OK. The variable is inserted at that point in the step.
  - d) Repeat to add additional variables.

## ***Results***

During test execution, the literal values are replaced with the test data.



## 5.8 Managing Lab Resources

You can track the resources in your lab using the lab management tool. You can create data for physical machines and virtual images, search for resources with specific configurations, and manage requests. If you have administration permissions, you can create reservations for lab resources. If you are integrated with an external provider application with virtualization capabilities, you can work with virtual machine and image collections. If you are integrated with provider applications that provide deployment and inventory capabilities, you can update lab resource data to display the latest lab resource configurations and new lab resources that have been added, and you can run scripts, projects, or libraries that are configured in the external provider applications on remote lab resources.

The test lab management tool helps you manage the lab resources in your testing labs by providing a way to track your resources and administer lab-resource reservations and requests.

Test lab resources include anything that is used as part of running a test. For example, lab resources include the following:

- Physical machines – computers that are used for running tests
- Virtual machines – a virtual machine that hosts virtual images for running tests
- Virtual images – images that are set up with software for running tests
- Test environments – a set of configuration attributes, such as operating system and test execution software, that you define for running tests
- Test cells – a collection of lab resources, such as physical and virtual machines with specific hardware and software configurations

Lab resources can be defined with a specific make and memory capacity, a certain version of an operation system and browser, and other hardware and software attributes.

### Lab management roles and users

The test lab manager and testers carry out various lab management activities.

The test lab manager oversees these lab functions:

- Installing or coordinating the installation of software onto test lab resources
- Setting up and maintaining lab resources for the entire lab
- Managing requests for lab resources

The test lab manager uses the lab management tool to define the lab resources.

Testers request lab resources from the lab manager or reserve the resources directly. Then testers use software products under development by running tests on the lab resources.

## Creating and Viewing Lab Resources

You can populate the lab management database with lab resource information in several ways. You can import existing lab resource data and create new lab resource data in the lab management tool. You can create groups of lab resources and assign them to teams of testers. You can search for lab resources with specific attributes, for example, specific operating systems and software.

## Roles

Lab management tasks are often performed by people in the roles of lab manager and tester. This topic describes work activities that are associated with these roles and the features in the lab management tool that they use.

The role that your lab manager administrator associates with your account determines the options that are available in test lab management and the functions that you can access in the lab manager editors.

User role	Tasks
Lab manager	<ul style="list-style-type: none"><li>• Manages lab resources in test labs. The lab can include physical machines, virtual machines, and virtual images.</li><li>• Works with team managers to allocate lab resources across the products under tests.</li><li>• Installs, sets up, and maintains software applications for the entire lab.</li><li>• Collaborates with deployment engineers who deploy builds to test environments.</li><li>• Ensures that the lab resources are set up in the necessary topologies with the required operating systems and middleware.</li></ul>
Software tester	Tests software that is under development. Runs tests on lab resources.

The following types of tasks are typically associated with these test lab roles.

### 5.8.1 Customizing Lab Resource Properties

Lab resource properties are types, root types, subtypes, and attributes that define lab resources such as machines, virtual images, or software in IBM Rational Quality Manager.



## About this task

In the lab management sections of Rational Quality Manager, you can display or hide lab resource properties. Click the Admin (Administration) icon in the upper-right portion of the banner and select Manage Project Properties.

In the lab management feature, examples of lab resource types are Machine and Operating system. These are examples of root types that can contain subtypes. Both root and subtypes can contain attributes. Attributes are fields that are displayed in the user interface and that can hold values for a lab resource type. Most attributes are associated with root types. However, you can also add attributes to subtypes and to new types that you define.

If you add new attributes to a root type, those attributes are also displayed for any subtypes that are associated with the root type. When you hide a type or an attribute, it is no longer displayed. However, you can view and restore hidden types and attributes.

**Note** When you modify lab resource properties, the changes that you make are at the project area level, not at the server level.

To customize lab resource attributes:

### Procedure

1. Log on as a user with JazzProjectAdmins repository permissions.
2. Click the Admin (Administration) icon in the upper-right portion of the banner and select Project Properties.
3. Click Lab Resource Properties.
4. To view lab resource attributes and types that have been removed from the user interface, select Show hidden lab resource attributes and types.
5. To add a new lab resource type, in the Types column, click the Add icon (Add).

**Note** Types that you define must be root types; the new type can have attributes and can be used as an attribute for other types and subtypes.

6. Click New Type 1, and type the name of the new lab resource type.
7. To add attributes to the new type, click the Edit icon (Edit). The Manage Attributes window opens.
  - a) Click the Add icon (Add).
  - b) Under Attribute Name, type a name for the attribute.

- c) Under Data Type, select the data type for the attribute. The options are: String, Number, Multiple Values Type, Type, and Date. If you select String or Number, you can add enumerated values, for which you can create options that are selectable in the user interface by a drop-down menu. To add or edit an enumerated value, under Value, click Add Enumerated Value. In Value, type a value for the option to be displayed in the user interface. To add more values, click the Add icon (Add). If you select Multiple Values Type or Type, under Value, you can select an existing type.
  - d) Click OK.
- 8. To add a new subtype to an existing type, for example, to add "Chrome" as a browser under Installed Software, under Type, select the type, click the subtype in the next column (if appropriate), and then click the Add icon (Add).
  - 9. Click the new, highlighted field and type the name of the sub type.
  - 10. To hide types or attributes so that they are not displayed in the user interface, select the type or attribute, and then click the Hide icon (Hide lab resource).
  - 11. In the Project Properties editor, click Save.
  - 12. Log out of Rational Quality Manager, and then log back in again. The types and attributes that you created are displayed in the user interface where you can edit lab resources.

## 5.8.2 Creating New Test Environments

You can create a test environment for lab resources, which can be stored and reused in many lab management tasks, for example, lab resource requests, searching for lab resources, and creating future test environments. A test environment is a set of lab resource attributes, for example, the make, memory, operating system, and software for a particular type of machine.

### Procedure

To create a test environment:

- 1. In the main menu, click Lab Management > Create > Test Environment. The Create Test Environment editor opens.
- 2. Click <Enter New Test Environment Name> and type a name for the test environment. This is a required field.
- 3. In Description, click <Click here to enter description> and type a brief description of the test environment.
- 4. Under Summary, click Add content to type a detailed description of the test environment. In this section of the editor, you can also attach a diagram or document. Click Preview to close the text editor.
- 5. Under Related Channels, click Add channels to add channels to the test environment. To remove them later, click Delete channels.



6. Under Coverage, in Lab Resource Descriptions, click Enter a label for the Lab Resource Description 1 and type a name for a lab resource that is part of the test environment.
7. In Type, select a lab resource type, for example, PhysicalMachine.
8. Click the Add Criteria icon (Add attribute). The Select Attribute window opens, showing fields of information that you can include in the lab resource definition.
9. To select items one at a time, click the item, and then click Add. Repeat until you have entered all the fields you need, and then click Add and Close. To select multiple items and add them all at the same time, press and hold the Control key, and select the items, and then click Add and Close. These fields are added to the lab resource description and are displayed under Type. Enter information for each of these fields as needed.
10. To add an additional Lab Resource Descriptions section, in the upper right corner of the Lab Resource description section, click the plus sign icon (Add). You can add as many lab resources as you want to the test environment. To delete a Lab Resource section, click the X icon (Remove).
11. At the top of the page, click Save.

Note: When channels are enabled in the project, saving changes to a test environment's lab resource descriptions will prompt the user to synchronize the test environment with channels. Channels are subsets of test environments based on common lab resource properties. For example, imagine creating two new test environments, both with Windows 7 Enterprise as an operating system and x86-64 CPUs. If a channel exists in the project for Windows workstations with 64-bit CPUs, these two environments would be linked to that channel once they were synchronized. After, when selecting one of these test environments to run a test case execution record, the x86-64 Windows workstation channel would be available for use.

12. If one of those workstations had its operating system or processor changed, after which channel-test environment relationships were synchronized, that channel would no longer be available when running a test case execution record on that environment.
13. Optional: Manually update the test environment-channel relationship at any time by clicking Synchronize test environment with channels (Synchronize). This relationship will persist until channels and test environments are synchronized again.
14. In the main menu, click Lab Management > Browse > Test Environments. The test environment you just created is displayed in the list of test environments.

## Results

This test environment is now available to use when defining a new lab resource, searching for resources, and creating requests and reservations.

### 5.8.3 Viewing Test Environments

You can view all the test environments that are available to be used in lab management activities such as creating resources, searching for resources, and requesting and reserving lab resources.

#### Before you begin

To view all test environments:

#### Procedure

If 1. In the main menu, click Lab Management > Browse > Test Environments. All the test environments that are available in lab management are displayed.

2. To view details about a test environment, click a test environment name.
3. To create a template for a new test environment based on one of the test environments, select the test environment check box, and then click the Copy icon (Copy).
4. To see a preview of a test environment, select the test environment check box, and then click the Preview icon (Preview). A summary of the test environment is displayed at the bottom of the editor.
5. To create a request for a lab resource that is based on a test environment, select the test environment check box, and then click the Create Request icon (Create request).
6. To search for lab resources that use one of the test environments, select the test environment check box, and then click the Search icon (Search).
7. To create a test cell that is based on a test environment, select the test environment check box, and then click the Test Cell icon (Test cell).
8. To delete a test environment, select the test environment check box, and then click the Remove icon (Remove).
9. To synchronize a test environment with channels, select the test environment check box, and then click the Sync the environment with channels icon (Synchronize).

Note: Channels are subsets of test environments based on common lab resource properties. For example, imagine two test environments, both with Windows 7 Enterprise as an operating system and x86-64 CPUs. If a channel exists in the project for Windows workstations with 64-bit CPUs, these two environments would be linked to that channel once they were synchronized. After, when selecting one of these test environments to run a test case execution record, the x86-64 Windows workstation channel would be available for use.

If one of those workstations had its operating system or processor changed, after which channel-test environment relationships were synchronized, that channel would no longer be available when running a test case execution record on that environment.



## 5.8.4 Creating Data for Physical Machines

You can create lab resource data for physical machines to track in the lab management tool.

### Procedure

To create lab resource data for a physical machine:

1. In the main menu, click Lab Management > Create > Machine. The Create Machine editor opens.
2. At the top of the editor:
  - a) Click <Enter New Machine Name>, and type a host name or IP address for the machine.
  - b) In Owner , select an owner for the machine.
  - c) In Operational Status, select a status, for example, Available.
  - d) In <Click here to enter a description>, type a description of the machine.
3. In the General Info section:
  - a) In Type, select PhysicalMachine.
  - b) In Location, type the location of the machine.
  - c) In Administrative Status, select a status, for example, Enabled.
  - d) To set up a connection to a remote machine, expand Remote Connection and then click the Add icon (Add).
  - e) In Remote Connection, select the type of connection. The options are Remote Desktop, VNC, ftp, http, and telnet.
  - f) In URL, click the Edit icon (Edit) to enter or edit the URL for the remote connection.
  - g) To add additional remote connections, click the Add icon (Add).
4. Point to Hardware and expand the section. Provide information in as many of the fields as you can.
5. Point to Operating System and expand the section. Provide information in as many of the fields as you can.
6. Point to Software and expand the section. Provide information in as many of the fields as you can.

Note If the lab resource that you are defining has attributes that do not display in the Create Machine editor, ask the project administrator to add the attributes that you need to the lab resource properties in Admin > Manage Project Properties > Lab Resource Properties.

7. At the top of the Create Machine editor, click Save. The new physical machine definition is added to the test lab manager database and the new machine name is displayed at the top of the editor.



## 5.8.5 Creating Virtual Images

You can create data for virtual images to track in the lab management tool. A virtual image is a .vmx file that consists of software to be used as a basis for applications under test. For example, you could set up a base image .vmx file consisting of a Windows XP, SP2 operating system and security software for the English language. You would then assign it a name (for example, WinXP-SP2-eng.vmx) and save it. When you deploy the image onto a physical machine and rename the image, it becomes a virtual machine. You could then load an application for testing onto the virtual machine. Virtual images can easily be reused, which can save time setting up additional testing environments.

### Procedure

To create a virtual image:

1. In the main menu, click Lab Management > Create > Virtual Image. The Create Virtual Image editor opens.
2. Click <Enter New Virtual Image Name>, and type a name for the virtual image.
3. In Owner, select a person to own the virtual image.
4. In <Click here to enter a description>, type a description of the virtual image.
5. In the General Info section, provide information in as many fields as you can. Refer to the context-sensitive Help for more information about these fields.
6. Expand Operating System and provide information in as many fields as you can.
7. Expand Software and provide information in as many fields as you can.
8. At the top of the Create Virtual Image editor, click Save. The new virtual image definition is added to the test lab manager database and the new virtual image name is displayed at the top of the editor.

## 5.8.6 Creating and Viewing Test Cells

Test cells provide a way to conveniently group together a set of lab resources that describe a test environment. For example, a test cell might include an application server, database server, a client desktop computer, and a computer running the correct adapter to execute tests. You can reserve all the lab resources in the test cell for any duration that is not already reserved by another user. When you create test cells, base them on specific test environments and the type of test execution you use, especially if you choose to create test execution schedules.

### Before you begin

Choose a test environment on which to base the test cell.



## Procedure

To create a test cell:

1. In the main menu, click Lab Management > Create > Test Cell. The Create Test Cell editor opens.
2. Click <Enter New Test Cell Name>, and type a name for the test cell. Give the test cell a name that reflects the test environment on which the test cell is based.
3. In Type, select the type of lab resources that the test cell will contain. You have two options:
  - a) Machine
  - b) Virtual Image
4. In Owner, enter the name of the person who is responsible for the test cell.
5. Click <Click here to enter summary information>, and type a description of the test cell. You could type the name of the test environment on which the test cell is based and information about any execution schedules that could use this test cell.
6. Expand Description and type a detailed description of the test cell. Click Preview to close the text editor.
7. In Lab Resources, click Test environment (Configuration), select a test environment, and click OK. If any lab resources have the attributes that are specified in the test environment, the lab resource information displays those resources.
8. Click the Assign a Lab Resource to each row icon (Assign lab resource to each row). If there are lab resources that meet the attributes that are defined in the test environment, the lab resources are displayed under Resource Name.
9. Expand Machine Availability to reserve the test cell.
  - a) In Reserve from, select a future date.
  - b) In Reserve to, select a date later than the Reserve from date.
10. Notice that the graph displays the dates that you selected.
11. Click Save. The test cell that you created is reserved for the selected dates and is available for executing test cases, test suites, or test execution sequences. When you run a test script, test case, test suite or test execution schedule, after you click the Run icon (Play), you can select a test cell to use for test execution.
12. In the main menu, click Lab Management > Browse > Test Cells. The test cell that you just created is displayed in the list. You can deploy or undeploy test cells from this page.

### 5.8.7 Managed Virtual Images and Machines

Managed virtual images and managed virtual machines are logical groupings of either virtual images or virtual machines that represent application topologies. An example of managed virtual images could be a collection of one virtual image for a web server and another virtual image for a database server that are used together. Managed virtual images and managed virtual machines can have test environments associated with them that describe the application topology that they represent.

**Note** You must be integrated with an external provider, such as VMLogix or Surgient or another tool that manages virtual images and machines, to view or use managed virtual images or virtual machines. For information about setting up the integration of the external provider application and IBM® Rational® Quality Manager, see the product documentation for the external provider application.

Managed virtual images and managed virtual machines are synchronized from an external provider. You cannot manually create them. The external provider applications provide information about the virtual images and virtual machines that is visible in Rational Quality Manager. The user can create a test environment for managed virtual images or managed virtual machines. The test environment describes what is created if a managed virtual image is deployed and virtual machines are created from the images. For example, users could see a managed virtual image imported into Rational Quality Manager, that contains a virtual image that they recognize as their web server test environment. They could then create a test environment that describes their web server and add it to the managed virtual image to identify it for other users.

You can view and work with managed virtual images and managed virtual machines by pointing to the Lab Management (Lab Management) and clicking Browse > Test Cells.

### 5.8.8 Creating Lab Resource Groups

You can create a group of lab resources to track and manage. You can associate teams of people to a lab resource group. For example, you can create a group of machines that are dedicated to a particular testing team. The team of people always has exclusive access to the group of lab resources that you define.

#### Procedure

To create a group of lab resources:

1. In the main menu, click Lab Management > Create > Lab Resource Group. The Create Lab Resource Group editor opens. Note This is a required field.
2. Click <Enter New Lab Resource Group Name>, and type a name for the lab resource group.



3. Enter a start date and an expiration date for the lab resource group or select the No Expiration check box for the lab resource group to be permanent.
4. In Team Area, select the name of a project team. Note This is a required field.
5. In <Click here to enter a description>, provide a description of the lab resource group.
6. Click Save to save the definition of the lab resource group. If there are lab resources that are already defined for the team you selected, those lab resources are displayed in the Lab Resource section. You can filter the display of lab resources by selecting a category in Group by.
7. To add lab resources for the group, in the Lab Resource section, click the Add to group icon (Add). The Add to group window opens and displays a list of all the available lab resources.
8. Select the lab resources for the group. You can select one, more than one, or all of the lab resources.
9. Click OK. The lab resources are added to the lab resource group definition.
10. To remove a lab resource from the group definition, select the lab resource, and click the Remove from Group icon. The lab resource is removed from the group but is still in the lab management database.
11. Click Save. The lab resource group is saved and displays in the All Lab Resource Groups editor. To open the editor, go to the main menu and click Lab Management > All Lab Resource Groups.

