Test Management with Rational Quality Manager 4.0

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 key processes 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.”

2.1.1 Defining Test Management

Test management is project management done in accordance with professional standards and practices. The ISTBQ 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 and 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

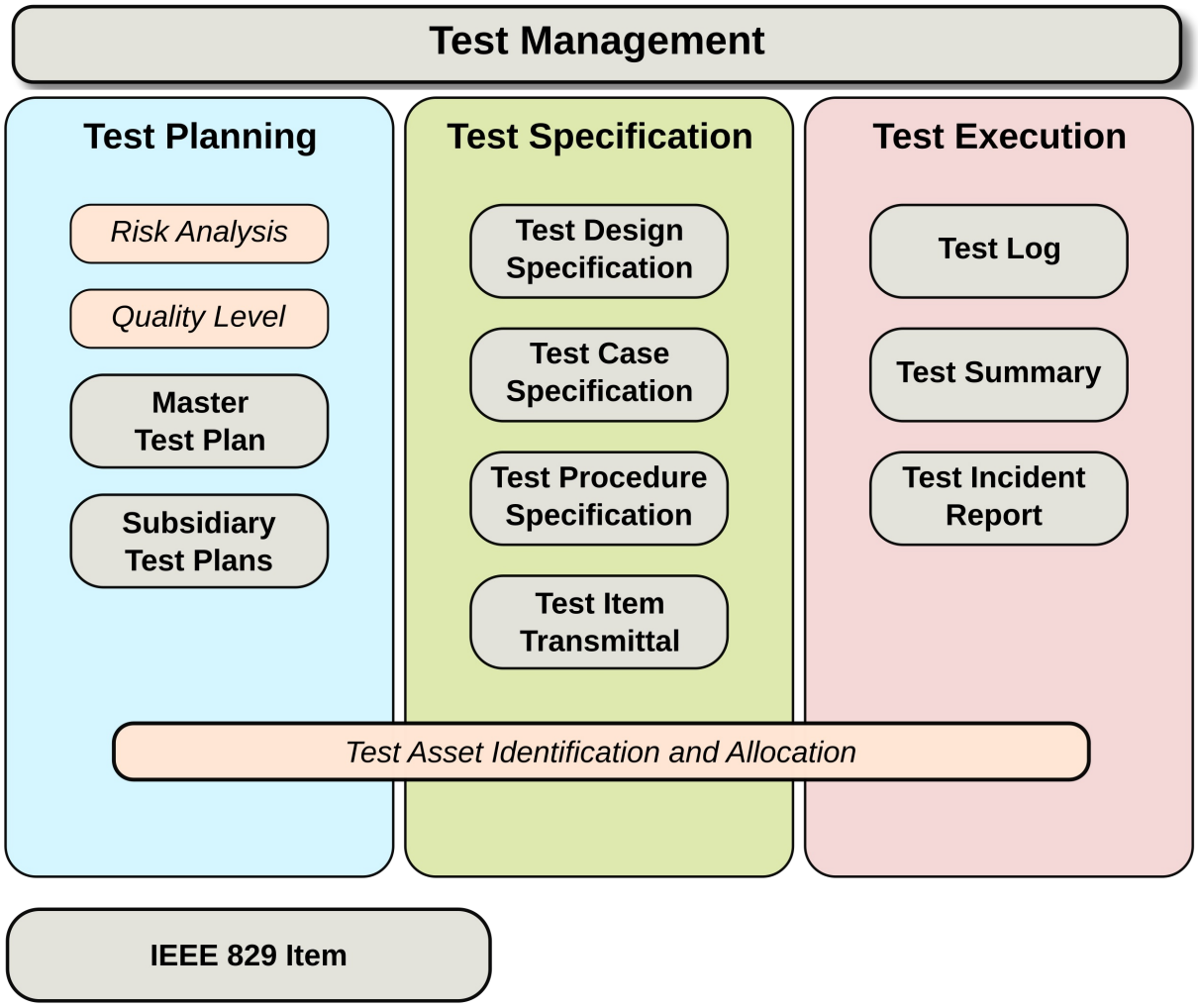
Obviously these topics are out of scope for what we will be discussing in this overview but it is important to keep in mind that being a good test manager does involved more than being able to manage the processes described in this module – in the same way that a good project manager also has to deal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

Bjarne Strourstrup, 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 Software and System 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 the Rational Quality Manager 4.0 uses the terminology and concepts of the IEEE 829 standard so we will stick with that.

The 829-2008 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 information on 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 the 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 described 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 some form. 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 about topics like: how much testing we should be doing, to what level of rigour or detail our testing should be done to, and how we know that we are finished testing.

There are a couple of standard answers to the question: *"When do stop testing software?"* These 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.

All of these are symptoms of a bad testing process. There is only one right answer: *We are finished testing when then testing has met the quality control goals.*

For example, it seem 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.

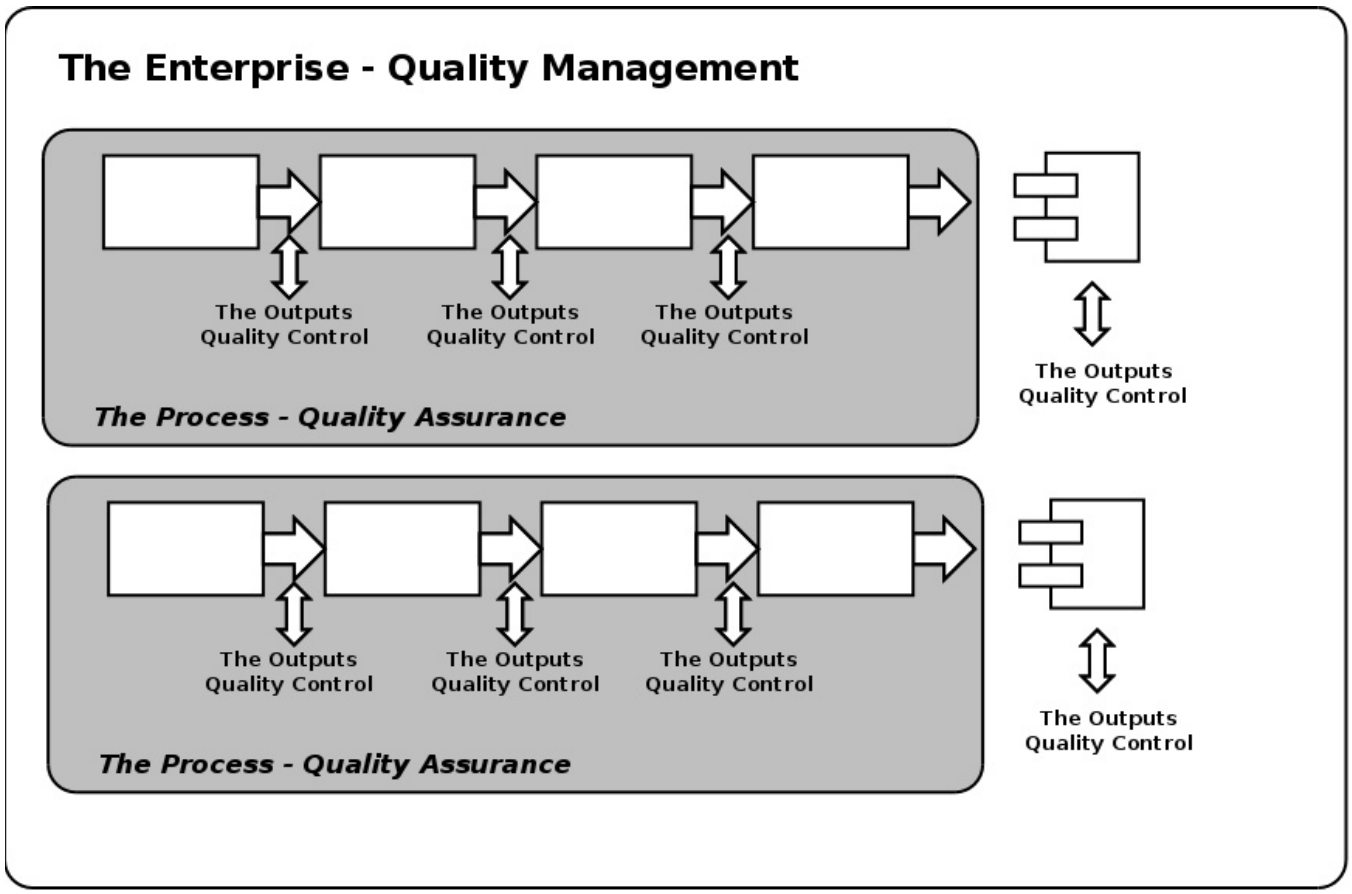
2.3.1 Defining Quality

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 (the *"That's the way we do it here"* beliefs of a company) with regards to quality 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 role of quality management is to identify what the expectations of the organization are in terms of quality and then communicate those to stakeholders in terms of a quality mission. The quality mission is a framework and set of principles for people in the organization that guides their decision making process on what to do about quality in their area of responsibility and in the various processes they participate in. Quality management is the systematic way of ensuring that organized activities happen the way they are planned 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 will be using, specifying when we actually do the testing, identifying 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, is 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 the 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 perception, suppose in the example on the next page that the above cell phone plan is offered by a wireless phone company. The quality of the plan, and ultimately the decision to get 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 important theme of the work represents 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 the stakeholders expect from a product at a specific time.

I text all the time
I visit family in Europe often
My son needs a phone

Good Quality

Attributes

Unlimited Texting
Free International Roaming
Add a free family member

Cell Phone Plan

Inever text
I never travel outside the country
I have no family

Poor Quality

Quality, in this view, is a combination of the attributes of the product as 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 an appropriate level of quality for a particular project means at a specific time and within a specific context. These decisions in turn determine the metrics and processes that we use to achieve quality that is “good enough” for that specific set of circumstances.

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

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 outweigh its noncritical problems, and it would cause more harm than good to continue improving

2.3.3 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.

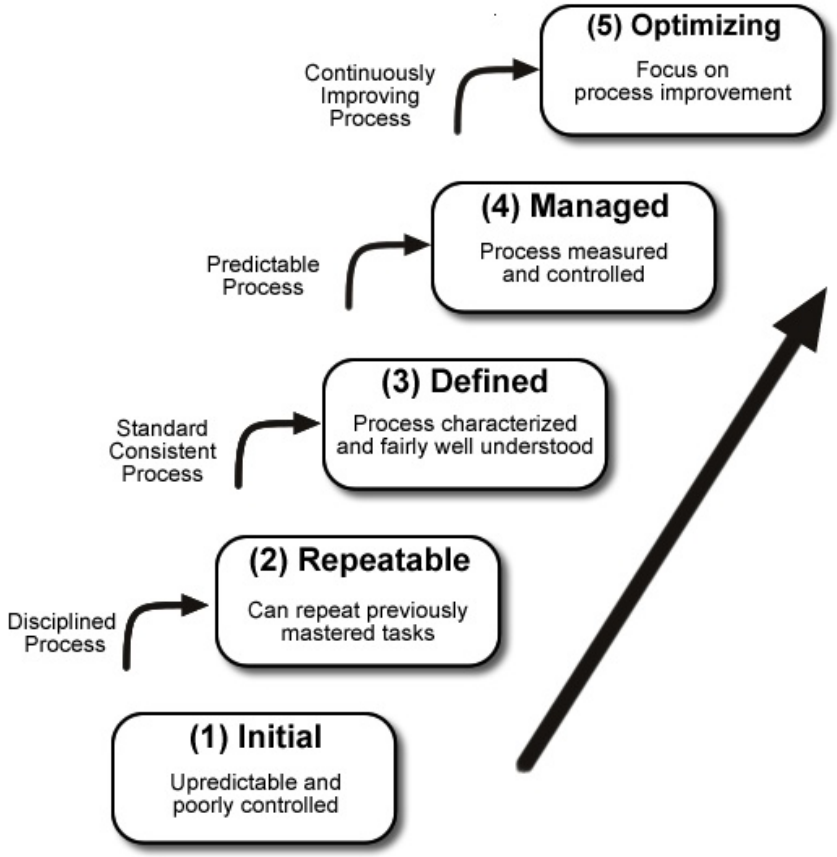
What we see here is an evolving of the testing activity from a rote "run tests, record result" to a careful consideration of choosing testing activities and methodologies that result in the last two phases where the focus is on the quality of the software. But what about the part about "not much testing 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 and they mature in their testing activities.

2.4 Process Maturity

One of the limitiations 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 suppose 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 be 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 a lot of the literature, this is often made to sounds like some dystopian, chaotic free for all. However when you take a look at a lot of organizations at this level we can separate out two basic types.

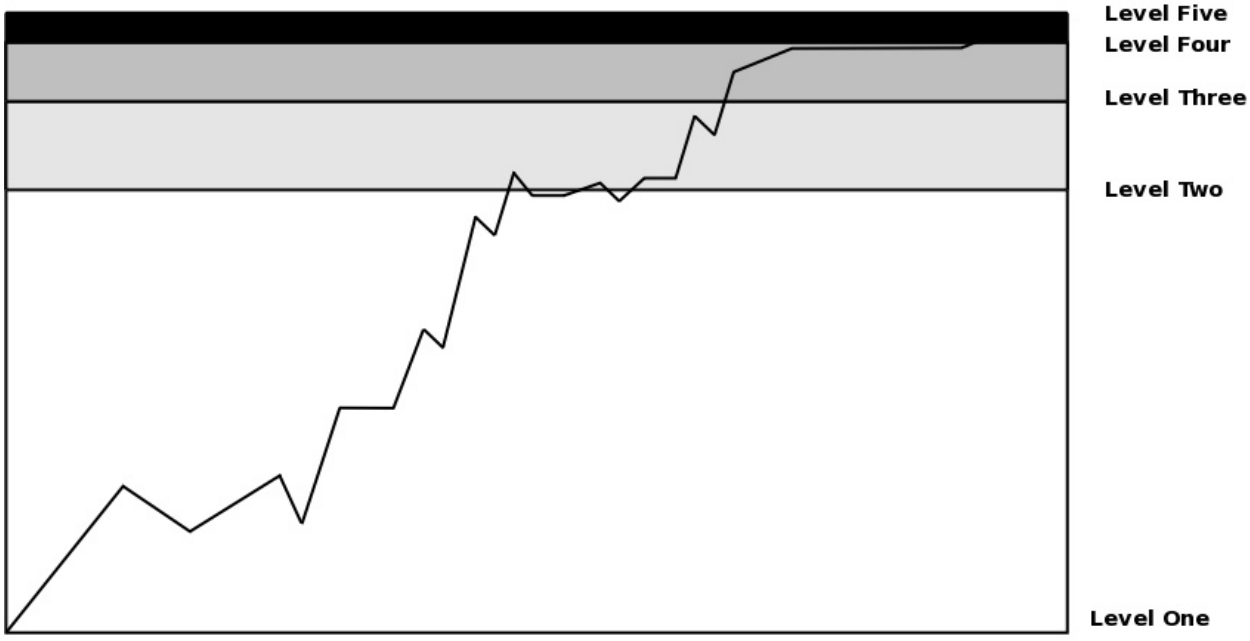
The first is the start up where the emphasis is on the personal skills and productivity of a few key people. It's often counterproductive to have a software development process when there is only two people – the programmer and someone running the business side.

The second 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.

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 reason is that 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.

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 implementations, however what really defines a level 3 maturity is that the process is institutionalized. What that means is that it is seen as “the way we do things around here” and is enforced by the people 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.

The result of having a standardized process is that the organization is able to start collecting data on their process – identifying what works and what doesn’t. And having a standardized process (not just defined by actually in day to day use by everyone) allows QA to actually be done effectively.

If you think of the organization like a basketball team, level one 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 two, each individual player has to master the skills of dribbling, passing, shooting, defending as well as learn the rules of the game. At level three, we can take those five level two players and integrate them into a team.

Level Four: Proactive Quality

Once an organization has been a level 3 for a while, it can start to prevent defects from occurring by developing best practices and establishing a review culture. For example is we know from experience that certain design decisions result if certain kinds of errors, we 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 experiences and what they have learned from previous projects.

The reviews follow naturally from having these standards or best practices. When work is done, it is reviewed to ensure it has 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.

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. Level 5 can be thought of as work being reviewed and tested as it is being done. The example I use is of a spell checker. Level 4 is analogous to running a spell checker on your document while level five is like having the spell checker identify misspellings as you type and offer on the fly suggestions.

Another characteristic of level five is that this integration of best practices into the development activities allows the process to improve just through using the process.

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 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 Behavioral 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 practiced. 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, subgoals, 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 subphases 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 prod-

ucts, 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, catalog, 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]. Actions 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 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/

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

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.

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. 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: 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.

2.6 Lab Management with RQM

This section turns to the more practical side of test management, the definition, identification and allocation of lab resources. The reality of testing is that we have to run tests in a test environment. In many cases we may be required to run tests in a variety of different environments and configurations. This environments are lab assets.

a A lab asset can be any number of things – a physical machine, a virtual image, a specific physical location – but usually we define a configuration or a set of these assets to produce a test environment. Of course the allocation of lab assets is a classic project management problem. As part of test management, RQM has the facility to support this aspect of test management.

2.6.1 Some Terminology

Roles

There are two primary roles in the test lab.

Lab managers

Lab managers typically manage resources in test labs which include physical machines, virtual machines, and virtual images, and lab managers can work with team managers to allocate lab resources across the products under test, and to install, set up, and maintain software applications across the entire lab. The lab manager role is for a team member who deploys builds to test environments and who ensures that the lab resources are set up correctly.

Testers

Testers simply run tests on lab resources. They re the consumers of lab resources.

Lab Resources

Lab resources are the individual machines, operating systems and other pieces of hardware and software needed for testing.

Test Environment

A test environment is a description of a set of lab resources that provide the needed capability for testing. For example, a Linux Fedora 21 operating system running on an Intel 32 bit environment using FireFox 54 as a browser and running Java SE version 7,

Test Cell

A test cell is a set of lab resources that provide a specific test environment – or another way to think of it is a collection of physical machines, images and other resources that bundled together are a real physical implementation of a test environment.

In the lab we will work with these specific concepts as well as making requests for resources, fulfilling requests and reserving resources.