UNIVERSITÄT LEIPZIG

Fakultät für Mathematik und Informatik
Institut für Informatik

Thema:

Softwaretests – Der Weg vom Konzept bis zur Testdurchführung bei Individualsoftware

Theme:

Software tests – the way from concept to test execution for custom software

– Diplomarbeit –

eingereicht im:	Januar 2009
von:	Qingli Liu
Studiengang:	Informatik Diplom
Hochschulbetreuer:	Prof. Dr. Hans-Gert Gräbe
Firmenbetreuer:	Dipl. Inf. Karl-Friedrich Bach

IBM Deutschland Enterprise Application Solutions GmbH
Pascalstrasse 100
70569 Stuttgart

Declaration

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been quoted or acknowledged.

Date: …… Signature: ……

Qingli Liu

Abstract

In this thesis a high-level theoretical analysis and understanding of software test will be described through analyzing its characteristics in different types of software development process models and outlining a static infrastructure consisting of relatively complete key aspects for managing and performing software test. Then new concept named Virtual Testcenter, which is template-based, would be analyzed and developed to realize that the basis environment can be made to ensure fast start of software test projects, which can be controlled and managed centrally as well. The goal of Virtual Testcenter is to define the requirements of software test projects, so a survey would be performed based on a catalog of criterions or influence factors, which define typical project situations, then the results of the survey from the project praxis will be analyzed to conclude the requirements on Virtual Testcenter, where the high-level theoretical analysis, IBM best practices, is-analysis results of the available IBM testing Assets and the property of custom software have also been taken into account. Two different scenarios of the requirements for small and large projects will be defined separately, and accordingly two different solutions including specific process models, service components, and hard- and software infrastructures for these two scenarios will be analyzed and described. Virtual Testcenter will function as a new effective and efficient testing solution or service for IBM or external customers.

Acknowledgements

I would like to thank Dr. Frank Hollenberg for excellent support, advices, feedback and information during this thesis. I would like to thank Oliver Kerschhaggl, Ralf Trübenbach and Torsten Welk for technical questions and information about IBM Rational tools. I would like to thank any other colleague of the department named IBM Rational Center of Competence & Technical Solutions for very friendly help and support for this thesis.

Contents

1 Introduction

1.1 Motivation

Today there are different software development process models used in different projects, such as sequent/linear models, agile models and incremental models, every model has its own characteristics, so software test, which is the part of each development model, has also its own characteristics.

Software test cannot be isolated from development process models and plays a very important role for ensuring software quality. With the high requirements of software products, the complexity of software keeps increasing, but the duration of some tasks like module development or module test can be very short, as should be considered in the planning of tests today.

At the present time test services in GBS (Global Business Services) are always carried out newly each time according to specific projects, as is not very efficient, so in order to bundle the activities and support test projects centrally, *Virtual Testcenter* should be created, which makes it possible to provide these test projects with the fast-start and the central management, so the elaboration of a concept for Virtual Testcenter concerning to requirements, processes and tools is to be described and performed, which functions as a solution particularly for testing custom software for IBM or external customers.

1.2 Aim and objectives

In this thesis the infrastructure and models of testing software and different test concepts would be studied and described. The characteristics of software tests in different types of software development models would be discussed to give a high-level understanding of software tests. The characteristics and focal points of selecting and developing custom software are discussed. Some concepts will be especially described for custom software based on its properties.

Template as a concept or technology should be elaborated for software tests so that the quality, effectiveness and efficiency of software tests can be improved. Template could regard all the aspects of software tests and be of many types including documents, conceptual models, physical data stores and so on.

New concept named Virtual Testcenter, which is template based, should be analyzed and developed to realize that the basis environment can be made to ensure fast start of software test projects, which can be controlled and managed centrally as well.

The goal of Virtual Testcenter is to define the requirements of projects for two aspects: One for processes, which are used and suited for customers or project situations; the other for tools, which support above required processes with the form of Quick Start Templates, hosting and consulting.

Requirements on Virtual Testcenter for different projects should be different, so a catalog of criterions or influence factors, which define typical project situations, would be made, then based on this catalog a survey would be performed from the project practice

and from its results customers' concrete requirements on Virtual Testcenter will be got and summarized, the general requirements would be defined and two scenarios about other requirements on Virtual Testcenter for small or large projects would also be defined differently.

Through is-analysis of available IBM assets, which support software test processes based on IBM Rational Tools, concerning to the following questions:

- What does each solution accomplish?
- Are they generally valid or which part of them is generally valid?
- What is the difference between them?
- Which part of software development process do they cover?

In the is-analysis not only actual problems but also important advantages or best practices will be summarized for the late elaboration of Virtual Testcenter.

Then a concept for Virtual Testcenter will be elaborated. In the concept the hard- and software infrastructure, process models, service components and so on would be analyzed, defined and described (and also later implemented in the practice) with the emphasis on testing custom software. Two different well suited solutions, both of which would satisfy the general requirements, should be provided for each scenario of specific requirements on Virtual Testcenter (small projects or large projects). In particular IBM Rational Tools should be used here.

1.3 Environment

1.3.1 About IBM in Deutschland

International Business Machines Corporation (abbreviated IBM, nicknamed "Big Blue") is a multinational computer technology and consulting corporation headquartered in Armonk, New York, USA. IBM has been known through most of its recent history as the world's largest computer company; with over 355,000 employees worldwide, IBM is the largest information technology (hardware, software, services) employer in the world. It has engineers and consultants in over 170 countries and revenue $98.8 billion USD (2007).

IBM holds more patents than any other U.S. based technology company and has been Nr. 1 for 15 years. Its research has eight laboratories all over the world.

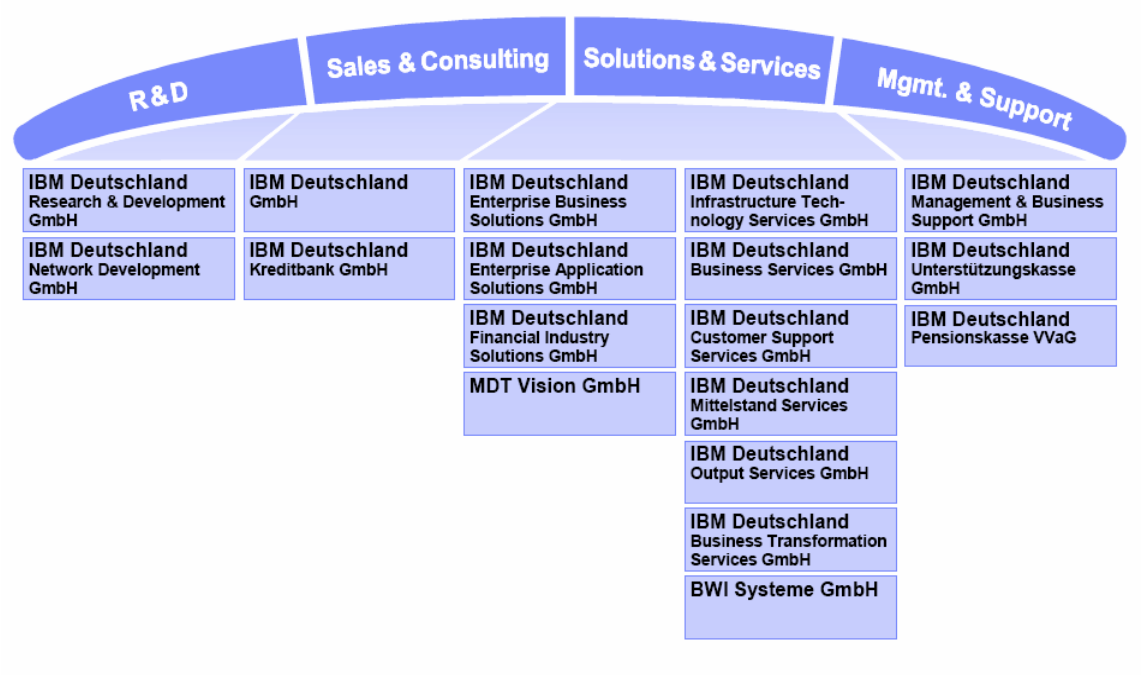


Figure 1.1 Structure of organization of IBM Germany since 01.July.2008

IBM in Deutschland has been newly aligned structurally since 01 July 2008 and has 21,500 employees in 18 GmbHs, and this alignment has been made along the chain of additional value and classified into the following four core competences (see figure 1.3.1) [IBMGermany]:

- Research & Development

2 GmbHs are engaged in research and development tasks.

- Sales & Consulting

2 GmbHs are set up for sales and consulting competences, which include the whole consulting business and sales of IBM products and services of IBM in Deutschland and also maintenance and consulting services. Branch expertise for all the core industries in Germany and focus on middle class are outstanding for this core competence.

- Solutions & Services

11 GmbHs are engaged in providing services for customers. Relatively more GmbHs in this core competence than others have proven that IT-based services play a more and more important role in IBM. These firms belong to *Global Technology Services* (GTS) or *Global Business Services* (GBS).

- Management & Business Support

3 GmbHs are responsible for management and business support functions, e.g. finance, personnel, law, marketing , communication and so on.

1.3.2 About IBM Deutschland Enterprise Application Solutions GmbH

IBM Deutschland Enterprise Application Solutions GmbH (IBM Deutschland EAS) is one of the 11 GmbHs in the Solutions & Services competence of IBM in Deutschland. As leading System Integration Provider IBM Deutschland EAS focuses on design, de-

velopment and supervision of individual applications and assets to support the customers in their market. The foundation therefore is the innovative solutions and the newest technologies and legacy integrations and transformations.

IBM Deutschland EAS is the service provider accompanying through all the phases of complex projects, particularly right around IBM software products und realize Client Value for customers, which has been conceived together with Sales and Consulting – in particular as integral component of the global Delivery-Model of IBM.

The main competences of IBM Deutschland EAS:
- Planning and execution of system integration projects
- Design, development and maintenance of applications for the whole enterprise
- "Early adaptor" services for newest technologies
- Services right around IBM software products
- Integration of web technologies, SOA and Information on demand solutions

Skills provided by IBM Deutschland EAS:
- Web technologies
- Legacy integration
- Content management
- Business warehouse
- Test methods & tools
- IBM software products
- Web design

Professions of IBM Deutschland EAS:
- Project management
- IT specialist
- IT architect
- IT consultant

There are 1587 employees in IBM Deutschland EAS (July 2008), who work in more than 20 cities or locations in Germany to provide convenient and efficient Services for customers.
This thesis is written in the department named Rational Center of Competence (CoC) & Technical Solutions in Schwentinental, which is led by Dr. Frank Hollenberg .

1.4 Structure of this thesis

This thesis consists of 5 parts: introduction, theoretical analysis for software tests and custom software, is-analysis of available IBM assets and IBM testing services, elaboration of a concept for Virtual Testcenter and realization of the concept through tools.

This first chapter presented an introduction to this thesis, providing the motivation, aim and objectives and environment where this thesis is written.

The second chapter introduces theoretical basis to have a high-level understanding of software tests through analyzing software test in different available software development process models. Many aspects and concepts of software test are also described. A very important point of this chapter is to describe the definition and properties of cus-

tom software, where focal points of custom software development and test and individual solutions are discussed.

The third chapter presents a typical software test process model and a static infrastructure of managing and performing software test, where 11 important aspects or processes or workflows will be interpreted shortly, as has constituted a relatively complete picture for theoretical understanding and preparing for the late elaboration of the solution for testing custom software.

The fourth chapter analyses the main available IBM Assets and testing services supporting software tests. Two tasks will be made during the is-analysis: one for showing the problems of actual IBM Assets and testing services, which should be dealt with later; the other is to find specific important and valid points or best practices from them, which can then be used in the late solution. [**Note**: this chapter should be IBM confidential]

The fifth chapter will elaborate Virtual Testcenter (VT) for testing custom software; a survey would be performed and its results would be analyzed to define the requirements on VT, where earlier is-analysis and theoretical analysis results and the properties of custom software are also adopted. Two scenarios (small and large projects) for solutions of Virtual Testcenter will be outlined with the emphasis on process and service components of VT's requirements.

The sixth chapter will outline the realization of Virtual Testcenter with the help of appropriate IBM Rational Tools. The infrastructure is to be set up to provide the whole analysis, description and elaboration of realizing VT, also including two different scenarios for small and large projects, where automation of processes and hardware- and software infrastructure would be elaborated to realize the requirements on VT.

The seventh chapter will discuss the test solution – VT.

The eighth chapter will summarize this thesis and offers an outlook about possibilities for further development.

2 Theoretical analysis

This capital introduces theoretical basis, which will give a high-level understanding of software test. Because software test is a part of a software development process, it cannot be isolated from any process model; there are different types of software development process models, which can be categorized into linear/sequential models, agile models and incremental/iterative models, each type of process model has its own characteristics here, so the corresponding software test has also its own characteristics, which should be analyzed and described for different types of process models.

The brief description, the architecture, main processes, the usage field and advantages and disadvantages will be stated for some specific examples of software development models. In particular role and range of software test are discussed, when, how and how often the software test is performed would be different for different models. Many concepts of software test, software quality, classification of software tests and software testability are discussed. At same time the definition, selection reasons and properties of custom software are also described, as well as characteristics of custom software in comparison with standard software, individual solutions with custom and standard software, and focal points of custom software development and test.

2.1 Software development process models and software tests

Software development process model encompasses all the task or activities involved in the software development and describes a framework in order to ensure that the software development is performed as expected.

There are many different types of software development process models. But based on its characteristics models can be categorized into:

- Linear/sequential models: e.g. Waterfall Model and V-Model
- Agile models: e.g. Extreme Programming (XP)
- Incremental/iterative models: e.g. Spiral Model, Rational Unified Process (RUP).

In the following the above mentioned models are stated:

2.1.1 Linear/sequential models

In this type of models software development process is divided into different separate phases, each of which has a specified task to process the output of its preceding phase and deliveries the corresponding milestone as input to its succeeding phase. These phases should be executed in a strong linear sequence so that a phase must be completely finished at first before its next phase could begin. Software development using linear, sequential models can be seen as inflexible and non-iterative.

The sequential process, e.g. waterfall, is normally fine for small projects that have few risks and use a well-known technology and domain, but it cannot be stretched to fit projects that are long or involve a high degree of novelty or risk. [Kruchten03]

Two examples – Waterfall Model and V-Model of linear models, which are often used in practice, are outlined in the following:

Waterfall Model:

It is believed that the Waterfall Model has been the first process model that was introduced and widely followed in software engineering. In 1970s Royce proposed what is presently referred to as Waterfall Model as an initial concept, which consists of 7 phases (see Figure 2.1).

System requirements: all requirements, which the system needs and have to be developed, are collected by analyzing the needs of customers. A system requirement specification document is to be generated and used an input for the next phase.

Software requirements: the requirements of the software are gathered from system requirement specification. It describes the behaviour of the system e.g. with the help of a set of use cases that describe the interactions the customers will have with the software. Other non-functional requirements will also be included in this step.

Analysis: the software requirements are analyzed for the next phase.

Program design: results of analysis will be translated into a representation of the software and the main blocks and components of the system and software are outlined.

Coding: the design is translated into a programming language.

Testing: software tests, e.g. unit testing and integration testing are conducted to prove if the software functions as required.

Operations: the finished system is delivered to the customer and will be run on his environment.

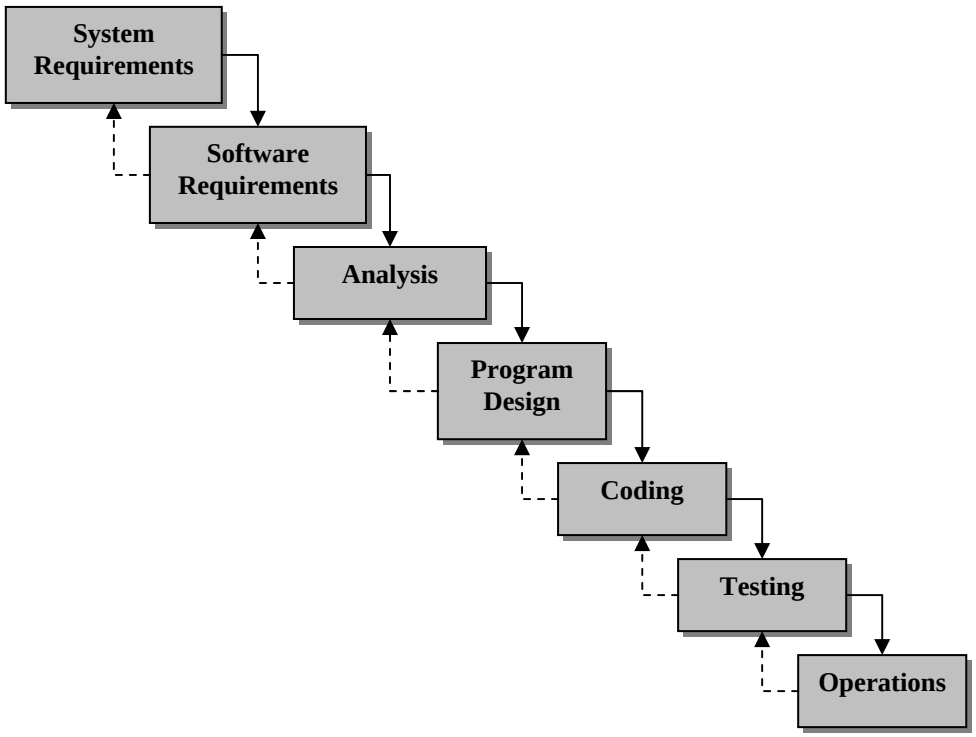


Figure 2.1 Waterfall Model [Royce70]

As can be seen that all possible requirements need to be collected and fixed in the early phases, and then other proceeding phases can start; if some requirements have not been gathered at beginning of the project, the subsequent phases will suffer from it. But in reality usually only a part of the requirements is known at the beginning and a good deal will be gathered during the complete development time [TSE].

A software development following the Waterfall Model is also document driven, that is to say, documents will be produced at the end of each phase e.g. requirement specification documents and design documents.

Software test in Waterfall Model

Software test as an explicit phase lies between coding phase and operations phase. The input for testing phase is the programs from coding phase, which need to be finished completely before testing is performed. In testing each unit or component will be tested for its functionality, as is also called unit testing. Both functional tests (black-box test) at the interfaces of the software modules and detailed tests (white-box test) of the inner structure of the software modules will be involved. Integration test can also be executed to test if all of the components cooperate as expected if the units are integrated into a complete system. Although software test is needed in Waterfall Model, but it has not covered all the other phases. Linear development process results in limited feedback on the results of the previous phases.

V-Model:

V-Model is another linear software development process model, which is developed by the German Federal Ministry of Defence in cooperation with the Federal Office for Defence Technology and Procurement in Koblenz since 1986.

The V-Model can be regarded as the extension of Waterfall Model. The process steps are bent upwards after the coding phase instead of moving down in a linear way, so the typical V shape is formed (see Figure 2.3). In V-Model for each phase of the development lifecycle there is an associated phase of testing, as is the major difference with Waterfall Model.

In the structure of V-Model there are 3 levels (see Figure 2.2) [Bucanac99]:

1. The Lifecycle Process Model (Procedure): in this level the procedures establish what activities are to be performed, which results these activities should produce and what contents these results must have. “What has to be done?” is answered.
2. The Allocation of Methods (Methods): the methods are determined to be used to perform the activities in the procedure level. “How is it done” is answered.
3. The Functional Tool Requirements (Tool Requirements): the functional characteristics, which the tools must have to be able to perform the activities, are determined. “What is used to do it” is answered in this level.

At each level the standards are structured according to submodels, there are 4 submodels, each of them stands for the specified areas of functionality, such as

1. Project Management (PM): plans, monitors, controls the project. It also passes information to the other submodels.
2. System Development (SD): develops the system or software.
3. Quality Assurance (QA): specifies the quality requirements and informs the other submodels of it. It specified for example test cases and criteria to assure that the products and processes comply with the standards.
4. Configuration Management (CM): administrates the generated products.

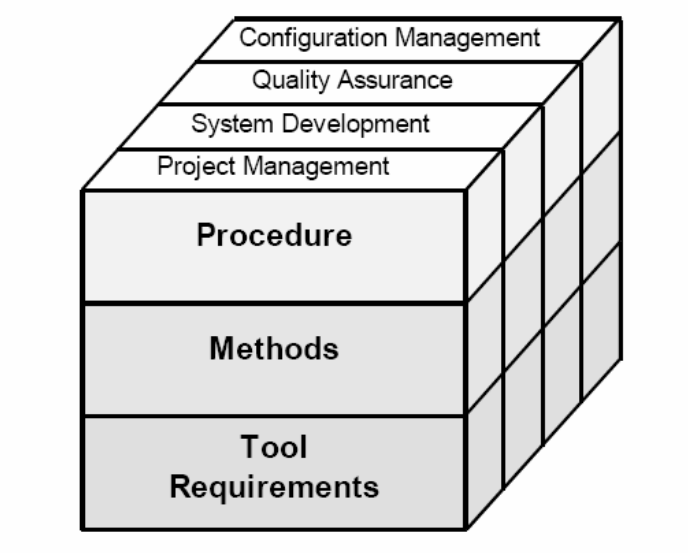


Figure 2.2 Architecture of the V-Model

The V-shape of V-Model can be found in System Development, which will be detailed in the following, the detailed description of the other three submodels can be found in [Balzert98;Bucanac99]

There are 9 main phases (activities) in Software Development submodel, which are performed in a linear sequence.

- System Requirements Analysis: system requirements are analysed from the customer's point of view, a description for the system and its technical and organizational environment are set up and a risk analysis is also performed.
- System Design: the system architecture is divided into hardware and software segments.
- Software/Hardware Analysis: technical requirements and operational information are updated and described with regard to software and hardware requirements.
- Preliminary Software Design: the software architecture and interface description are designed.
- Detailed Software Design: the design of the software architecture and interface description is further detailed, including the specification of each software component, module and database.
- Software Implementation: software components, modules and databases are realized in programming languages.

- Software Integration: software components, modules and databases are integrated and verified.
- System Integration: both software and hardware components are integrated and validated, so the integration of the system is made.
- Transition to Utilisation: the system is installed and put in operation in the intended environment.

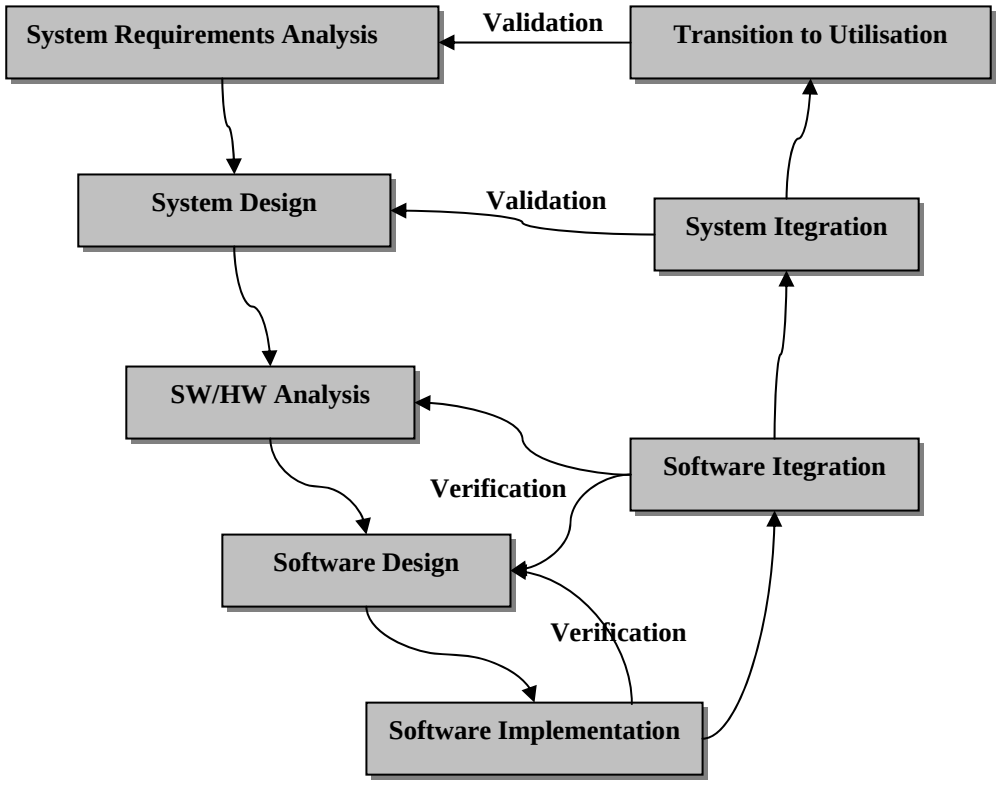


Figure 2.3 V- Model

Software test in V-Model

Software test plays a very explicitly important role in V-Model and associates a specified testing with each phase of the development cycle. Test design is prepared at the beginning of the project before coding, so that later tests can be written and performed efficiently to save much project effort.

There are 4 test levels in V-Model, which are widely used also in other models:

- Component test: in association with component design, also named Unit test, it is the first level of the dynamic testing. The source codes of every unit, which is the smallest part of the application, are analyzed to reduce the errors and it is also verified that these codes are efficient, behave as expected and adhere to the adapted coding standards. Component test is usually white-box and made using component test design prepared during the component design phase.

- Integration test: in association with software design, the separate modules, which have been verified in the unit test level, are tested together, where the

failures both in the interfaces of modules and in the integration between integrated components can be found.

- System test: in association with system specification, on this level the whole system of hardware and software components is tested against the system specification. All the integrated software modules and the software system itself integrated with any applicable hardware system are to be tested. All the functional and non-functional requirements are checked if they have been met.

- Acceptance test: in association with system requirements, the system is checked against system requirements, that is to say the system is to validated if the system delivers what the customer requests. So on this level the customer not the developer or tester will do acceptance test.

In addition to above mentioned 4 levels, in [CC] the fifth level – Release test for V-Model is also pointed and described.

At the same time in the submodel Quality Assurance (QA) the quality requirements, test cases and criteria are specified, as is very helpful to perform software test in the submodel System Development (SD).

In some books, the name of V-Model is also described to stand for Verification & Validation to show the importance of Verification & Validation in the V-Model.

2.1.2 Agile models

Agile Modeling (AM) is *a practice-based methodology for effective modelling and documentation of software-based systems. At a high level AM is a collection of best practices. At a more detailed level AM is a collection of values, principles, and practices for modelling software that can be applied on a software development project in an effective and light-weight manner*. [AgileModeling]

Agile models are suited for small up to medium projects with small number of developers, where requirements could change very often. An iteration of the agile model is the software, which is developed during one unit of time (typically 2-4 weeks). Each iteration passes through a full software development cycle: planning, requirements analysis, design, writing unit tests, then coding; when source codes have passed the earlier written tests, a work product will be delivered.

Extreme Programming (XP)

With the increased competitiveness of software products, the products need to be introduced into the market very quickly, so in the 1990s Extreme Programming was developed as a lightweight, agile development process to support rapid application development and has become by far the most popular of agile models. The purpose of the relatively new XP development methodology is to create quality programs in short time frames with low risk in vague or rapidly changing environment.

XP is based on a series of rules and practices (Table 2.1), which ensure the XP development process works properly.

Planning	Coding
• User stories are written. • Release planning creates the schedule. • Make frequent small releases. • The Project Velocity is measured. • The project is divided into iterations. • Iteration planning starts each iteration. • Move people around. • A stand-up meeting starts each day. • Fix XP when it breaks.	• The customer is always available. • Code must be written to agreed standards. • Code the unit test first. • All production code is pair programmed. • Only one pair integrates code at a time. • Integrate often. • Use collective code ownership. • Leave optimization till last. • No overtime.
Designing	**Testing**
• Simplicity. • Choose a system metaphor. • Use CRC cards for design sessions. • Create spike solutions to reduce risk. • No functionality is added early. • Refactor whenever and wherever possible.	• All code must have unit tests. • All code must pass all unit tests before it can be released. • When a bug is found tests are created. • Acceptance tests are run often and the score is published.

Table 2.1 Rules and practices of XP [XPHP08]

Figure 2.4 shows the general process of a XP project. The basis for the project is User Stories and Architectural Spike. User Stories will be constructed by the customer and describes functionality, which the system should provide to the customer. Spikes are small programs, which show the ability of realization of technical or design problems. Consequently it is determined in the Release Plan that which User Stories should be contained in the next release and the Stories will be divided into separate iterations.

An iteration encompasses three components: an iteration plan, the implementation and a runnable version of the implemented User Stories. This version will then be passed to Acceptance Tests, which are executed against the User Stories. In the end a smaller release will be put out, which was discussed with the customer. So it is tested if the requirements of the customer have been implemented correctly. The change requirements can be directly comprised in the next Release Plan.

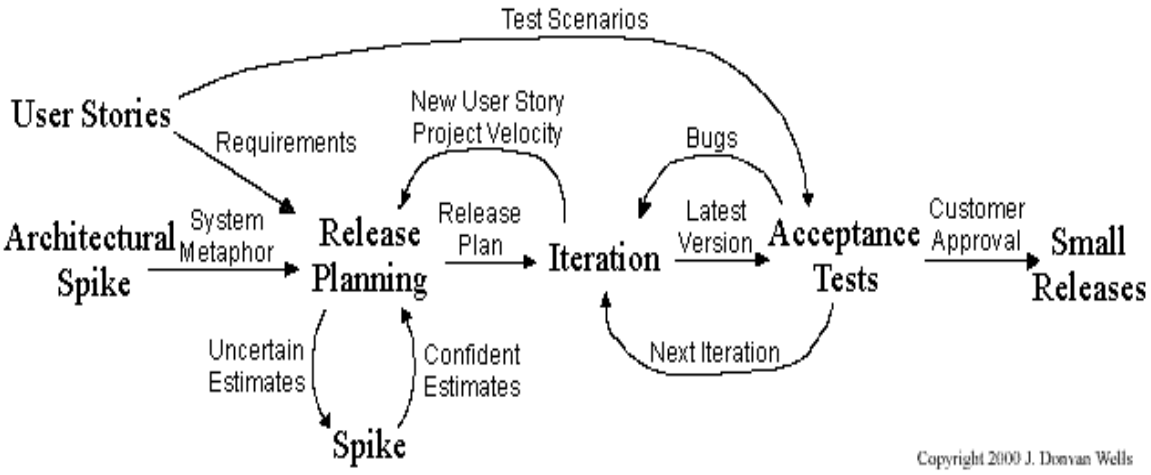


Figure 2.4 XP Project [XPHP08]

Software test in Extreme Programming

The XP Model is test driven, namely software test is so important in XP that the model requires that the unit and acceptance tests (see section 2.1.1) must be first designed and created before the programming can start, as is much different from other types of development models. XP relies strongly on unit and acceptance tests of modules; unit tests have to be performed for every incremental code change, no matter how small, to ensure that the code base still meets its specification. The testing aspect of Extreme Programming is also termed *Extreme Testing*. [MSBT04]

In Extreme Testing Unit tests are the primary testing method and have two rules (see Table 2.1): All code must have unit tests before coding begins, and all code must pass unit tests before being released into production. By writing unit tests before coding the project member can better understand the specification and requirements of the application from customer, as is helpful to do programming later. Once unit tests have been written and validated, the testing codes become as important as the software application itself and are to be backed up for the reuse later.

When a bug is found tests are created to guard against it coming back. A bug in production requires an acceptance test be written to guard against it. Acceptance tests are run often directly by the customer. [XPHP08]

There is also an implicit *code review* (static software test) in XP model, which happens in *Pair Programming*: two programmers share one computer, one programs and the other thinks together and checks the code so that the design is improved and the failure can be found more quickly.

In XP Model other test levels e.g. integration and system tests are not explicitly mentioned and emphasized, so from this point of view XP is particular suited for not very large projects, which will be rapidly developed to be brought into the market place.

As can also be seen that test driven XP Model has regarded the software testability very well.

2.1.3 Incremental/iterative models

In practice it is always impossible that all the requirements of the customer can be determined completely in the definition phase at the beginning of the project. So if the new or changed requirements are proposed at the late phases of linear models, the complete work process will be destructed. But the iterative models are so flexible that the changed or new requirements are allowed, the software products are developed iteratively, and the milestone of every iteration can be regarded as the increment, which must be an operational product.

Spiral Model

The Spiral Model was a meta-model defined in [Boehm88], which is risk driven in particular suited for large projects. The software product is divided into several cycles and every cycle produces a milestone, which passes four quadrants (see Figure 2.5) [Boehm88;Balzert98]:

- Determine objectives, alternatives, constraints: the objectives of the product, the alternatives to implement this product and the constraints, which need be paid attention to for different alternatives, will be identified.

- Evaluate alternatives, identify, resolve risks: these alternatives are evaluated under the consideration of objectives and constraints, if the evaluation shows that there are risks, a cost effective strategy is to be developed, in order to overcome risks e.g. through prototypes, simulations, benchmarking, reference checking and so on.

- Develop, verify next-level product: depending on the relative remaining risks, a process model such as evolutionary model, prototype model or waterfall model, will be determined to develop the product, in order to reduce the risk, as can also be got using the mixture of different models.

- Plan next phases: the next cycle including the necessary resources is planned. It contains a possible distribution of a product in components, which can be developed independently. Review of the above 3 quadrants including the plan for the next cycle will be also made by the involved persons or organizations. The commitment about the next cycle is set up, too.

Through these 4 quadrants each cycle produces a milestone such as software requirements, software product design, and separate software components until the operational real software product is finished in the last cycle. The objectives of each cycle derive from the results of its proceeding cycle, where the reduction of risks is very important and determines which process model should be used for the specified cycle.

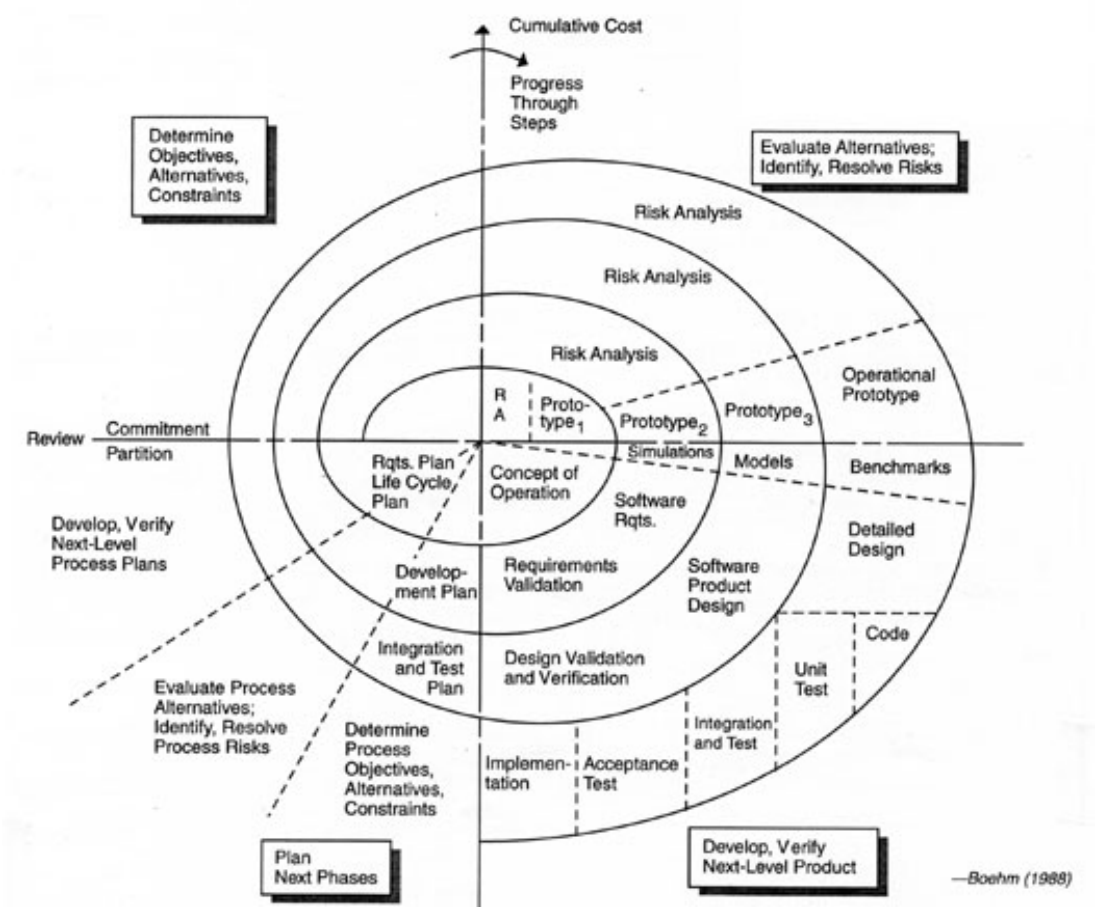


Figure 2.5 Spiral Model [Boehm88]

Software test in Spiral Model

Software test in Spiral Model is performed in each cycle for each milestone. This is an important feature of the Spiral Model that each cycle is completed by a review in the 4. Quadrant - Plan next phases, which involves the primary people or organizations concerned with the product. This review covers all products developed during the previous cycle, including the plans for the next cycle and the resources required to carry them out. The review's major objective is to ensure that all concerned parties are mutually committed to the approach for the next phase. The review and commitment step may range from an individual walk-through of the design of a single programmer's component to a major requirements review involving developer, customer, user, and maintenance organizations. [Boehm88]

In the 3.Quadrant - Develop, verify next-level product each level of software specification is validated and (/or) verified e.g. software requirements and software product design are validated and verified. The detailed software design such as codes and modules is tested through the test levels of V-Model.

So there is no separation between development and test, tests are performed iteratively for each milestone, so that any type of defect of the product can be found earlier and the risks are also reduced. Such a flexible Spiral Model is better suited for (very) large project.

Rational Unified Process (RUP)

The IBM Rational Unified Process is a software development process that covers the entire software development lifecycle, which ensures the production of quality systems in a repeatable and predictable way by assigning tasks and responsibilities, specifying the artefacts to be developed and offering criteria for monitoring progress and performance. It is an open process framework that software development organizations can configure and extend to suit their own needs. [Kruchten03]

The RUP is object-oriented and uses Unified Modeling Language (UML) as notation language. RUP is based on a set of six key principles (in short **ABCDEF**) for business-driven development: [KR05]

- **A**dapt the process
- **B**alance stakeholder priorities
- **C**ollaborate across teams
- **D**emonstrate value iteratively
- **E**levate the level of abstraction
- **F**ocus continuously on quality

The RUP brings these industry's best practices in the creation, deployment, and evolution of software-intensive systems, together in a form that is suitable for a wide range of projects and organizations (from small to large projects).

In RUP the software product will be developed through iterations, which have four phases, each of which is concluded by a major milestone. (see Figure 2.6)

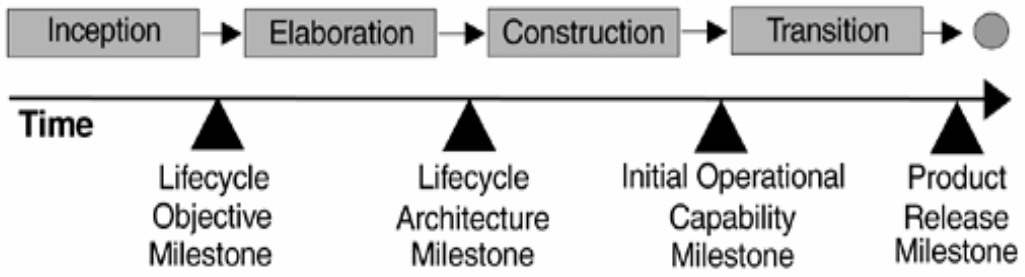


Figure 2.6 The four phases and milestones of RUP [Kruchten03]

The four phases with milestones in details:

- **Inception**: in this phase the end-product vision and business cases are specified and the scope of the project is defined, in addition the project plan, the fundamental Use Case Model, risks estimation etc. are created. The inception phase is concluded by the *lifecycle objective* (LCO) milestone.

- **Elaboration**: the focus of this phase is the architecture of the project, Use Cases are elaborated and a development plan is set up. In the end the *lifecycle architecture* (LCA) milestone is made.

- **Construction**: the purpose is the development, integration and testing of component and other features of the system being designed. The first external release

of the software is produced and ready for delivery to its user community. This phase is concluded by the *initial operational capability* (IOC) milestone.

- **Transition**: moving the product to its users, which includes manufacturing, delivering, training, supporting, and maintaining the product until users are satisfied. Beta-test and small changes could also take place. It is concluded by the *product release* (PR) milestone, which also concludes the cycle.

Each of the above 4 phases consists of one or several iterations, each iteration looks like a small project and involves some requirements planning and some designing, implementing and testing of applications, till producing a deliverable that is one step closer to the final solution (see Figure 2.7). In this way the progress can be demonstrated and risks are addressed early. All team members are involved in iterations of all 4 phases of the solution delivery lifecycle, so the entire team owns quality.

The emphasis on which activities of each iteration varies, e.g. in Figure 2.7 more analysis and design is required in the earlier construction than in the later.

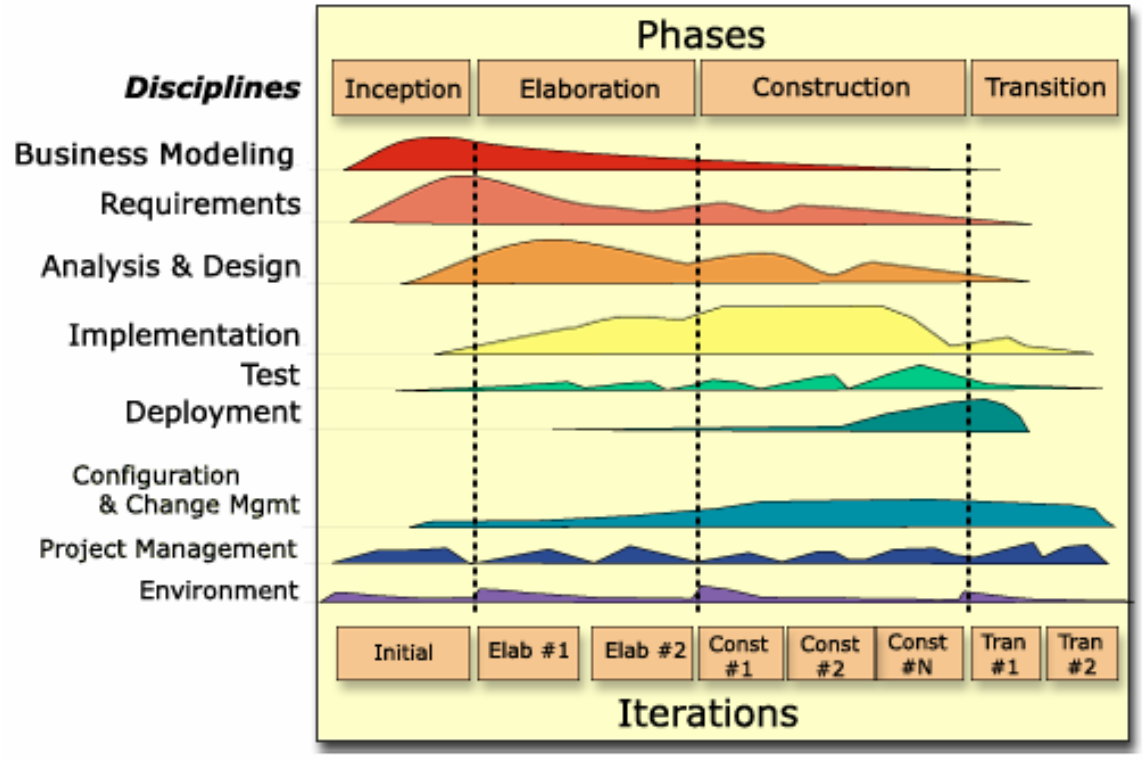


Figure 2.7 RUP [Kruchten03]

Within each iteration the tasks are categorized into nine Process Disciplines, which are divided into six technical disciplines and three supporting disciplines: [Kruchten03]
The technical disciplines are as follows:

- Business modeling discipline
- Requirements discipline
- Analysis and design discipline
- Implementation discipline
- Test discipline

- Deployment discipline

The supporting disciplines are as follows:
- Project management discipline
- Configuration and change management discipline
- Environment disciplin

In Figure 2.7 it can seen that not every discipline will be necessary for every iteration.

Software test in RUP

Software test in RUP takes place continuously and iteratively in all phases of the lifecycle, so that every milestone of the project is tested (all the test levels from V-Model are also performed here), its quality is ensured and an objective assessment of the (even being produced) project's status can be enabled. In Figure 2.7, as can be seen that the workload of the testing team is spread out throughout the lifecycle. In this way early feedback on product quality is used to measure the quality and identify and resolve the defects, as leads to improving the product quality as it is designed and built, so the software product is always of better overall quality than the one developed from a linear or sequential process.

For the RUP is a use-case-driven approach, in the object-oriented theory user cases that are defined for the system are the basis for the entire development process, so in the software test user cases become the basis for the identification of *test cases* and *test procedures*. Use cases are a means of expressing requirements on the functionality of the system. Each use case is performed to verify the system. [Kruchten03]

In addition to all test levels: unit test, integration test, system test and acceptance test, other more test types are to be performed in the RUP (see section 2.2.3)

Another feature of the RUP is that different activities of its software test can be supported by the IBM Rational Tools.

2.2 Software test

In the above analysis, as can be seen that software tests is the necessary and important part of the software development process and can regard every activity and milestone of the development of the project, so the definitions of software test and its related concepts are to be described in this section:

The software test is defined as „*The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item.*" [IEEE98] here the software item can be the complete system, subsystem, program, or module, which are made of source code, object code, job control code, control data, or a collection of these items. The features could be not only functionality but also performance, portability, etc.

Software test is also viewed as an important part of the software quality assurance (SQA) or software quality management (SQM), which is analytical, dynamic approach to ensure the software quality and its primary purpose is to find and then resolve the defects of the software. The other approach of SQA is the *constructive approach*, which provides the methods to develop a quality software product (for details in [Balzert98]).

2.2.1 Software quality

The software quality is the entirety of the characteristics and the values of these characteristics of a software product, which are suited to accomplish the stable or presupposed requirements [Balzert98], so in [ISO9126] the software quality is defined in a structured set of characteristics and sub-characteristics:

- **Functionality**
  - Suitability
  - Accuracy
  - Interoperability
  - Security
  - Compliance
- **Reliability**
  - Maturity
  - Fault Tolerance
  - Robustness
  - Recoverability
- **Usability**
  - Understandability
  - Learnability
  - Operability
- **Efficiency**
  - Time Behaviour
  - Resource Behaviour
- **Maintainability**
  - Stability
  - Analyzability
  - Changeability
  - Testability
- **Portability**
  - Installability
  - Replaceability
  - Adaptability

This definition only regards the product quality not the process quality. These quality characteristics can be classified into functional and non-functional requirements. The software tests should regard all the aspects, which are above introduced, of the software product. Each quality sub-characteristic is further divided into attributes. An attribute is an entity which can be verified or measured in the software product. Attributes are not defined in the standard, as they vary between different software products, so software

tests measure the software quality by the amount of the found defects. These defects will be resolved and then the software quality will be improved accordingly. (For the contents of each characteristic, please see materials referenced here or glossary)

2.2.2 Principles of good software test

There are following principles for good test [EM07;Imbus08]:

- Business risk can be reduced by finding defects: the defects can be resolved before software product runs in the operation, as reduces business risk.

- Positive and negative testing contribute to risk reduction: positive testing verifies that the software works as expected. Negative testing verifies that the customer cannot break the software under normal situations.

- Static and execution testing contribute to risk reduction: static testing aims to find defects of documents produced in the development, and will reduce the number of execution defects, which are found in the execution testing.

- Automated test tools can contribute to risk reduction: automation test tools improve test quality significantly, in particular for performance test.

- Make the highest risks of the first testing priority: at least the sufficient testing must be made for the top business tops.

- Make the most frequent business activities (the 80/20 rule) the second testing priority: concentrate the testing on the 20% of the business system functions, transactions, or workflow, which really drives the business, because the 20% can satisfy 80% of daily business tasks.

- Statistical analyses of defect arrival patterns and other defect characteristics are a very effective way to forecast testing completion: using the statistical model can predict when the test can stop.

- Test the system the way customers will use it: the customer is “God”, the software system should be tested from the customer’s perspective.

- Assume the defects are the result of process and not personality: the tester should find a way to focus on the defect without seeking to place blame.

- Testing for defects is an investment as well as a cost: testing tools can reduce the overall cost of testing when compared with the same testing done manually. The reuse of test scripts and other patters is also helpful.

- Complete test is impossible: a complete test, where all possible inputs and their combinations under all different pre-conditions are performed, is impossible. So test effort is controlled by risks and priorities.

- Begin the test as early as possible: test activities should start in the software development process as early as possible, so that defects could be found early and defect cost is reduced.

- Test is dependent on the environment: the test should be customized according to the environment of running the system. The definition of test strategies and criteria for the concrete application is to be made.

- Test shows the presence of defects: test can show the influence of defects of software, but cannot prove that there are no defects in the test object. Through enough tests the likelihood that not found defects exist in the software will be reduced.

2.2.3 Classification of software tests

There are different ways to classify the software tests:

2.2.3.1 Test levels

According to different abstraction levels or different documents and artefacts, there are 4 test levels: Unit test, Integration test, System test and Acceptance test. (See section 2.1.1 software test in V-Model)

2.2.3.2 Functional and non-functional tests

Based on a specific test objective of the to be tested component or system, there are following two different test types:

Functional test: the software behaviour is verified and validated against the functionality, which is documented in the software requirements and specification. The functionality can come from Use Cases that describe a system's behaviour as it responds to a request that originates from outside of that system.

Typical functional requirements to be tested are [LFH08]
- *Business Rules*
- *Transaction corrections, adjustments, cancellations*
- *Administrative functions*
- *Authentication*
- *Authorization –functions user is delegated to perform*
- *Audit Tracking*
- *External Interfaces*
- *Certification Requirements*
- *Reporting Requirements*
- *Historical Data*
- *Legal or Regulatory Requirements*

Functional test is concerned with the functional requirements and covers how well the system executes its functions. These include user commands, data manipulation, searches and business processes, user screens, and integrations.

Non-functional test: non-functional requirements of the test objects will be tested which stand for the non-functional characteristics of the software quality and specify criteria that judge the operation of a system, rather than specific behaviours.

Typical non-functional requirements are [LFH08]:
- *Performance - Response Time, Throughput, Utilization, Static Volumetric*
- *Scalability*
- *Capacity*

- *Availability*
- *Reliability*
- *Recoverability*
- *Maintainability*
- *Serviceability*
- *Security*
- *Regulatory*
- *Manageability*
- *Environmental*
- *Data Integrity*
- *Usability*
- *Interoperability*

Non-functional test enables the measurement and comparison of the testing of non-functional attributes of software systems:

- *Performance test*
- *Security test*
- *Usability test*
- *Dependability test*
    - *Reliability*
    - *Maintainability*
    - *Availability*
    - *Recoverability*
- *Miscellaneous test*
    - *Interoperability*
    - *Compatibility*
    - *Portability Configuration*
    - *Installability*

2.2.3.3 Static and dynamic tests

Static test:
All the documentations produced in the development process can be tested through manual methods or static analysis, where the to be tested software is not executed. Static test is performed by the corresponding roles or tools. It aims to find the defects as early as possible, before they go to the next development phases.

Types of manual methods [Balzert98]**:**

- *Inspection*
- *Review*
- *Walkthrough*
- *Round Robin review*
- *Peer review*

Above mentioned manual methods test the characteristics of artefacts, which are not able to be tested using automation tools. There are different roles, who attend one type of review and include e.g. manager, moderator, author, supervisor and so on. 60-70% of the defects or failures or defects in a document can be found manually here.

Static analysis:
Static analysis is to test the documents with a formal structure. It is always sensible to perform static analysis with automation tools; in practice the program source codes are

always the first and unique formal documents during the development process, which need static analysis.

At first all the compilers will execute static analysis of program source codes as the analyser, in order to find the violation of the syntax.
In addition, analysers are able to perform the following static analysis:

- *Test of conventions and standards*
- *Data flow analysis*
- *Control flow analysis*
- *Count of metrics (e.g. McCabe´s Cyclomatic Number)*

For formal documents static analysis takes place before manual methods, as static analysis is performed with automation tools so that less effort is needed than manual methods.

Dynamic test:
In contrast to static test, the to be tested component or system need to be executed in dynamic test, in order to test the dynamic behaviour of codes. So the software must be already compiled and run.

During dynamic test input values are given for software, and its according output will be checked if it is as expected.

There are two types of dynamic tests:

- *Black-box tes*t: the test object is regarded as a black box; the inner structure and the program code are not known. This test is based on the specification of the test object. It consists of the following:
    - *Equivalence partitioning*
    - *Boundary value analysis*
    - *random test*
    - *test of state transition tables*
    - *smoke test*
- *White-box test:* it is code-based, and the inner structure of the program should be regarded, the goal is that all code parts must be run at least one time, normally suited for lower test levels particular for Unit test. It is based on following technologies:
    - *Statement Coverage*
    - *Branch Coverage*
    - *Condition Coverage*

(For the contents of each concept in section 2.2.3, please see materials referenced here or glossary)

2.3 Software Testability

One important aspect of the software is its testability, which can determine the necessary effort of verification & validation of the software [Jungmayr04]. In this section different definitions of testability are introduced and in the end the important criteria of testability are discussed.

Definition:

There are many different definitions of software testability:

IEEE Standard Computer Dictionary defines testability as "(*1) The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met, and (2) the degree to which a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met.*" [IEEE90]. The focus of this definition lies on the support of the establishment of test criteria and on the execution of tests. A good testability here mentioned is that it is easy to define test criteria of to be tested software and to implement and execute these tests.

The ISO defines testability as "*a set of attributes that bear on the effort needed for validating the modified software* "[ISO9126]. In this definition only the test effort is related.

In [Jungmayr04] testability is defined as "t*he degree to which a software artifact facilitates testing in a given test context"* an important aspect is about the connection of a given test context. That can be so understood, e.g. if only the function tests are needed and the software provides also an easy possibility to execute them, so other characteristics (e.g. performance) have no influence (at least not directly) on the testability, as they should not be tested.

In [Kahlbrandt98] also defined: "*A software system is testable if 1) its components can be tested separately, 2) test cases can be identified in a systematic manner and repeated, and 3) the test results can be observed."*

Criteria:
The lower the testability is, the more software test effort will be needed. In extreme cases of bad testability, it is impossible at all to test parts of the requirements. So in order to get a high testability, it should be observed during the complete development process. In particular in the requirements analysis functional requirements as well as non-functional requirements should be formulated quantitatively, so that test cases can be inferred directly from the requirements [Jungmayr05]. Then in the following design phase the realization of testability requirements should be observed, the three most important steps are identified in this phase [Jungmayr04]:

- To transform the testability requirements defined in previous development activities into testable classes and testability features.
- To define the responsibilities of the classes in a way that facilitates testing.
- To define the interactions of the classes in a way that facilitates testing.

So a testable design needs to be implemented accordingly, in order to keep the testability.
In the implementation phase the following factors should be observed [Jungmayr05]:

- Complexity: interfaces with many parameters, deep class hierarchy and long cascades of method calls should be avoided.
- Understandability: recursions, complex algorithms and implicit control logic are to be avoided.
- Controllability: complex loop constructs and unreachable output values and paths should be omitted as possible.

- Observeablity: explicit object states and open control paths help to observe the test results and test coverage.
- Repeatability: all the dependences should be explicitly documented.
- Dependence: particularly cyclic dependences should be omitted as possible.
- Automateability: standard program constructs (in particular for GUI) should be used as possible, in order to make it easy to use test automation tools.

In [Testbarkeit08] the different approaches to develop testable software are introduced such as design for testability, test driven development, requirements driven development of testability.

2.4 Custom software

2.4.1 Definition

Custom software (also named bespoke software) is a type of software, which is developed in accordance with the requirements of an individual enterprise or a specialized type of task. It is the customized software in order to solve an individual problem of the customer.

Custom software is opposite of standard software, which is already available and is targeted to the mass market. Custom software can be classified into custom system-software or custom application-software [Vaher04]. In most cases custom application software is required e.g. the central systems of the banks or insurances are normally based on custom software, as well as other many web sites and web-based applications.

2.4.2 The reasons to select custom software

- No suitable standard software: normally the standard software costs less than custom software, but no suitable standard software can be found for the so specialized requirements of customers e.g. 1) the expected software would work with very special software or hardware parts, or 2) very special or unusual Use Cases are to be implemented, so customized software will be developed.

- The customer wants to have a complete control of the development in the future and guarantee to control the source codes. In contrast standard software usually will not make public its source codes to customers.

- The customer wants to be present individually in the market and be distinguished from other competitors. So through custom software, the enterprise can provide much better services or products to attract more customers.

- The customer desires to develop a better solution than what the existing products can provide.

2.4.3 Characteristics of custom software in comparison with standard software

As custom software is produced according to special requirements, it should support the work flow optimally [Aldebaran08]. The requirements of an enterprise can be better satisfied by custom software. If new requirements are requested, custom software can be extended relatively easily, for the source codes are available and can be changed or improved to implement new functionalities, not like users of standard software, who

must normally wait till the owner enterprise of standard software issues the next release with expected functionalities.

Figure 2.8 shows the characteristics of custom software in comparison with standard software; the advantages of custom software are normally the disadvantages of standard software and conversely.

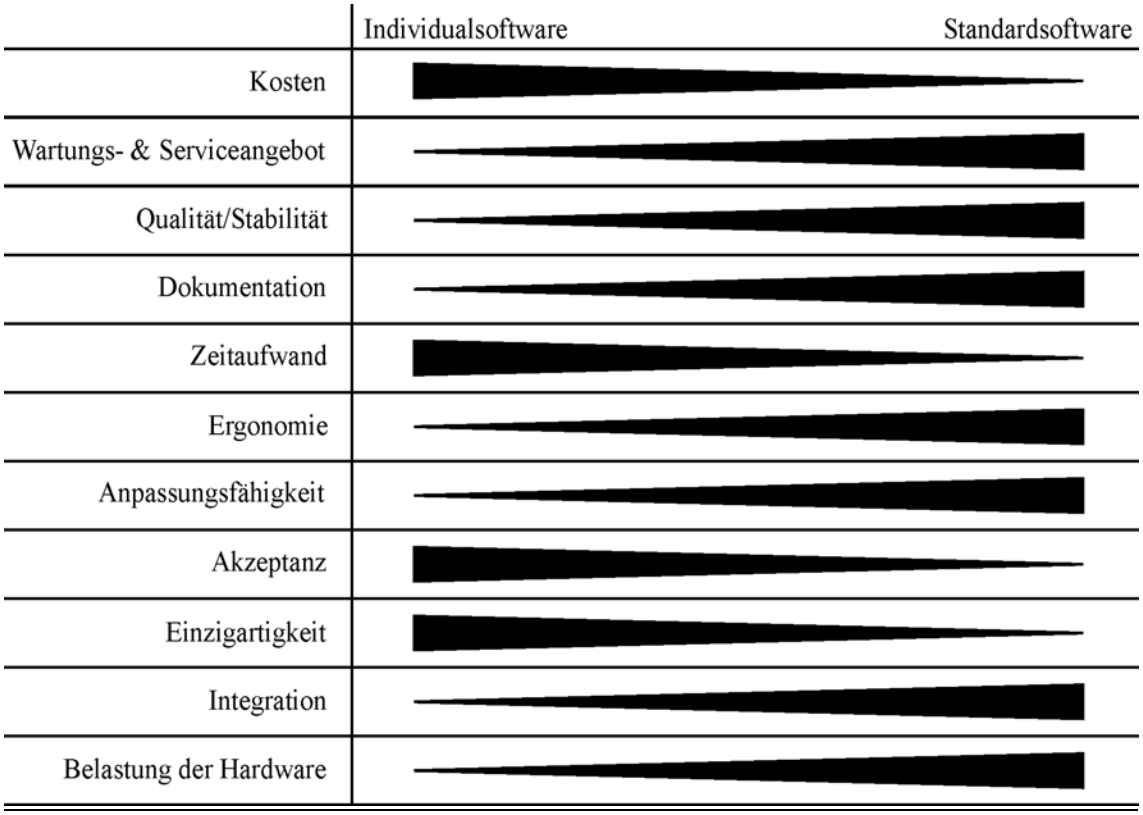


Figure 2.8 Comparisons between custom and standard software [Vaher04]

2.4.4 Individual solutions with custom and standard software

Custom software is developed only for individual solutions, which are suited for an individual enterprise or a specialized type of task. The most users of individual solutions are large enterprises or similar organizations, because seldom exactly suitable standard software products on the market are available for variety and changeability of directions, responsibilities and political interest of these large enterprises or organizations [Wiki08b].

Individual solutions can be realized not only using custom software but also through the adjustment or customizing of standard software, there are three possibilities of adjustment for standard software: parameterization (selection of program functions with different parameters), configuration (selection of expected program modules) and individual programming (individual adjustment or complement of software) [Vaher04].

In practice individual solutions can lie in the mixture of custom software and standard software, where it would be optimal to make use of advantages of the both; for the well understood and relatively general problems, standard software is suited, as well as for

competence-neutral fields, but the core processes, which ensure the core competences and the actual gain of the enterprise, should be realized by custom software.

2.4.5 Focal points of custom software development and test

Based on the characteristics of custom software, the focal points needed specially for the custom software development and test are pointed as follows:

- Custom software is for the very individual solution of the customer, so no experiences about these functionalities exist. Therefore the customer must be active in the whole development process to discuss with other roles, answer the questions of other roles and make the project plan, in order to minimize the risks.

- For this very new problem, in the reality the formulated requirements are often changed at each phases of projects, till the really correct requirements, best suitable solutions and new creative decisions are found. So effective change and configuration management are necessary.

- The critical business processes responsible for the core competences of the enterprise must be treated carefully. How are they creatively and quality developed and tested? Which modern suitable technologies and models are utilized?

- Custom software, which aims to provide a unique solution to meet the specific requirements and preferences of the customer as quickly as possible, implies both high cost and high risk. How are these potential problems, risks and costs effectively controlled? (It is worth adopting custom software, as the ROI (Return on Investment) may be very high.)

No matter what software development model (linear models, agile models or iterative models) has been adopted, the above mentioned problems must be treated and dealt with carefully, effectively and efficiently during each phase of the development process, in order to provide best satisfying custom software to the customer.

3 Infrastructure and models of testing software

This chapter gives a theoretical overview about infrastructure and models of testing software, where a typical test process model and a relatively complete picture about key processes and aspects of software test are shortly outlined. The objective is to give a theoretical explanation how to perform software test, so that it can be adopted or referenced later in the elaboration of the test solution named Virtual Testcenter.

3.1 Software test process model

In order to perform tests structured, software test process model is very useful. In Figure 3.1 a typical software test process model is shown, where these different phases: test requirements analysis, test planning, test development, test execution, test reporting, defect analysis, retest and regression test are discussed shortly sequentially, in practice they must not be performed sequentially, however they can be overlapped or executed partly parallel. This typical test process cycle can take place on all different test levels.

Test Requirements analysis: Software test begins from the requirements phase of software development process; what aspects of the software should be testable and with what parameter the tests should be run, are determined in the early phases of the development process. In particular for custom software, test requirements are discussed and determined directly with customers.

Test planning: the software tests should work as planed. The necessary resources need be planned for the test process. When and who is needed to execute the tasks, how much time is estimated and what aid and equipment are necessary? These questions should be answered in the planning and decided in the *test plan*.

In addition the *test strategy* is also made, so that the priorities of tested objects are determined based on risk estimation. Test methods (e.g. black-box test or white-box test, automation or manually), the sequence of these methods used on different test objects and coverage degree of each method are described in the test strategy, too.

The *test bed* realizes the parts of the test infrastructure, which are necessary for executing tests. Normally test bed consists of all the programs test tools will execute.

Test development: from the test strategy of test planning phase, test method for a specific test object has been determined, now *test specification* is developed to how test procedures work, which defines two types of *test cases*: logic and concrete test cases. Logic test cases depend on the used test method and consist of: the pre-condition, input, a series of steps (also known as actions) to follow, expected result or system behaviour. As the description of logic test cases is too abstract, they will be concreted so that concrete test cases are realized easily. *Test scripts* make test cases runnable and can be manual, automated, or a combination of both.

Test execution: if test environment have been ready, testers execute the tests (test scripts) based on the test plan and the priorities determined in the test strategy and test plan.

Test reporting: once test execution is finished, a report is created about all important information of this test execution: test effort, test cases description, related configura-

tions, test results and so on. Test report could also show if test results have been success or failed. Success means that the test object has passed the tests and been developed correctly, otherwise defects (reasons) for the failed tests should be found and delivered to the development team.

Defect analysis: the defect is every type of deviation between is- and expected results that can come from the software itself, test data, test environment or test requirements. So what defects should be treated, fixed, rejected or deferred to be dealt with at a later time point, as will be decided in this phase.

Retest: if the defects have been resolved by the responsibilities, the test will be rerun in order to check if defects were resolved actually.

Regression test: a previously tested program is tested if any change has been made to it, in order to ensure that new defects have not been introduced in unchanged areas of the software, as a result of the changes made. In iterative development process models regression test is performed for each iteration. A set of tests are included in regression test.

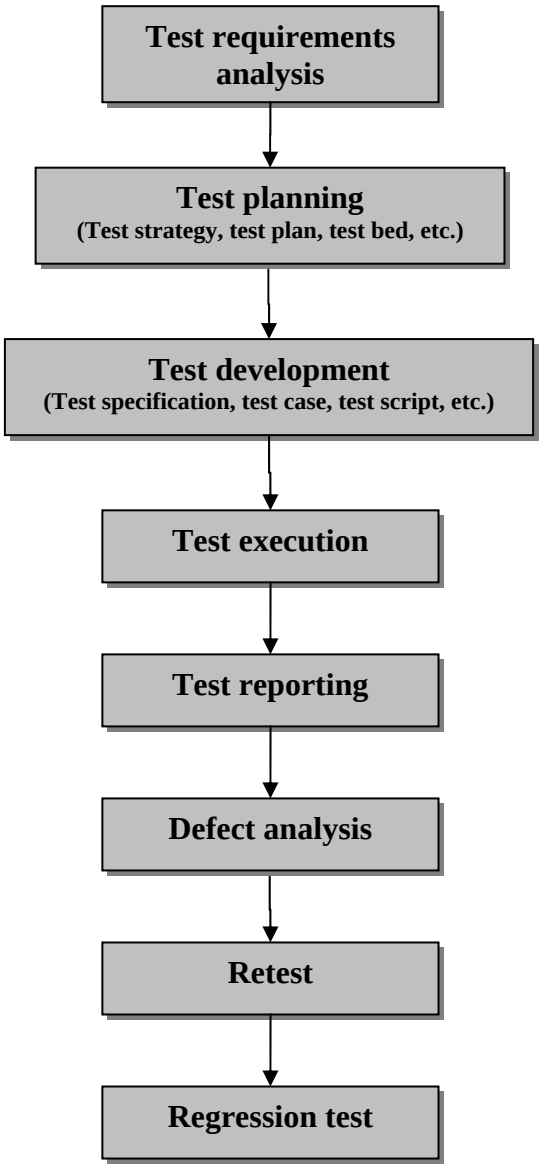


Figure 3.1 A typical test process model

3.2 Infrastructure of software test

The infrastructure of software test is to ensure that software tests are performed effectively so that the defects of software can be found as early as possible and resolved as planed. At the same time test team is managed and the test resources and cost are controlled. Software quality is improved through tests. The infrastructure should regard the aspects of software tests and encompass activities and processes of managing software tests.

The following key processes and aspects of managing software tests are discussed: test management documentation, test organisation, test process monitoring and controlling, test process assessment and improvement, defect management, configuration management, risk management, management of test standards and norms, test effort management and test tools management.

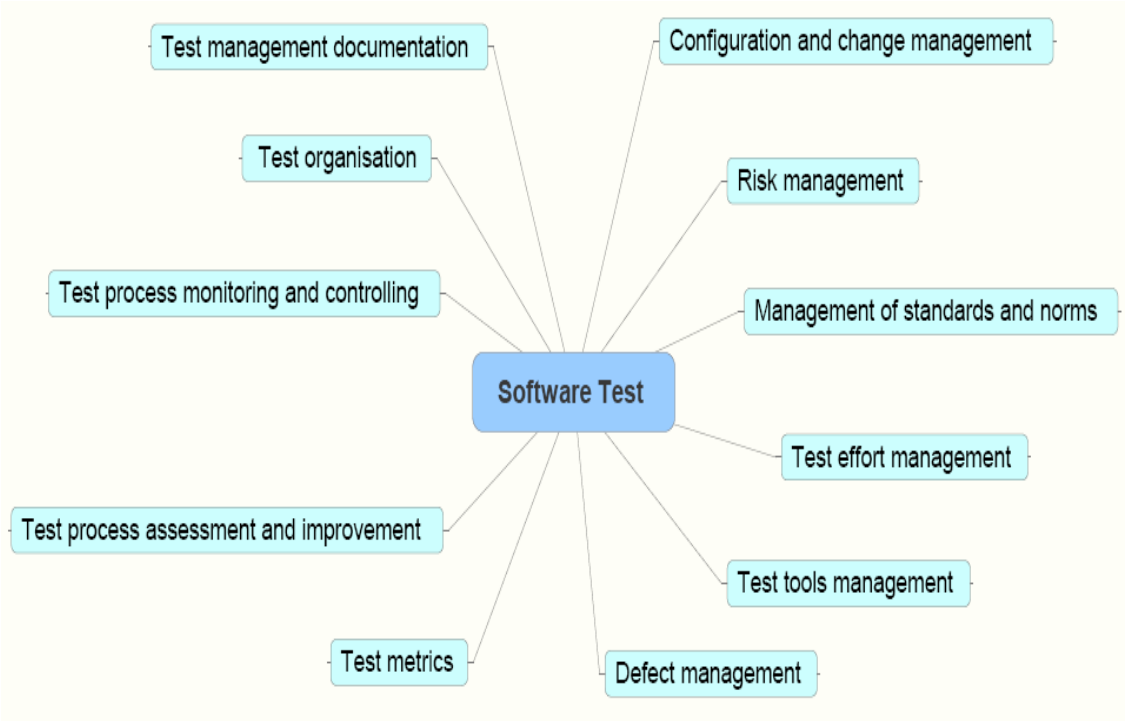


Figure 3.2 Key processes and aspects of software test

3.2.1 Test management documentation

Each enterprise has always specific requirements on the quality of its software products to satisfy the requirements of customers, which are often described in the *quality policy*. Quality policy is at the high level and the basis for the software quality management of the enterprise.

So based on its own quality policy of each enterprise, the documentations especially concerning to software tests are to be drawn up, which are on the sequence of abstract levels so listed: *test policy, test handbook, test concept and test level planning.* [Imbus08]

Test policy prescribes the enterprise policy concerning to software tests: What is the definition of the test in the enterprise? Which quality level should be got for software tests? How is the test process performed and improved?

Test handbook is based on test policy and details possible test levels for the specific software products and all the activities of each test level. Test policy and handbook are strategic guidelines for an enterprise and will always be implemented in test concept and level planning in specific test projects.Test concept describes the concrete implementation of test handbook for a specific test project. The concrete test levels and all the activities of each level of the test project are detailed. Test level planning implements test concept for a specific test level.

3.2.2 Test organisation

Activities of software tests can be carried out by developers, test team or both of them [Balzert98], normally depending on specific test levels, e.g. Unit tests are executed by developers, System tests by testers and Acceptance test by customers.

The roles for software tests should be appointed: *test manager* and *tester*.
Test manager is responsible for the execution of the test project, who plans, monitors and controls the whole test project and reports results and information to the management of the enterprise. Test concept must be made and customized to a specific test project also by test manager.Tester implements and runs tests for all test levels. Testers evaluate test results and report defects to related persons. Test tools are often also installed, prepared and grasped by testers so that test automation is realized.

Qualification of test team: all members of test team should have been educated before in the colleges or schools or be trained on the job, so that they possess the needed ability and qualification to ensure test quality.

Management of test team: test members will be motivated to be a successful test team, so that higher productivity in the organisation is got for all.

3.2.3 Test process monitoring and controlling

In order to ensure the successful and effective execution of test processes (Figure 3.1), test manager must monitor and control the complete test process actively: all the necessary approaches are used to make different test phases run as planed and test goals are got finally. Detailed information about executing test processes is recorded in the report. Test processes for different test levels or different versions of software products should also be performed parallel without conflict.

3.2.4 Test process assessment and improvement

The quality of software products is improved with the help of software tests, at the same time the quality of test process is also necessary to be assessed and improved. Which model can be used to improve test process without deriving much change on the original process? This question should be settled in this process.
Many available models such as SEI Capability Maturity Model® Integration (CMMI), SPICE, and Testing Maturity Model (TMM) can be used to assess and improve the software test process. [CMMI06; Balzert98] (See Appendix)

3.2.5 Defect management

One of the most valuable points of software tests is to find, report and resolve defects of software. Defect is every type of deviation between is- and expected results and can exist on all the artefacts including analysis, requirements, program source codes, modules, systems, etc. This process would describe how defects are categorized, prioritized and tracked. Normally a defect state model will also be established a specific test project, so that defects are settled correctly from “submitted” to “resolved” by the responsibilities.

3.2.6 Change and configuration management

Without reliable configuration management a good test process cannot be realized. Configuration and change management is to identify all the elements of test process through

the version control and to track changes of test objects or other artefacts, so that each configuration can be reproduced and the complete test process is able to be retraced.

Any change of documentations, test cases or test objects results in a new configuration. So every element of the development process need be identified uniquely. In particular for testers, it helps them to find unique test objects, test documentations, test cases, test specifications, test reports, defects and so on.

3.2.7 Risk management

Risk is a problem that could happen unexpected later in a test project or software product. The risk is prioritized according to its likelihood of occurrence and its importance priority. Risks in a test project are related to qualification of test team, quality of test documentations etc. These risks could be reduced e.g. by training the inexperienced testers.

Risks in the software product are often about software quality, e.g. which functionalities are not running well, so product risks should be mentioned early in the requirements in order to prepare for *risk oriented tests*, which consist of test methods and test environment so as to find the defects as early as possible, which resulted from risks. Risk management encompasses all the approaches to analyse and minimize the risks of software tests.

3.2.8 Management of standards and norms

One of test manager's tasks is to select standards and norms for the to be tested software product (product norms) or the test project (process norms). Possible sources of these standards and norms are [SL02]:

- **Enterprise standards**: intern guidelines and instructions of the enterprise or customers, e.g. quality management handbook, programming guidelines or other concrete processes.
- **Best practices**: not standardized, but technically already used and tested more effective methods or processes in practice, which represent the status of the technology of an application field.
- **Quality management standards**: cross-sectoral standards, which specify the minimal requirements on processes without formulating concrete requirements concerning to the implementation, e.g. ISO 9000, CMMI, SPICE, TMM (mentioned in the section 3.2.4)
- **Branch standards**: sectoral standards (e.g. DIN EN 60601-1-4 for medicine products), which define in which minimal extent tests must be executed or verified for a specific product catalog or an application field.
- **Software test standards**: process standards, which are independent of products and define how to execute software tests professionally, e.g. [ISO9126] for software quality, [IEEE829] for software test documentation, [IEEE1028] for software reviews.

3.2.9 Test effort management

Test effort consists of the whole cost for a test project. It depends on test process quality (e.g. maturity of the process for CMMI), testability of test objects, test strategy, quality goals, skills of test team, degree of test automation etc. With the short time to market and improved quality to market at present, test projects today are required to be accom-

plished quickly and effectively. Some approaches need be taken to control test effort and improve test quality as well as.

3.2.10 Test metrics

Test metrics are to analyse the software quality as well as the process quality of software development and tests, using quantitative and periodic assessment of the measured objects. There are different types of metrics such as *product metrics, process metrics and project metrics* [OST08]. From the view of different measured objects, test metrics can also be categorised into defect based metrics, test case based metrics, test effort based metrics and so on [Konda05]. From specific metrics such as *coverage analysis*, exit criteria to test phases can be provided and exit time points of the complete test would also be estimated. The history data of defects makes it possible to estimate the likelihood of remaining defects and reliability of the software. Test metrics help to monitor and control the test process and manage risks of projects and test.

3.2.11 Test tools management

Test tools should be able to regard each phase of test process (Figure 3.1). Test activities are supported by test tools to realize *test automation*. At the same time many above mentioned processes or aspects (e.g. configuration and change management, defect management) of software test could be also supported by appropriate test tools. So in practice it makes sense to realize these processes with the help of test tools so that the test project can be conducted effectively. There are lots of types of test tools according to test levels and test methods. How to adopt test tools well with test process and the training of testers are also necessary to be considered and planned so that test projects are able to profit from test tools really.

4 Is-analysis of IBM Assets and Testing Services

[Note: this chapter should be IBM confidential]

The available IBM Assets, which are implemented through IBM Rational Tools and can represent the actual situation of IBM testing services, will be analyzed here in this chapter to give an overview of the actual situation of IBM testing services. These IBM Assets will be specially analyzed regarding testing software. The following questions will be thought of during the analysis: What does each solution accomplish? Are they generally valid or which part of them is generally valid? What is the difference between them? Which parts of software development process do they cover? The supported processes or workflows in each Asset will be introduced.

Is-analysis has two tasks: one for showing the problems of actual IBM Assets and testing services, which should be dealt with later; the other is to find specific important and valid points or best practices from them, which can then be used in the late solution.

So at the end of is-analysis an assessment of IBM Assets and Testing Services will be discussed.

4.1 CQTM Standard Schema

Rational ClearQuest Test Manager(CQTM) is a feature integrated with IBM ClearQuest (in the version 7.0 Enterprise schema or by applying the CQTM package to an existing schema) that manages the components of a testing environment. The following components are managed:

- Test plans
- Test cases
- Test requirements
- Test configurations
- Test scripts
- Test results

Test plans identify test cases, which represent the ways in which users are likely to use the product. Test scripts are written to ensure that the requirements for the test cases are met and can be executed. Test configurations represent a runtime where a test is run. The results returned when the test scripts run are evaluated to determine project status, in particular, the progress toward completing the work represented by the next milestone in the project. The hierarchical relationship between these files, documents, and data is represented by records in a Rational ClearQuest database. The records in the database are organized in a test planning hierarchy [IRCQ].

Figure 4.1 shows the object model of CQTM, where objects in yellow are all managed and created directly in asset registry of CQTM, asset registry is the starting point for creating a test planning hierarchy and defines the scope of testing at the highest level e.g. to represent a product or a release, in addition to above mentioned components of a testing environment, these objects in the following are also important for test management:

Iteration: represents a software development milestone for which testing is required, it can be same as in RUP. It can be associated with many other objects.

Configuration: collects attribute values that define testing environment, which may define a hardware or software definition for executing a test script e.g. operating system – unix, processor – 3 Ghz, memory – 4 GB, and network – 100 Mbs. A test case is configured with a configuration record to be configured test case, so that it is executable.

Test suite: contains a set of configured test cases, which must have the same configuration (testing environment) and can be executed sequentially. Here regression test can be implemented through test suite.

Test log: displays a summary of the results for an executed configured test case record.

Suite log: contains a set of test logs for each executed configured test case record of a test suite to display a summary of the results of executed test suite.

File location: represents the location for each external file e.g. test motivator files associated with a test plan or test case record, a single test script file associated with a test case or configured test case record or a test log file referenced by a test log record.

In addition there are Computer and Computer group record types to enable test execution on remote computers, where Computer record identifies the name and network address of a computer where tests are planned to run and computer group record identified multiple computer records.

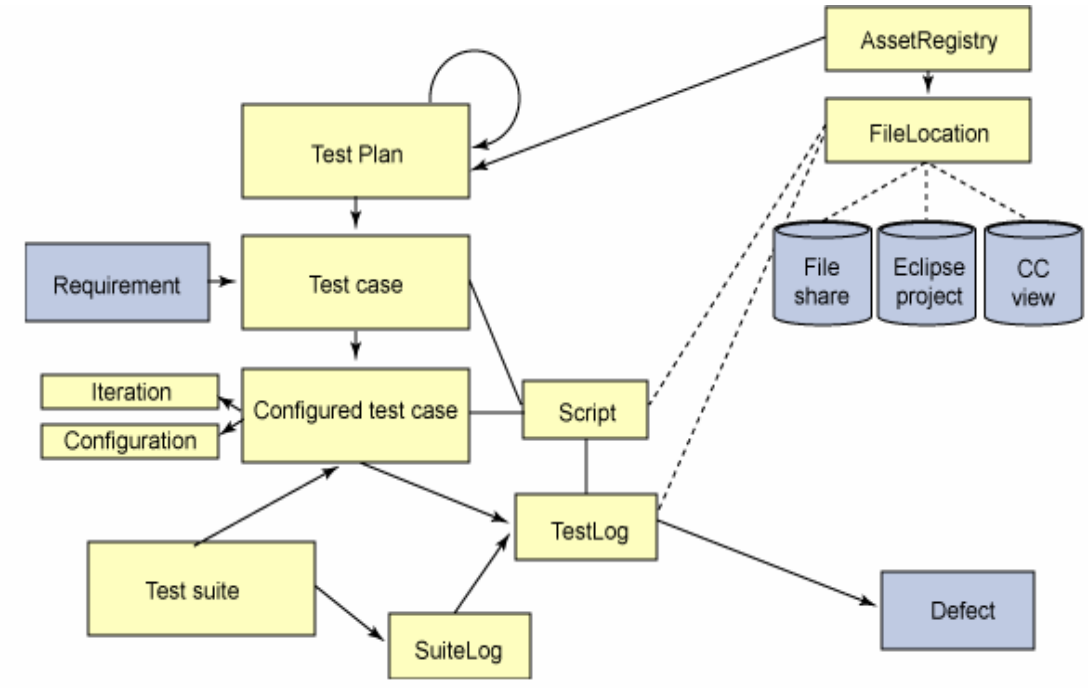


Figure 4.1 CQTM object model [SR]

The objects in blue of figure 4.1 are implemented with the help of the integration between CQTM and other tools, where the Enterprise schema could provide more additional functionalities than CQTM schema.

In the CQTM object model some objects are based: test plan, test case, configured test case, and test suite.

State-based object	States
Test plan	Draft, ForReview, Approved
Test case	Draft, Planned
Configured test case	Draft, Implemented, Blocked
Test suite	Draft, Implemented, Blocked

Table 4.1 State-based objects in CQTM

A state can be changed to another by an action from test lead or test member. An object is in Draft state while being written or defined, in ForReview state during the review process, in Approved state when having already been reviewed, in Planned state when finalized, in Implemented state when executable and in Blocked state when a defect or other issues exist.

The other objects such as iteration, configuration, test script, file location, etc are stateless. They record necessary information and are associated with other state-based objects.

After a configured test case has been executed and test results are committed in the database, a test log can be associated with a defect record which was created before or is created directly from test log form, if test case failed. Although CQTM does not support defect management directly, this additional functionality can be added to CQTM through the customization of CQTM, e.g. the supported defect model in the Common schema is shown in figure 4.2.

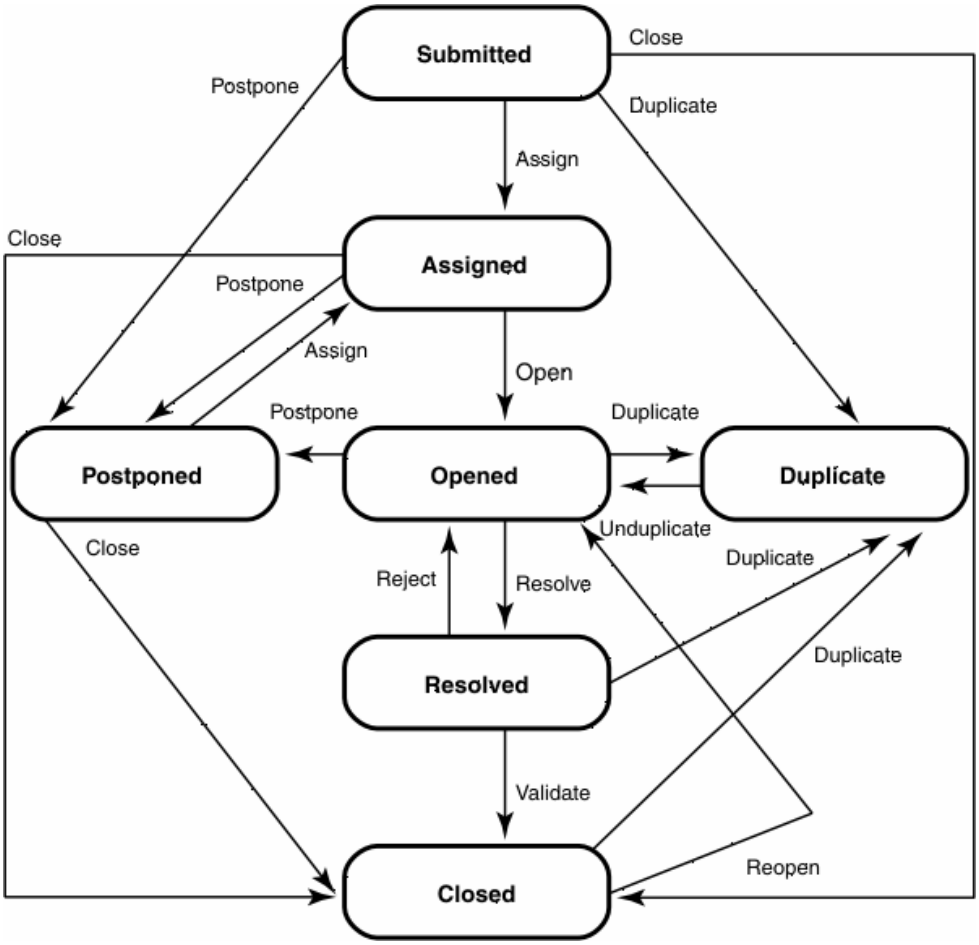


Figure 4.2 CQ Common schema - State model of Defect record type [IRCQ]

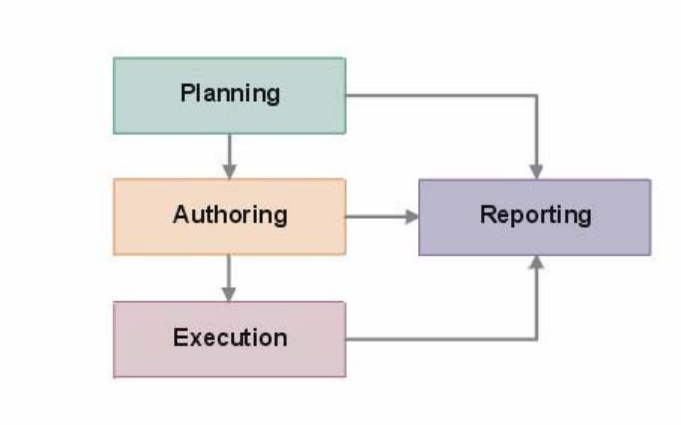


Figure 4.3 CQTM three-phase usage model

CQTM automates the test process; all testing roles – project lead, test lead and testers – are supported. Rational CQTM covers the planning, authoring, and execution phases of the testing cycle (see figure 4.1).

- Planning
  The asset registry record is created first, which contains all the information that describes and implements the test. An asset registry can represent a product or a

release. Test plan, test case, and configured test case records are then created to support the planning hierarchy.

- Authoring
  Test scripts are created and associated with test case and configured test case records. Supported test types for test scripts can be Rational testing tools such as RFT, RMT and RPT, and also the Eclipse Test and Performance Tools Platform (TPTP) test tools such as Junit, Manual test and URL test.

- Execution
  Configured test case or test suite records are executed and then test results are reviewed; If useful, the results are committed to the Rational ClearQuest database to create test log and suite log records.

Reports can be created in each phase to monitor progress of each phase.

This standard schema provided by CQTM supports a test structure solution for test management process, so that not very high requirements on test management could be got in this solution, as CMMI level is not here supported.

4.2 OPAL CQTM Schema R1V4

This testing solution – OPAL CQTM schema release 1 version 4 – is developed with the help of IBM CQTM Standard Schema (see section 4.1) package. The CQTM package has not been applied directly, but components defined in the CQTM package have been reused and imported into the schema (based on CQTM package v2.1). This allows full control of the CQTM components with the ability to customize the solution to be OPAL compliant. In this solution the forms and workflows will be customized. So these processes or workflows from CQTM for test plan, test case, configured test case, and test suit have been improved.

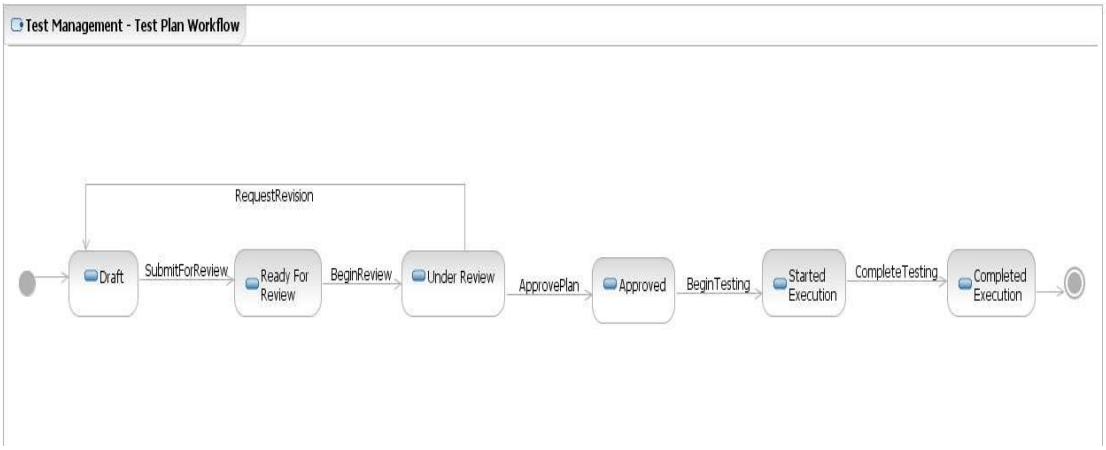


Figure 4.4 OPAL CQTM Schema – test plan workflow

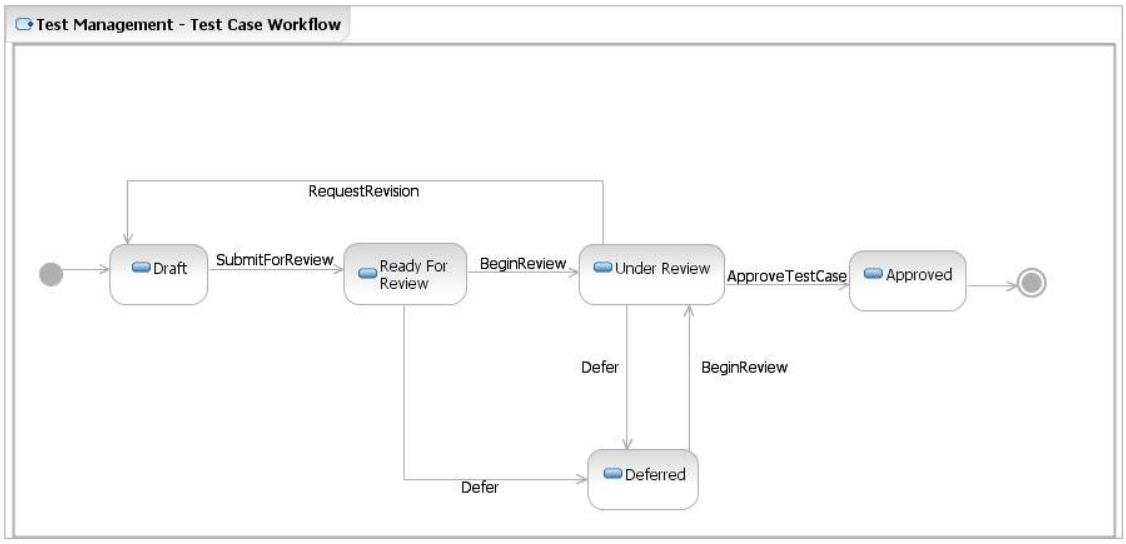


Figure 4.5 OPAL CQTM Schema – test case workflow

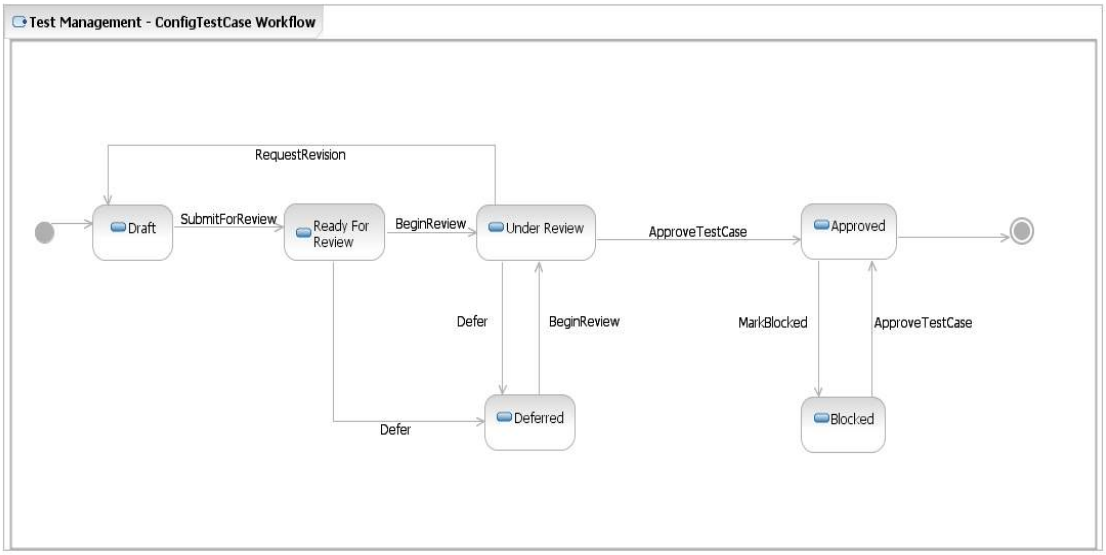


Figure 4.6 OPAL CQTM Schema – configured test case workflow

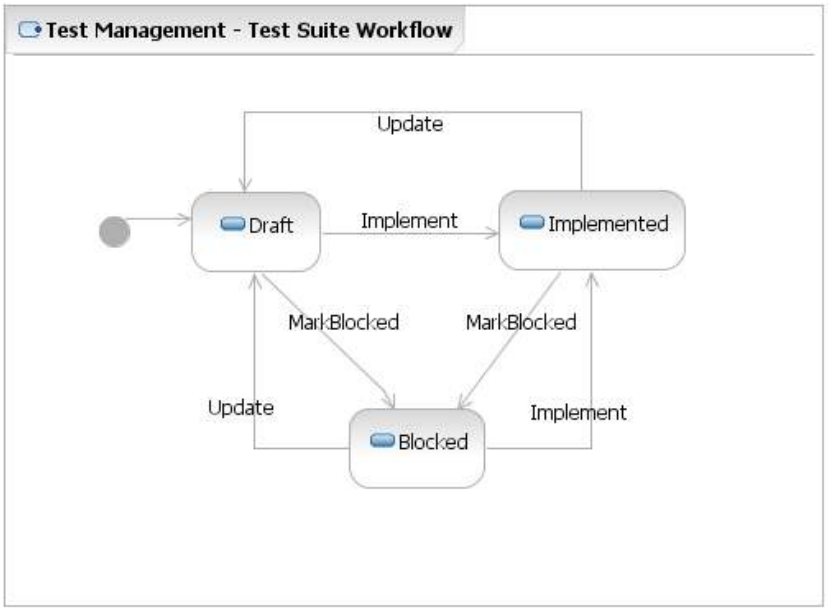


Figure 4.7 OPAL CQTM Schema – test suite workflow

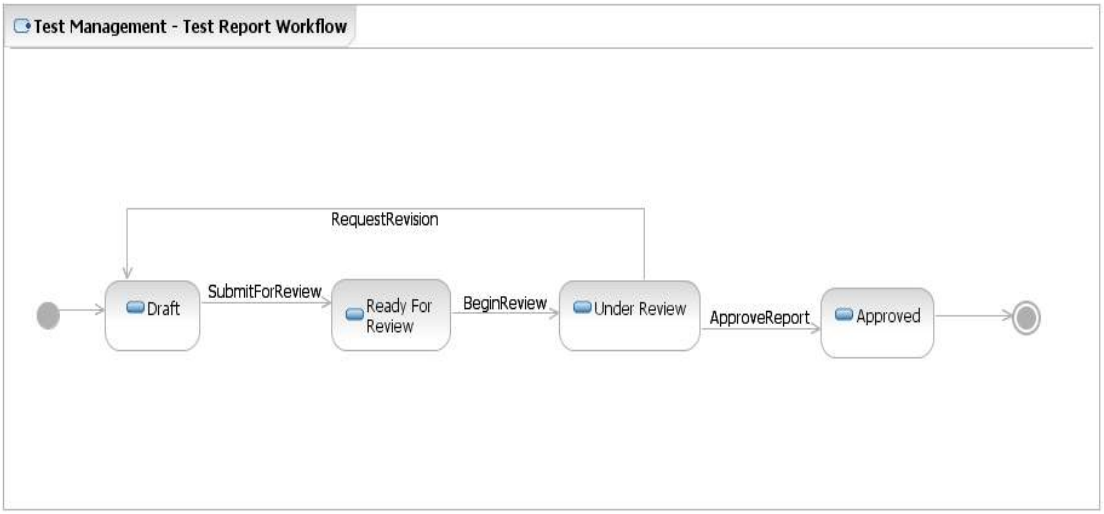


Figure 4.8 OPAL CQTM Schema – test report workflow

In addition to test management of OPAL CQTM Schema, defect management is also improved to be OPAL compliant, where build team, test team, test manager, developer and project manager are involved in the defect cycle. So the better communication and cooperation have been got here in order to find an efficient solution for resolving defects successfully.

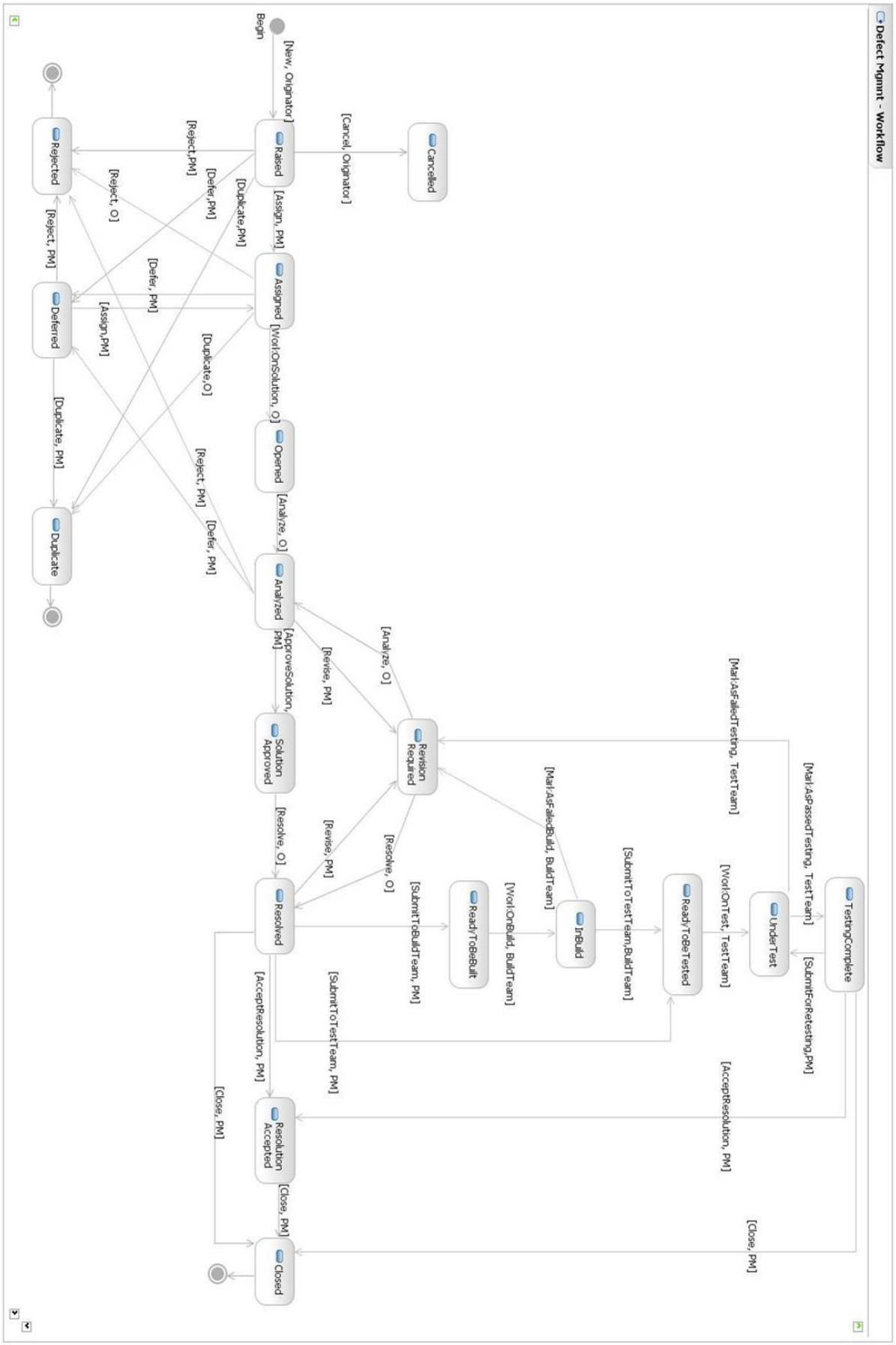


Figure 4.9 OPAL CQTM Schema – defect management workflow

Additionally issue management, change Management and work product inspection are supported well in this solution, too.

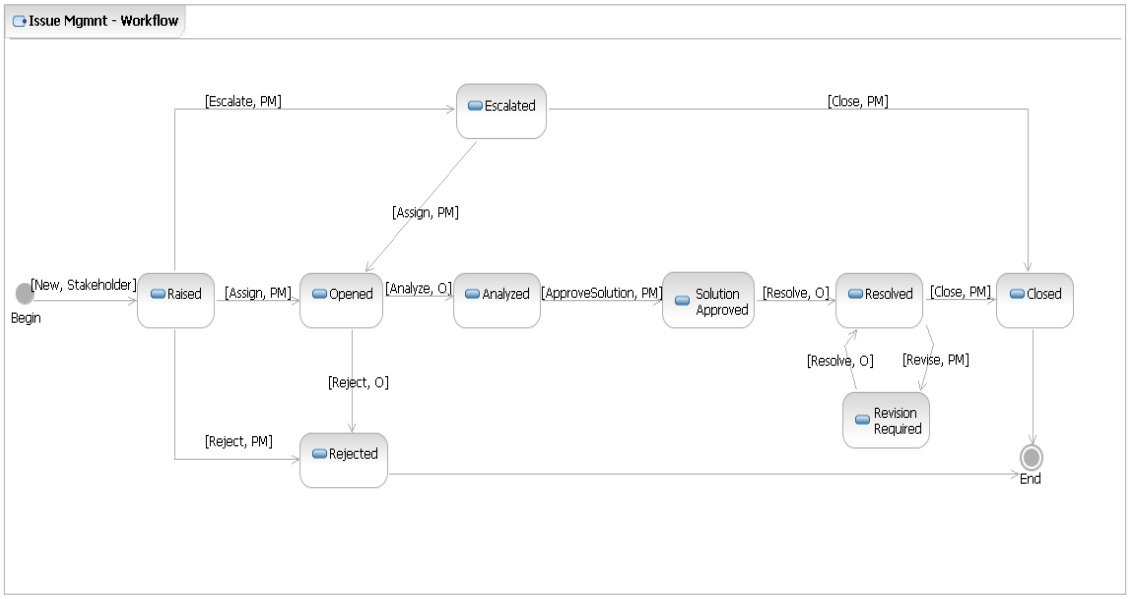


Figure 4.10 OPAL CQTM Schema – issue management workflow

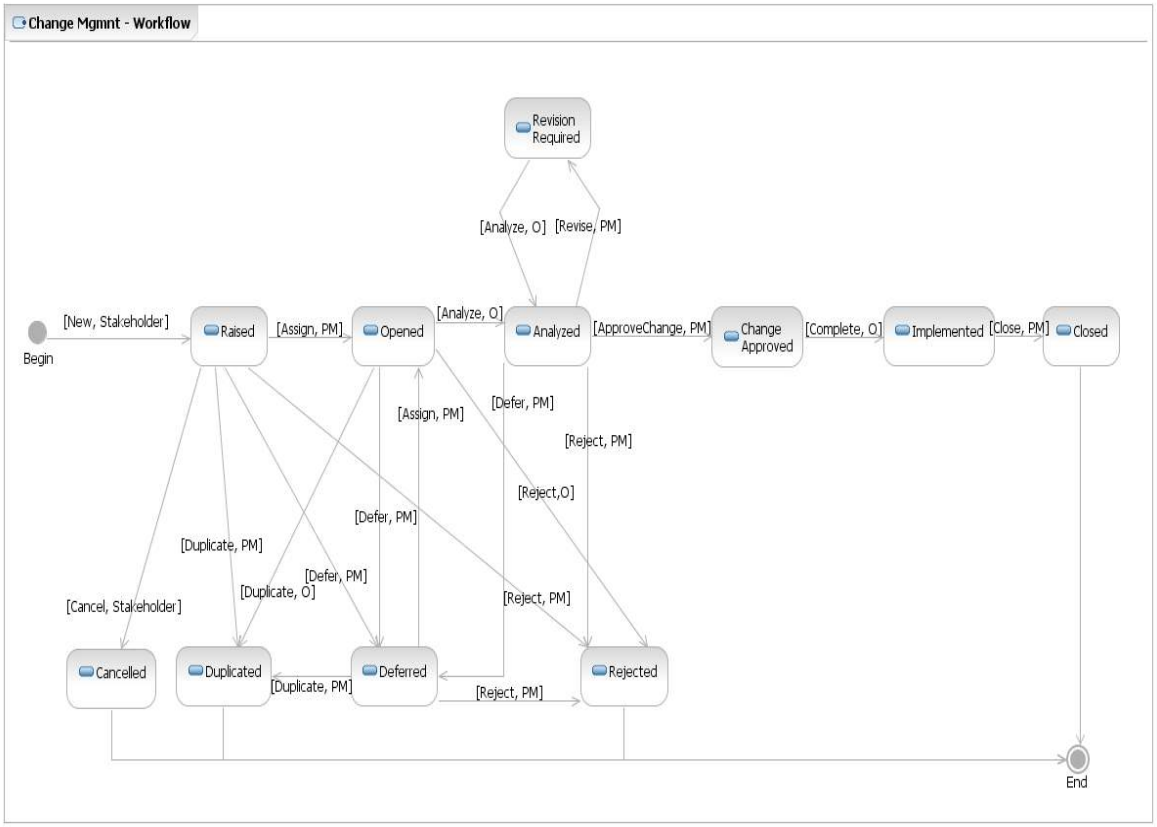


Figure 4.11 OPAL CQTM Schema – change management workflow

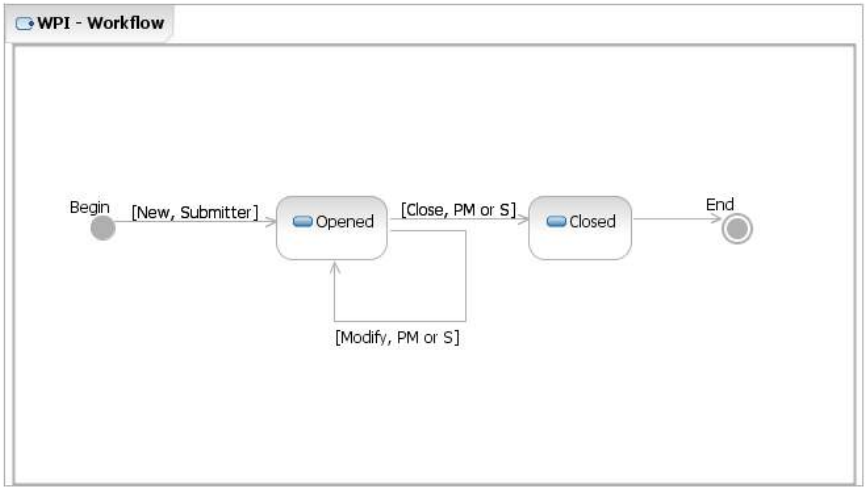


Figure 4.12 OPAL CQTM Schema – Work Product Inspection (WPI) management workflow

The main difference between this solution and CQTM standard schema is not only that other more processes (change management, issue management and work product inspection) are supported here, another very important point is that all these supported processes are OPAL compliant, so that CMMI level 3 or higher will be satisfied, but not for CQTM standard schema.

4.3 Service Management Tool (SMT)

Service Management Tool (SMT) Application is developed in IBM Italy, which provides a standard and automatized solution to support the call, problem and service request management processes within the entire Build & Manage SW life-cycle.

SMT develops a new workflow schema to efficiently design, implement and execute the above 3 mentioned processes. Customer's and IBM's development teams use the SMT interface to daily synchronize their communications. So follows there 3 supported processes are described:

- Call management process: application support management is here supported in figure 4.13. The first level Help Desk of clients or the technical/business analyst receives a phone call from an application user and opens a ticket which is passed to the second level Help Desk, which is managed by the service provider, then the Help Desk analyzes the call, resolves it and informs the client in order to obtain his approval for closing the ticket, if the call refers to an application support, where no code change is necessary. While if the analysis proves that the call refers to a defect correction or a new functionality implementation request, the provider's Help Desk assigns the ticket (in case of defect) or passes it to the Delivery team (development team) for estimation (in case of request). An exception is made for a call already handled before through another ticket; in this case it will be closed as duplicate.

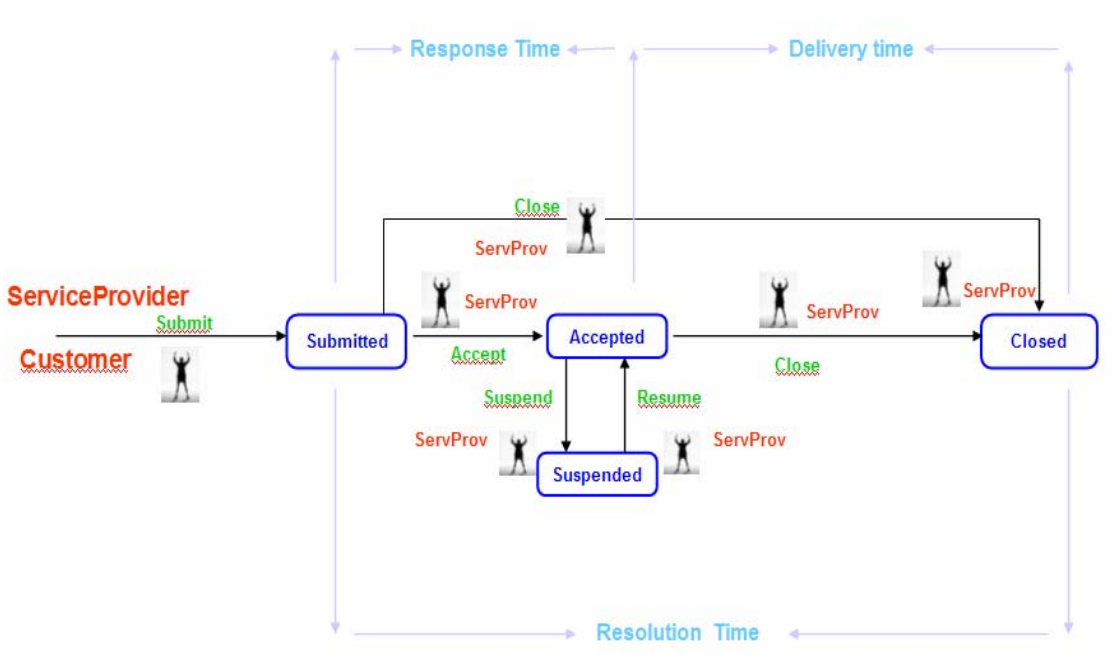


Figure 4.13 SMT – call management workflow

- Problem management process: supports defect management and describes correction of application defects, which occurred in production. A problem is directly opened by the client to the supplier which analyzes and accepts it and then passes it to the provider's Delivery team that resolves it by modifying the code, tests and updates the technical documentation. And finally the same Delivery team passes it to the client that validates the solution and closes the ticket. A Problem can also be closed as duplicate.

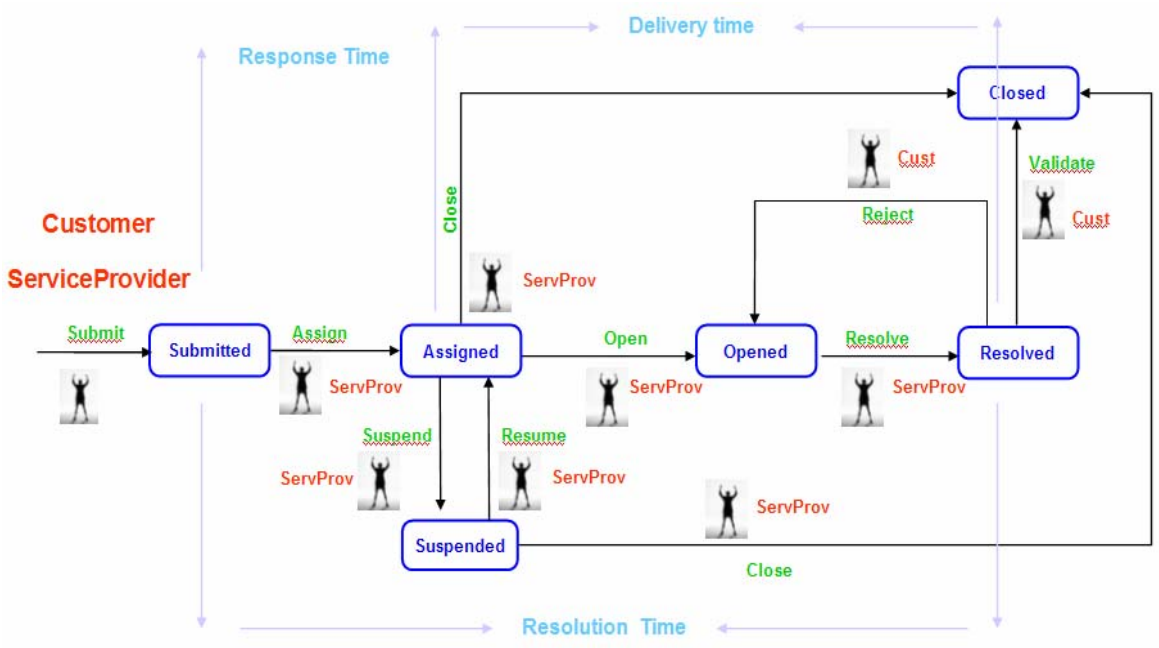


Figure 4.14 SMT – problem management workflow

- Service request management process: covers application enhancement requests. A service request is opened by the client and then moves through a workflow of various statuses, each of which corresponds to a specific action performed by various figures involved in the process: the analyst evaluates it, the client approves it, the delivery team initiates and completes the solution development and finally the client validates and closes it.

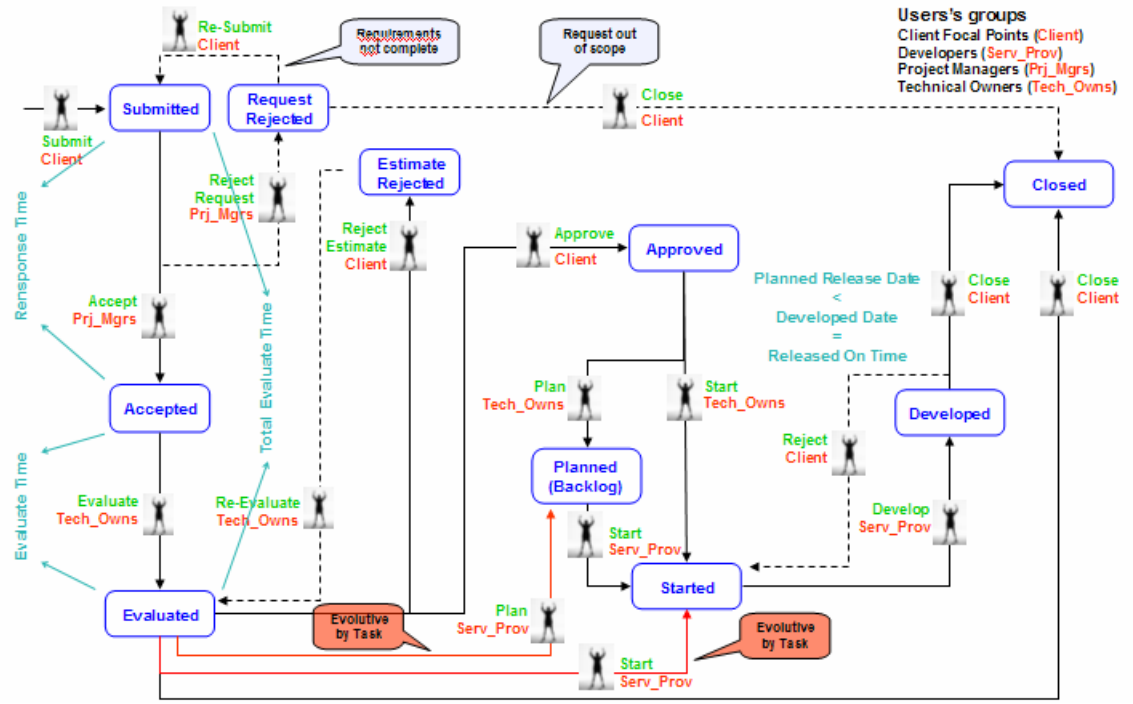


Figure 4.15 SMT – service request management workflow

SMT solution has referenced to CMMI model and the IBM OPAL (On Demand Process Library). A highly customizable defect and change tracking system is designed here to be able to be used to support software test.

4.4 Nordic Generic Solution (NGS)

AMS Nordic Rational CoE provides a solution named *Nordic Generic Solution (NGS)*, which is developed by the AMS Nordic Management Team. NGS supports a complete state machine (see Figure 4.16) handling Service Request Management (SRM) workflows (processes) for the following service request types:

- Support request types (Application support):
  - Support requests
  - Advanced Ad Hoc Support requests
- Defect request types (Defect management):
  - Defect found in production
  - IBM initiated defect (e.g. during system test)
  - Defect found during UAT (user acceptance test)
- Change request types (Change management):
  - Minor enhancement

- Major enhancement
- Project

Although all these request types will be handled in one workflow shown in figure 4.16, depending on specific request type, certain states and actions are blocked, e.g. for IBM initiated defect, the states on the side of clients are blocked, so its real state diagram will be much simpler.

At the same time NGS also implements the workflow for work product inspection procedure defined in OPAL and in compliance with CMMI level 3, so do service request management, defect management, change management.

Test management through CQTM standard schema is also here supported, but not compliant with CMMI level 3. Configuration and requirements management are also implemented in this solution.

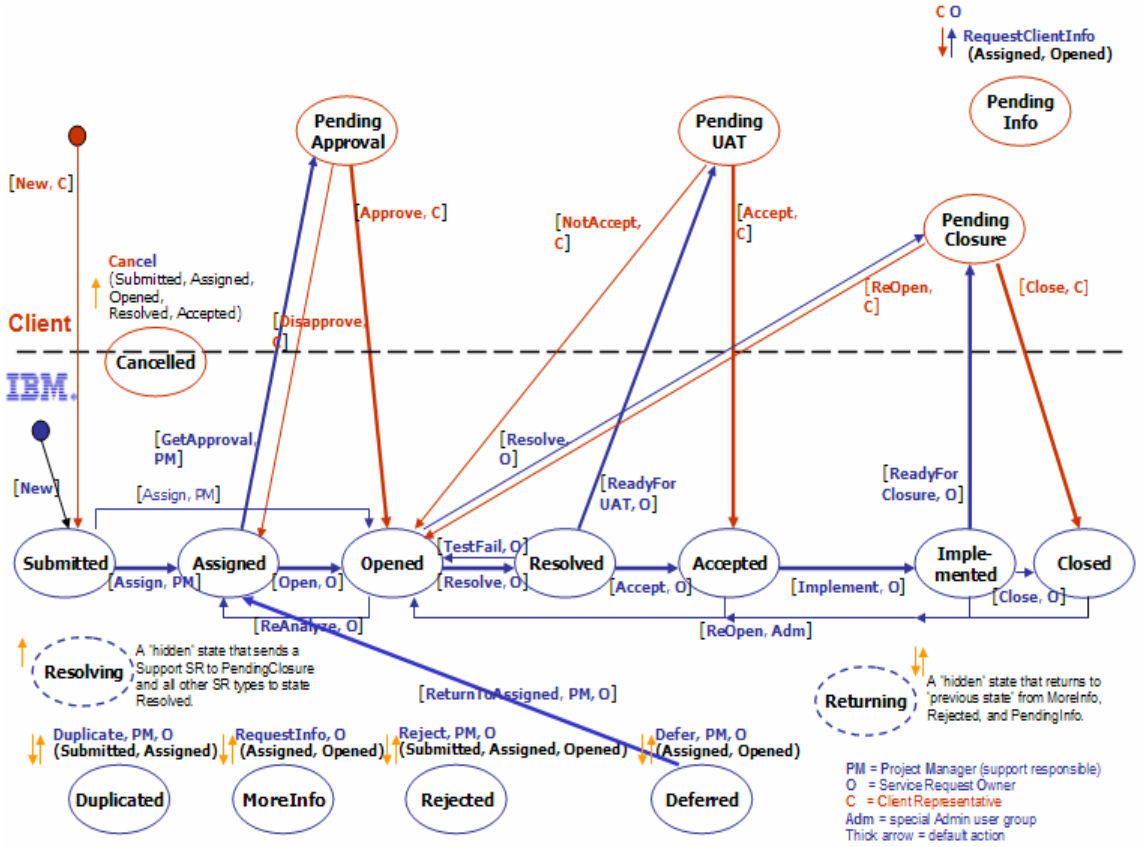


Figure 4.16 SMT – the complete Service Request Management state diagram

4.5 Test Automation Starter (TAS)

Test Automation Starter is from *Project Go!* of *IBM testing services*. This solution provides 4 service packages for software tests:

- Test management package: allows clients to effectively manage test, requirements, change and defect activities. Real-time access is provided to test teams, who are engaged in early test case planning to the project's business and functional requirements, use cases and service level requirements.
- Functional test package: supports in particular functional test and the validation of functional requirements will be automated. The process of functional test

creation, execution and results analysis is realized and to enable the early capture and repair of application errors. So requirements traceability, test planning, test case design, defect recording and tracking, execution recording and tracking are automated.

- Performance test package: supports in particular performance test, one of non-functional requirements. The process of performance test creation, execution and result analysis is realized to find performance problem of software product.
- Life cycle test package: support clean communication between development, quality assurance and IT operations. Requirements traceability, test planning, test case design, defect recording and tracking, executions reporting and tracking, functional test and performance test are automated.

Each of above packages can ensure a high process maturity level and provides an automation solution.

4.6 Summary of is-analysis

From these above mentioned available IBM Assets and testing services, an overview of them is shown in Table 4.2, the following problems can be found during the is-analysis:

- **Slow start of test services**:

Every solution must be setup from the beginning of each test project, that is to say: according to specific projects, each time test services are always carried out newly, as is not very efficient and last too long till test projects begin really. So a basic environment for test projects is required to be made for these test projects.

- **No central management of test projects**

The test projects are performed without central management. Each test project will be executed on different locations, so it is very difficult to manage these test projects well.

- **No consistent global test offering**

All these IBM Assets and testing services have different solutions or emphasises for testing software, e.g. some especially for test management, some especially for defect management, some for functional test etc, so no consistent global test offering is provided.

- **Test services within IBM competing with each other**

Although each IBM Asset or test service has its own emphasises for testing software, there are yet same part of solutions or services provided by them, where they will compete with each other.

- **Varying levels of testing services**

These Assets provide different levels of solutions. Some support CMMI level 3 or higher, some not. Some have concerned to only external customers, some for both IBM and external customers.

At the same time following important advantages or best practices from these IBM Assets can be summarized here:

- **CMMI supported**

CMMI as model for continuous process improvement is chosen by many IBM Assets and test services, where process maturity will be measured and process quality will be ensured by supporting CMMI, so that high quality IBM test services will be implemented.

- **OPAL (OnDemand Process Asset Library) adopted**

OPAL includes GBS tailored procedures and templates that satisfy the corporate practice for project management (WWPMM). OPAL also fulfills part of the requirements of the CMMI model by providing project management and organizational policies, procedures, work products and guidance for delivery organizations. So solutions based on templates or procedures from OPAL support CMMI level 3 or higher.

- **Efficient processes**

Different Assets provide many different processes, but not all these processes are necessary or efficient for all types of projects, so based on different types of test projects, different set of processes with different process quality should be taken, so that requirements will be best satisfied in practice and test projects are performed more effectively and efficiently.

- **Automation and tools**

All these solutions have been or are being or will be implemented through tools, so the automation degree of performing test services is high, as will improve test service quality and be useful to control test effort. So tools, which are selected to implement the solution, play also a very important role.

- **Template-based solutions**

It can be concluded that templates can ensure the quality of the solutions e.g. CQTM is predefined as a template with IBM ClearQuest, and based on the infrastructure of CQTM, another new schema can be developed to implement new functionalities, and then these schemas can be customized or further developed to satisfy specific requirements for different new testing solutions. Template can be useful on all different abstraction levels. It is also very important and necessary to analyze and describe how and which template can be customized or further developed and how the new created template can work properly in the new scenario.

So in the late elaboration of the solution for testing custom software, the above mentioned problems should be considered much and dealt with; best practices and important useful advantages from the is-analysis should be also made good use of, so that the new solution can fulfill the GBS policies and standard and provide a high quality service and solution.

	CQTM	OPALCQTM	SMT	NGS	TAS
CMMI supported	No	Yes	Yes	Partly	Partly
OPAL compliant	No	Yes	Yes	Partly	Partly
Requirements management	No	No	No	Yes	Yes

Defect management	No	Yes	Yes	Yes	Yes
Test management	Yes	Yes	No	Yes	Yes
Configuration management	No	No	No	Yes	No
Change management	No	Yes	Yes	Yes	Yes
Work Product Inspection	No	Yes	No	Yes	Yes
Issue management	No	Yes	No	No	Yes
Application support management	No	No	Yes	Yes	No
Service request management	No	No	Yes	No	No
For IBM	No	No	No	Yes	No
For external clients	Yes	Yes	Yes	Yes	Yes

Table 4.2 Overview of IBM Assets and testing services

5 Virtual Testcenter (VT) for testing custom software

Virtual Testcenter will be introduced and described in this chapter, in order to develop a solution to realize that the basis environment can be made to ensure fast start of software test projects, which can be controlled and managed centrally as well. At the same time requirements on VT will be made with regard to the is-analysis results (see chapter 4) as well as with the help of the survey (see Appendix), which would be based on a catalog of criterions or influence factors, which define typical project situations.

As said in is-analysis results, not all the processes or workflows are necessary and efficient for all types of projects, so through analysis of the survey's results general requirements are defined, and two scenarios about requirements on Virtual Testcenter for small or large projects would be defined differently in addition.

Then a concept for Virtual Testcenter will be elaborated. In the concept the hard- and software infrastructure, process models, service components and so on would be analyzed, defined and described with the emphasis on testing custom software. Two different well suited solutions, both of which would satisfy the general requirements, should be provided for each scenario of specific requirements on Virtual Testcenter (small projects or large projects).

5.1 Introduction of Virtual Testcenter

Virtual Testcenter is needed to function as a concept for providing the basis environment for different test projects, of course different sizes and types of projects have different specific detailed requirements and environments for testing software, but based on the project types, there should also be the general requirements for performing software test. The goal of Virtual Testcenter is to define these requirements (e.g. processes, services) of test projects and tools, which should be developed or customized to support the requirements with the form of Quick Start Templates, hosting and consulting, so that test projects can be executed more effectively and efficiently.

5.2 Requirements on Virtual Testcenter

The requirements on Virtual Testcenter for testing custom software will be summarized by analyzing results of the survey on Virtual Testcenter, where is-analysis results of available IBM Assets and testing services should be regarded and the properties of testing custom software should also be taken into account for elaborating the solution.

This survey is based on a catalog of criterions or influence factors, which define typical project situations and especially have to be dealt with in practice. Many employees in IBM, who have performed test projects for other customers, have also attended this survey. From the results of this survey (see Appendix), as well as based on Best Practices in IBM, the general requirements for all project types are defined and two scenarios about other requirements on Virtual Testcenter for small or large projects would be defined differently, at the same time based on the properties of custom software, particular requirements are given to be a supplement of requirements on VT.

General requirements

For some criterions or questions, much more projects have the same requirements and so these requirements will be regarded as the general requirements for all types of test projects.

- Geographic distributed teams should be supported: team members will always work distributed in different cities or nations in actual test projects.
- Transparent hosting strategy: it should be possible or transparent that the used testing tools are managed and accessed on the side of IBM or customers, at the same time customer data can be stored on the side of IBM or customers.
- Further use of test processes/test tools after finished projects should be supported: the available environment of test processes and test tools should be further used after actual projects are finished.
- Non-IBM suppliers could be involved: in the test projects non-IBM suppliers are also able to be involved. Here in particular the security principle should be defined to allow non-IBM suppliers to access the server. At the same time the complexity of the corresponding process models should be higher if necessary, where the state model could regard more different types of involved roles e.g. tester, builder, developer, IBM, customer, etc.
- English should be used: English is the unique common language in the international teams or projects. Not only for the communication between team members, but also for technological documentations, so templates needed in the test project should be designed in English. (Although in the survey many customers have selected German as project language, English is always used in the development phase and German documentations need to be also translated into English after project finished.)

Specific requirements

For other criterions or questions, no general requirements could be defined or agreed. IBM best practices will also be used to define these 2 scenarios.

Scenario 1 (small projects) about specific requirements

- 5-30 users are supported.
- Basic mind-set of customers regarding test processes should be supported: for here so named small projects, only basic mind-set with regard to test processes is possessed and required, as can be performed more quickly for small teams, so the necessary services should be provided to bring basic mind-set of test processes to the customers (training, testing courses needed, the education or help system), so what roles and whose responsibilities should also be defined in the test project. In addition the terminology used in test projects should be unified for all involved roles.
- Supported parts of the lifecycle: Test- and Defect management process. These 2 processes should be implemented for small projects.

Scenario 2 (large projects) about specific requirements

- More than 30 users are supported.
- Advanced mind-set of customers regarding test processes should be supported: for large projects, higher level mind-set regarding test processes is always needed, so that large projects will be performed as expected, where different

roles would cooperate efficiently. So the necessary services should be provided to bring advanced mind-set of test processes to the customers (training, testing courses needed, the education or help system), so what roles and whose responsibilities should also be defined in the test project. In addition the terminology used in test projects should be unified for all involved roles.

- Supporting the complete lifecycle: Requirements-, Configuration-, Defect-, Change- and Test management processes. These 5 processes are necessary to perform large projects efficiently. As has been known that communication plays a very important role in large projects, where many team members should communicate and cooperate very well, so these 5 processes should also work together logically and effectively.

In addition to scenario 1 & 2 about specific requirements, there is also a specific requirement relating to adopted process models of test projects to support development and test activities and assess and improve the test process quality, so testing maturity level should also be regarded for different types of test projects.

Particular requirements of custom software

Based on the properties of custom software:

- The requirements of customers could change very often (particularly for large projects), so software tests should also take it in thought, efficient and high quality requirements and change management play a very important role to ensure the efficiency of software test.

- Custom software is required and developed to satisfy the specified requirements of the customer, so critical business processes, which are to realize the core requirements and ensure the core competences and the actual gain of the enterprise, are always emphasized necessarily on being tested.

- Custom software is developed for specified customers, so some types of tests e.g. installation test, will not take much effort or be required, because customer software not like standard software needs not be run for so many types of operating systems and hardware and so mass market.

These above mentioned requirements describe “what” should be supported or realized according to specific types of test projects. It does not mean that specific test projects such as small test projects will not need other processes (e.g. Requirements-, Configuration- and Change management) at all in practice (the customer would have the own method for other processes), but from the view point of providing testing solutions or services and based on the concrete requirements from industry project praxis, it makes sense to define the 2 scenarios separately and develop two different test project “templates” for them, so that test projects could have an accelerated start-up rather than having to make it up or “re-invent the wheel”.

5.3 Solutions of Virtual Testcenter

Two different solutions will be described for scenarios of small projects and large projects, as is based on the requirements on VT (see section 5.2). In order to satisfy these requirements, the solutions should regard software/hardware infrastructure, process models, service components and so on, at the same time the 11 key processes and as-

pects of software test, which have been outlined in section 3.2, will be referenced to analyse and define the solutions. The result of is-analysis of available IBM Assets and testing services is also important to construct them here.

The emphasis of the solutions in this section lies in the construction of the processes and service components in each scenario and their relationship, the other technical requirements such as HW/SW infrastructure will be realized in the next chapter for realization of VT.

5.3.1 Scenario for small projects

The requirements on process quality and complexity for small projects

The most important factor for small test projects is to make them be executed quickly and effectively, so only some necessary processes will be needed here and the process guide should not be stringent, neither, because of the low complexity of small projects, their corresponding process and service components need not be very complex. Man can also say, the selected processes need not support CMMI. So the description of the processes needed for small test projects can be based on those for large test projects, where but not all of work products are necessary in practice and their complexity is also low, the workflow is also not so complex and only some necessary tasks from it will be performed based on actual need of the concrete typical test projects. Normally many work products except important and more complex work products, are not necessary to be reviewed formally, as their complexity is low, in order to accelerate the project execution.

The process and service components for small projects

The needed processes and service components for small projects would be described in the following such as:

- Test management
- Defect management
- Critical business process testing procedure
- Training process

Based on the above specified requirements of process quality and complexity for small projects the description of these processes can be referenced to those for large project (see 5.3.2 scenario for large projects), so only some important guidelines will be given for each process.

Test management

Each small test project has its own emphasis on the specific part of all test needs, test levels and test types. All these work products, which can be useful in the test management, would not be created so stringently, such as test strategy, master test plan, detailed test plans (e.g. static test plan, unit test plan, integration test plan, system test plan, acceptance test plan and so on), test environment, test execution plan, test specification including test scenario, test matrices, test case, and test script, test results, test measurements and test reports. Some of them can be recorded mixed together in a document, and normally many work products except important and more complex work products

(e.g. test plan) are not necessary to be reviewed as their complexity is low in order to accelerate the project execution.

Defect management
The status model of the defect must not be so complex and the number of the involved roles for managing defect is not so high, so that the duration from submit to resolved would not be so long and the project would be executed quickly.

Critical business process testing procedure
This process can be identical with the one for large test projects, as critical business processes are so important, regarded as the core requirements and competences of the customer and the actual gain of the enterprise, that they should be tested effectively and efficiently and all the documents (e.g. acceptance test plan and cases) about software test needed should be created in the test management.

Training process
The basic mind-set of customers regarding test processes is needed and trained for the project team in the classes, such as only the above mentioned 3 processes (test management, defect management and critical business process testing procedure) and their related documentations are mandated, so not much training effort will be taken before the test project does begin really.

5.3.2 Scenario for large projects

The requirements on process quality and complexity for large projects

For the large projects much more processes will be necessary than small projects and the process guide should also be stringent, where more work products are needed, because of the high complexity and high risk of large projects, their corresponding process and service components need be complex and of high maturity, so CMMI should be supported by these processes. Thus these processes will be compliant with OPAL and in particular tailored and customized with much effort to perform the test project for custom software, so that best practices from the is-analysis of available IBM Assets and testing services can be made good use of. In addition, the uniform terminology for the test project for custom software should be got. The relationships between these processes should also be described at the effective and efficient communication points during the large test project.

The process and service components for large projects
The needed processes and service components for large projects would be described in the following such as:

- Test management
- Work product inspection
- Defect management
- Issue management
- Change management
- Configuration management
- Requirements management
- Critical business process testing procedure

- Training process

In each process the purpose, the workflow, team roles and work products will be described. In particular in the workflow there are always some tasks involved to show what is to be done, which work products will be required or created during each task, all these components mentioned in each process are necessary to be performed or created during the large test project.

Test management

Purpose

The objective is to understand and execute the overall needs for test planning, execution and management, where the test planning hierarchy, which consists of test strategy, test plan, test environment, test specification, test report, test results and so on, is to be managed.

Workflow

There are 5 tasks in the workflow, which are described based on the sequence in the following:

1. Define test strategy

Applicable other work products can be used to identify the test strategy, such as agreement with client, business requirements specification, system requirements specification, project quality plan and so on. Test strategy is a high level system-wide expression of major activities that collectively achieve the overall desired result as expressed by the testing objectives. It starts with the high-level description of the "what" to be tested.

In test strategy all strategy statements are expressed in high level terms of physical components and activities, resources (people and machines), types and levels of testing, schedules and activities. The strategic plan will be specific to the system being developed and will be capable of being further refined into tactical approaches and operating plans in the detailed test plans for each level of testing.

In most situations, these are merely definitions of terms and terminology, used to establish a common understanding and lay the foundation to initiate the next step, which is test planning.

2. Develop test management plans

Based on test strategy and by reviewing other available work products (e.g. agreement with client, business requirements specification, system requirements specification, project quality plan and so on), the master test plan is developed to describe "what" (scope and objectives), the "why" (purpose), the "when" (key milestones), and the "who" (organizational roles and responsibilities) for testing custom software, in addition detailed test plans for each level of testing will be also created particularly for complex, large projects, such as unit test plan (normally created and maintained by the developers), integration test plan, system test plan, acceptance test plan and so on, as there is no adequate detail in the master test plan to support the ongoing management of the test activities. The acceptance criteria are as well addressed in developing these test plans. At the same time test environment is also defined in this task, which describes all of the aspects required to establish the target environment to support the type/level of testing being planned.

3. Prepare for testing
Based on test strategy and master test plan, technical specifications are used to prepare for unit and integration testing and functional specifications, system requirements specifications and architectural specifications are adopted to prepare for system and acceptance testing.

Then test specification is prepared, which must contain adequate detail to demonstrate traceability and coverage with reference to the requirements and the resulting test cases. The purpose of test specification is also to verify and validate the traceability and coverage of the tests against the original business requirements and technical specifications. The test specification work product uses the various functional and non-functional requirement documents along with the quality and test plans. It provides the complete set of test cases and all supporting detail to achieve the objectives documented in the detailed test plan. In the test specification the following components will be defined:

- Test Szenario

  A test scenario defines, at a high-level, how a given business or technical requirement will be tested, including the expected outcome. Free form text can be used to describe test scenarios, illustrative graphics may be used to help clarify understanding. It has an assigned priority (e.g. high, medium or low) that reflects the importance of the requirements fulfilled by the scenario to the business. A test scenario may be derived from the functional requirements as specified by use cases and/or non-functional requirements, such as peak load handling, response time expectations, and access control needs. It can be used to evaluate the testability of requirements, and to identify any ambiguous, incomplete or missing requirements. The test scenario description may include references back to the requirements for purposes of traceability. A test scenario will typically require the execution of multiple test cases to demonstrate its full functionality. The test scenario description may include references to those test cases.

- Test Matrices

  Test matrices are tables that are used for a variety of purposes in test design. Any cell, which is checked, indicates that it is used or is applicable. Test matrices can be used to help to ensure adequacy of test coverage, avoid oversights, prioritize test cases, illustrate and communicate what is being tested and not tested and demonstrate traceability from requirements to test conditions to test cases, and the reverse. For example, functional requirements matrix can document the functional requirements for the application system at a high level, test requirements coverage matrix can be used to illustrate that all requirements are covered by one or more test cases.

- Test Case

  As the smallest possible test entity, a test case is an action required to satisfy a test condition. It is the simplest data that can be processed in conjunction with a test bed state and execution pre-conditions to generate an output that is measurable against a predicted outcome. Test case is used to specify the test case conditions, test data, test script, and expected results.

- Test Data

Test data is the input data and test bed data associated with a specific test condition. A test bed is a set of test files, including databases and reference files, in a known state, used with input test data to test one or more test conditions.

- Test Script

A test script contains the detailed sequence of manual and automated actions, as well as the setup information to execute a test case. This content includes, but not limited to, listing specific file names, programs, transaction names, file layouts, specific keystrokes and control cards. A test script contains sections for test setup, actions or procedures to complete, expected results and actual results.

An addition the requirements traceability and verification matrix or other document to associate the requirements to the test cases is updated, which that shows how each requirement will be tested, when each requirement will be tested, and the test acceptance criteria for each requirement. (for detailed requirements see requirements management)

A test execution plan is prepared to use the test specification and design details to plan what tests will run, in what order, with what infrastructure, environment and data support, and who will run them. As the test execution plan is based on test environment, detailed test plans and test specification and design work products, it is critical that proper configuration management keeps all the documents involved in sync.

A test execution plan can help clearly identify the components required to effectively execute the tests, schedule the test runs and resources required for them, set up the defect tracking logs, complete and close off testing activities and provide this detail for each level of testing. In particular for test cases, between which there are interdependencies, the test execution plan is necessary.

When test data is prepared to ensure the test conditions identified in the test specification and the test environment is prepared, the test readies review can be performed.

4. Execute tests

In this task the test execution plan is implemented, where the test cases are scheduled and executed in accordance with test execution plan and all results are captured. In the testing results any variation from the test case's expected results are logged as a defect, which will be analysed and managed in accordance with defect management.

When defects are resolved with defect management, tests will be re-executed and the retesting results are captured to update the corresponding documentation of defect management. Test process will be reported based on the section of the test strategy and assessed against the defined exit criteria to determine the completion of testing.

5. Report on testing

During this task test reports are created to show status against test and quality targets at any point in time, at specific milestones, and/or at the end of testing. A test report summarizes the actual testing, documents details about any deviations from the original plan and the reason for the deviations, and also covers any new risks encountered. Through test reports it can be determined whether testing at a particular level has been satisfactorily completed

Test reports can also be created for internal checkpoints as they are snapshots of current status for testing activities. If needed the reports can be reviewed and approved, or can be kept in draft mode or deleted before approval if not formally required by the project. It's a project decision which of the test reports created should follow the formal workflow and which ones can be avoided.

When all appropriate levels of test that the test strategy has required have been completed, the exit criteria is met for all test levels; the final test report is created and reviewed with customers, who require the custom software, obtaining all necessary approvals and acceptance statements.

Team roles

There are mainly two roles for test management:

Test manager

This role coordinates all test management activities within the project. They are responsible for delivering the test project on time and within budget while meeting all quality objectives. The 3 tasks of the above workflow: define test strategy, develop test management plans and report on testing, are mainly performed by test manager.

Test manager manages not only the test team but also the relationship with other team leaders in the project. Communicating the test project status, resolving the test project issues, ensuring effective test process improvement feedback occurs and so on are the tasks of test manager, too.

Tester

This role (also named test specialist-technical) tests the applications and systems from a technical perspective. The 2 tasks of the above workflow: prepare for testing and Execute tests, are mainly performed by tester.

This role participates in test planning activities, developing the test design, evaluating technical elements for testability, participating in the definition/review of acceptance criteria, developing the technical test scenarios/test cases for verifying the system meets the intent of the design from a technical perspective and participating in testing of solution.

In addition to test manager and tester, developer would also be normally responsible for creating and maintaining unit tests.

Work Products

Many work products have been created and used by test manager and tester in the test management, such as test strategy, master test plan, detailed test plans (e.g. static test plan, unit test plan, integration test plan, system test plan, acceptance test plan and so on), test environment, test execution plan, test specification including test scenario, test matrices, test case, and test script, test results and test reports. At the same time many work products, such as agreement with client, business requirements specification, system requirements specification, project quality plan, project charter and so on, have been referenced to create the above work products.

Through different tasks of the workflow as can be seen which work products will be inputted and outputted by which role during the task.

Work product inspection

Purpose

The defects of the work products are found and removed as early as possible in the software development lifecycle, which could be all the artefacts of the development cycle e.g. customer business requirements, system/component requirements, system/component architecture, project quality plan, project charter, etc, and here also particularly the work products that are produced in here specified processes such as test strategy, test plan, test execution plan, test specification, test case, test script, defect management documentation, change management documentation and so on. The manual test methods are performed during this process, in order to ensure that the completed work products will satisfy customer requirements and the standards for development.

Workflow

There are 5 tasks in the workflow, which are described based on the sequence in the following:

1. Preparation for the review

When the to be reviewed work product has been completed and all other specified criteria for readiness have been met, a review method will be selected based on the size and complexity of the work product and the risk level of the project, where there are normally two types of review methods: the formal review method (e.g. named facilitator method) performed by more project team members for more complex work products and the other informal review method (e.g. named buddy method) performed by not so many project team members for less complex work products; both methods would compare characteristics of a work product to an expected set of attributes and standards.

The duration of the review is estimated and the participants and their roles of the review are determined such as the number the participants, the roles (e.g. project manager, moderator, author, supervisor) and customers if necessary. The relevant materials for reviews are also prepared and distributed to the participants such as work product to be reviewed, list of standards to be complied with, specification on which the work product was based, programming guidelines for code inspections, instructions on how to record defects, issues, risks and preparation effort and any other important background documentation.

Normally an overview of the work product is provided if the review team is not familiar with the work product, or is unaware of important design rationale and associated issues, so that the review can be executed more effectively. Then the review could be continued at 2.task – conduct the formal review or 3.task - conduct the informal review based on the earlier selected method.

2. Conduct the formal review

For the facilitated review of the complex work product, the inspector needs to prepare for the review before the meeting: the review materials are read and the work product is reviewed, line by line, to identify issues and defects. The project requirements and any applicable standards (see Section 3.2.8) should be used as the basis for the inspection. If any defect is found, it will be identified and classified as severity e.g. 1, 2, 3, 4 or 5. The defected can be recorded directly in the work product or in another document such as the work product inspection form or the defect management documentation. Comments,

issues, risks, precise clarifications and effort spent preparing for the review are recorded and submitted to the inspection moderator or facilitator earlier if necessary.

After preparation the meeting can be opened, the purpose of the meeting is to identify the defects, not to discuss solutions and suggestions for improvement. During the review of each section inspectors will point out and discuss defects in the work product, where defects with too low severity are not discussed. After the discussion of each raised point, it will be determined whether a defect in accordance with the defect management or an issue in accordance with the defect management is documented. The inspection should be conducted in a timely manner and work product inspection form is updated with the effort spent performing the review, along with other summary information such as the participants, the designated inspector, moderator and possible process improvements before an exit decision would be made (5.task). If the work product needs to be reworked then go to 4.task – rework the work product, then a new work product inspection will be requested later.

3. Conduct the informal review
For a buddy review of the less complex work product the duration is shorter and the preparation for the review is not necessary. During the review the work product is stepped systematically and the inspector's questions are answered. To every possible defect a short discussion is performed to determine whether a defect in accordance with the defect management or an issue in accordance with the defect management should be documented. Work product inspection form is updated with the effort spent performing the review, along with other summary information such as the participants, the designated inspector, moderator and possible process improvements before an exit decision would be made (5.task). If the work product needs to be reworked then go to 4.task – rework the work product, then a new work product inspection will be requested later.

4. Rework the work product
The work product with the defects, which have high enough severity to be removed necessarily, are returned to its suppliers; after these accepted defects have been resolved in accordance to the defect management, the changes to the work product are reviewed by the designated inspector to ensure that all the defects have been corrected, then the work product is accepted, the work product inspection form is updated and the rework is completed.

5. Close work product inspection
When the inspection results (defects, issues, etc) have been addressed appropriately, the status of the review can be updated as closed in the work inspection product form.

Team roles
All relevant project members (developer, tester, author, project manager, recorder, presenter, etc) would perform the tasks - preparation for the review, conduct the informal review and rework the work product. The moderator and inspectors (also named facilitator) would conduct the formal review and normally only the project manager can close work product inspection.

Work Products
In addition to defect management documentation and issue management documentation referenced here, the work product inspection form plays the most important role to pro-

vide a permanent record of the detailed description of the defect found during the work product inspection, its disposition, status and root cause. The to be reviewed work product during various phases of the lifecycle is of course the input of this process.

Defect management

Purpose

The monitoring and resolution of defects, which are found in deliverables during the testing process and other inspections, reviews, etc of the project lifecycle, are managed, where all defects will be resolved with minimum impact to the project. Audit trails of defects are also maintained and analyzed. The information, which is needed to perform root cause analysis of defects at the project level, is got in order to improve the quality of both the work product and the process during which it was created.

Workflow

There are 7 tasks in the workflow, which are described based on the sequence in the following:

1. Identify, raise and log defect

The defect is every type of deviation between is- and expected results. When a defect is found during the testing process, reviews or inspections, it will be identified and logged in the defect log. The defect will be validated, if it is not valid, the defect management will exit; if it is valid, so it will be continued to be dealt with.

There are also defects which are introduced based on the following cases:

- A change has resulted in this defect

  A change is needed to resolve the defect, e.g. change to project requirements, system specification, work products and so on. Assessing and managing change can be executed through the change management.

- An issue has resulted in this defect

  An issue is an internal matter of concern for the project, identifying and managing issues can be performed through the issue management.

The defect management documentation is created to record and report the defects and could consist of defect severity, defect type, status, title, description, resolution, responsibility and so on.

2. Accept defect and assign responsibility

The defect is accepted and assigned to a resolution owner (e.g. developer) for further investigation, where in defect management documentation the corresponding resolution date and severity code are defined and the resolution owner is informed of the defect.

3. Analyze the defect and record action for defect resolution

The defect management documentation is used to analyze the defect and the affected scope of the organization, project and code. Then the actions and schedules for resolving the defect are defined in the defect management documentation.

4. Review actions and decisions

The recommended actions and schedules are reviewed whether they are appropriate for the project. The final appropriate actions will be approved, prioritized and assigned.

5. Resolve defect

The defects are resolved by the resolution owner through the appropriate actions based on their assigned severity and the project prioritization, and then the fix should be tested to verify the defect is really fixed as desired. This verification may include the creation of a specific build and/or the verification of a number of test cases – eventually by a test team different from the development team. The corresponding defect management documentation is updated regarding the defect status.

6. Track and manage outstanding defects to closure
The outstanding defects with high prioritization are much important to the work product quality and need be tracked with more effort in order to resolve them as quickly as possible. The defect management documentation and defect log are reviewed periodically.

7. Analyze and report defects
The defect data from the defect management documentation is analyzed and summarized, and the analysis data, status and measurements data for defects are reported, where root cause of each of the defects is also analysed.

Team roles
The originator (typically tester) will perform the task - identify, raise and log defect. The test manager or the project manager could identify, raise and log defect, accept defect and assign responsibility, review actions and decisions and track and manage outstanding defects to closure. The resolution owner (typically developer) would analyze the defect and record action for defect resolution and resolve defect.

Work Products
The most important work product is the defect management documentation, which comprises the detailed information of each of the defects. Defect reports are useful to analyze the project status.

Issue management
Purpose
The identification, report, analysis and management of issues, which could be found in different phases of the development lifecycle, are performed. The issues are assigned to the appropriate resolution owner for resolution with minimum impact to the project. Audit trails of issues are also maintained and analyzed.

Workflow
There are 4 tasks in the workflow, which are described based on the sequence in the following:
1. Perform initial analysis of issues
Issues that can be identified from reviews, team meetings, discussions, etc, are reviewed first in order to have a quick and clear understanding of the details associated with the issues. The issue is prioritized based on its level of impact to the project and recorded in the issue management documentation, which consists of many other attributes of the issues such as originator, issue description and consequences, resolution, category and so on.

It is determined if the identified issue is the scope of the project: if not, the issue would be transferred to the appropriate organization or project and then closed; otherwise the appropriate solution actions and responsibility are analyzed for the accepted issue at the

2.task - assign responsibility and determine resolution actions, where a change request can be raised in accordance with the Change Management, if the potential impact of the issue is too severe that project cost, schedule or quality could be affected adversely.

2. Assign responsibility and determine resolution actions
The appropriate resolution owner is assigned for the issue in order to determine the resolution to implement, which would improve the ability of the project to meet its commitments and whose effort should be consistent with the priority. The issue management documentation is updated.

3. Monitor progress of issue resolution
The status of each issue not yet closed is assessed by obtaining status updates and any pertinent information from the assigned resolution owners. When the resolution has been implemented, it is verified that the issue is corrected and the corresponding status and associated data in the issue documentation are updated.

4. Analyze and report issues
From the issue management documentation the necessary statistics about the issue are made including such as number of open issues, number of resolved issues, number of recurrent issues, total number of raised issues, number of issues with other status (e.g. in progress, closed) and so on. All these statistics can be analysed and compared with other projects to identify trends and candidates for root cause.

Team roles
The project manager is mainly responsible for all these tasks in the issue management; the resolution owner (development team, test team or any other role of the project) is asked to resolve the issues. Any project member could raise the issues related to his phase.

Work Products
The issue management documentation would be created and used through this process to record the details of an issue in order to support its analysis, follow up its resolution and make the reports of statistics.

Change management
Purpose
Change management plays a very important role to ensure the quality of custom software, where all changes (such as cost, effort, content or schedule) are identified, recorded and tracked. Each request for change will be assessed, approved or rejected by the appropriate stakeholders, affected groups (e.g. test team) are aware of the status of all changes and approved changed are implemented, and all plans (e.g. test plan) and affected work products or deliverables are updated.

Workflow
There are 6 tasks in the workflow, which are described based on the sequence in the following:
1. Raise change request
When the need for a change to a project has been recognized, the appropriate project manager should be notified of this need, providing him/her with all available details regarding the need for a change to the project, including the reason and the nature of the change being requested.

2. Perform initial assessment, reject or accept for analysis
Details of requested change to the project are reviewed and the change management documentation is created to consist of all required information of the change request, then it is determined whether to accept or defer or reject the change request, if accepted, the analysis of the request will be performed to determine the size and impact scope of the request, so the corresponding test plans, other schedules, cost and quality of the project are updated.

3. Determine impact and potential solutions
During the impact's analysis the scope of the requested change should be determined. During the assessment of the impact to the project the project manager and the team should review active issues, risks and actions, identifying any relationship with the requested change; they also identify the first lifecycle phase affected, gather and consolidate the assessment results for all solution options.

4. Obtain formal agreement and schedule change orders
Based on the information of the change management documentation, the change request is agreed by the appropriate approvers and the recommended approaches for implementation to the identified approvers are presented. The authorized option for implementation of the accepted change request is made.

5. Implement change
Once a change request has been approved the resolution owner (e.g. developer) is able to start working on the implementation, at the same time the tests are performed to check if the change is really completed.

6. Close change
Once the work required for completing the change request is finished and the change has also been verified and validated by test team, the completion in the change request should be recorded and the change request is closed.

Team roles
Customers who require the custom software would always raise change request, the project manager and other stakeholders would be responsible for the tasks: perform initial assessment, reject or accept for analysis, determine impact and potential solutions, implement change and close change; the development team and test team are also active for the completion of the accepted change.

Work Products
All the information about the change requests are recorded and updated in the change management documentation. The service request is created during this process so that the project can implement it later.

Configuration management
Purpose
The integrity of all the work products during different project phases such as test plans, test cases, test scripts, requirements, source code, etc, which are subject to change by the project is established and maintained. It is ensured that any version of a work product that has been placed under configuration management control can be recreated at

any time to define and organize the elements of a system. Soft copy versions of all testware for testing custom software are managed in this process, e.g. all the work products in the test environment are configured automatically to backup and restore functions for testing any version of the work products.

Workflow

There are 10 tasks in the workflow, where tasks 1-5 for establishing project configuration management, tasks 5-10 for performing configuration management, which are described based on the sequence in the following:

1. Define the application/project inventory

Identify the inventory of software, documents, and hardware and supply items as appropriate to be controlled during the project lifecycle. Each project must decide which software and non-software work products are subject to configuration management, based on a project-defined criteria documented in the project configuration management plan. In the project inventory the identifier of the component to which it belongs, a description of the project, the identity of the client application owner, the start date of the project and the status of the project (e.g. planned, under development, released) are defined. The configuration items of the project are defined, where each configuration item record would contain the application development organization's unique identifier (conforming to naming standards), a statement of the function, purpose and scope of the configuration item, the original creation date and name of the project that created the configuration item, the current status of the configuration item, the last implemented project change request number or project that affected the configuration item. In addition the association of configuration items and projects are identified.

2. Establish and approve the project configuration management plan

The project configuration management plan is used to identify the configuration management responsibilities involved in the project, the activities to be performed and the specific schedules in which they will be performed. The plan also identifies the configuration items for the project, their relationship with other configuration items, and their control. Tools or practices to be used for configuration management are also specified. The sections of the plan need to be completed according to the concrete phase of the configuration management preparation. Finally the plan is distributed to all affected groups for review and approval.

3. Authorize the configuration baseline

The project or application inventory and the associated configuration items are reviewed and authorized. All changes are approved to items identified as controlled by the configuration management plan, for each baseline in which the configuration item participates.

4. Implement the library management system

The configuration management repository structure, disaster recovery and security plans in the configuration management plan are designed and documented; the library management system (LMS) is implemented by installing the LMS tool, adding application files to the repository, implementing functional security and implementing repository security. After testing repository operation, the LMS is activated for the project.

5. Create workspaces

The initial software baseline audit is conducted by comparing the contents of the software libraries to the planned controlled configuration items, to ensure the completeness and accuracy of the configuration items. The integration workspaces can be created to allow the teams to have separate test, build and preparation areas prior to delivery. Then the development workspaces (including e.g. development of work products for software test) are created to allow the developer or test team to check out and work on configuration management items prior to returning to the configuration management library. Only items that are defined as part of a configuration management baseline are kept in the development workspace.

6. Control configuration items

The changes to a configuration item are controlled:

- Include a new element

  The new element is mapped to its configuration item, and then the source of the element from the library designated by the author is copied into the staging library of the LMS.

- Check out an element

  An element is checked out of the LMS when required, the identity of the borrower, the project, and the check-out date are recorded on the configuration item record.

- Record changes to an element

  Changes to an element is recorded when required, an audit trial is created to log the reason for changes to configuration elements into the system. Every change should be justified, recorded and approved; each change could originate from the project's change management, defect management, issue management and requirements management, so all changes are able to be tracked back to a specific requirement.

- Check in an element

  An element that was checked out earlier is checked in when required, the configuration items that control the elements by checking the mapping is located. The check-in date and the associated project change control documentation or defect or problem report number is recorded. The item is moved from the staging area to the staging archive and then from the borrower's designated libraries to the staging library. If the element is to be deleted from the configuration item by this change, then mark it as deleted in the map.

7. Build and release change

All of the configuration items affected by the release change are identified, and the list of them is compared to the product integration plan and the interface specification to ensure that all products have been included. During the build procedure the resultant product (for both internal and external use) is ensured to be built from elements in the baseline library, and also ensured that previous versions can be recovered. For this new iteration as build or release regression test is normally performed to ensure that changes have not inadvertently caused unintended effects to the baselines. The release change number, the authorized name and date are recorded in the configuration item record. After approval a copy of all new versions of documents is provided to the person responsible for distribution, and then all affected elements are moved from release to archive and from staging to release.

8. Record baseline status changes

Based on the relevant approved change management documentation the change to the baseline status can be approved, where the status of the configuration item of the base-

line is revised accordingly. Then add to the application/project inventory at release, the completed revised mapping of elements to configuration items. Baselines change to include newly approved baselines and to delete existing baselines.

9. Revise configuration management plan
Configuration management actions are recorded in sufficient detail so the content and status of each item is known and previous versions can be recovered. The relevant customers are notified of the configuration status of the configuration items. The latest version of the baselines and the version of the configuration items of a particular baseline are identified. The differences between successive baselines are described. The status and history of each configuration item are revised using the configuration management plan.

10. Perform continuing baseline audits
During the project it is important to perform continuing baseline audits to ensure the completeness and accuracy of the configuration management records. The LMS contents are compared to the controlled configuration items as specified in the configuration management plan to assess the integrity of the baseline.

Team roles
The most tasks mentioned in the configuration management are performed by configuration manager. The project manager would establish, complete and approve the configuration plan.

Work Products
The configuration management plan comprises the tasks of configuration management and is used to identify the specific configuration management concerns, define what the plan will and will not address and identify the items to be managed. The plan is an ongoing document. Initially, the configuration management activities and the configuration items are identified and entered into the plan. Then, the individual segments of the plan are updated as the plan evolves. The library management system is used to document a change in state to the configuration management system, such as implementing or updating it.

Requirements management
Purpose
The requirements of the customers and projects are identified, traced and refined throughout the development lifecycle. All requirements should be consistent with customer business requirements and based on understanding the stakeholder needs, expectations, constraints, and interfaces. The agreed deliverables and their acceptance criteria are clearly related to the requirements. In particular in the test project it will be managed that how each requirement will be tested, when each requirement will be tested, and the test acceptance criteria for each requirement are defined.

Workflow
There are 4 tasks in the workflow, which are described based on the sequence in the following:
1. Obtain the understanding of the requirements
The requests of the custom software are reviewed to determine a total picture of the requirements, which comprises such as an overview, functional requirements, non-

functional requirements, usability requirements, interface requirements, etc. Each requirement is documented with requirement identifier, description, priority, acceptance criteria, and so on. The requirements should be correct, clear, concise and consistent with each other; the testability and traceability of the requirements would be taken into consideration.

Based on the requirements verification checklist criteria the requirements then would be reviewed (see work product inspection) with the customers to reach agreement on a clear understanding of the scope, objectives, requirements and associated acceptance criteria, where defects, issues and risks from the review are tracked, managed and resolved through their corresponding defect and issue management. So the requirements baseline will be established and later changed in accordance with the 4.task - baseline and track requirements, where these requirements will be traced and tracked throughout the development lifecycle and will become the basis for verification and testing activities.

2. Manage changes to requirements
Changes to requirements could happen often during the project of custom software, then in accordance with change management, the changes to requirements are managed, where change management documentation that affects requirements, business requirements specification, system/component requirements specification, requirements traceability and verification matrix are updated and approved.

3. Obtain commitment to requirements
It is determined what groups are affected by the requirements or requirements changes and the impact they will have on existing commitments by the affected groups is assessed, e.g. the test team would plan to perform test based on these requirements, a commitment is obtained from the affected group.

4. Baseline and track requirements
The baseline of each level of requirements and architecture documents is created by placing them under configuration control in accordance with the configuration management, as each baseline is approved. The baselined documents have been formally reviewed and agreed upon and can now be changed only through formal change control procedures – change management, as includes any documents produced, and any related source documents or attachments. Each requirement is tracked through the life cycle of the project using the specified requirements management tool or the requirements traceability and verification matrix, until each requirement is delivered, withdrawn or transferred to another project, e.g. tracking requirements through the test phases and updating the status of the requirement on an appropriate time and/or event driven basis.

Team roles
The customer plays a very important role in the task - obtain the understanding of the requirements, as the requirements of the customers are the fundamental and motivation of performing the project of custom software. Project manager would execute all of the tasks in this process.

Work Products
The requirements verification checklist consists of a series of questions (such as correct, clear and concise, manageable, consistent with each other, relevant, testable, traceable,

feasible and complete) and is used to review the requirements and system work products to verify they are complete and correct. Requirements traceability and verification matrix is a document that is used to verify that all of the requirements are traced and implemented by the custom software. It consists of customer or business requirements, system and component requirements, etc and traces them through the test project.

Critical business process testing procedure

Purpose

Integrated testing across all business critical processes, which support the core business of the customer, is performed to ensure cross-process conflicts are detected in the testing environment so that system disruption is minimized and the core business of the customer is ensured in order that the custom software can be moved into the production environment finally.

Workflow

There are 6 tasks in the workflow, which are described based on the sequence in the following:

1. Determine critical business processes

At first it is determined which critical business processes are required to ensure cross-process conflicts are detected so that system disruption is minimized. Business processes are sets of related activities, which result in the achievement of business objectives. The critical business could be such as financial accounting, controlling management or accounts payable. The customer will have the final decision on which critical business process need be tested based on the documentation of requirements management.

2. Document details of testing

The components will be identified to execute the required testing effectively such as:

- Environment: identify the system in which testing will occur, and any supporting data or log on IDs.
- Execution: identify test cases and/or test scenarios, plus the tasks and resources needed to physically set up and run them.
- Scheduling: identify order in which testing will occur, plus any dependencies.
- Logging: identify activities and resources needed to record the details of the test run.
- Issue management: identify procedures to evaluate risks, issues, changes, and adjustments to the test plan when needed.

3. Execute integrated test plan

All the integrated test cases and/or test scenarios are performed with the guideline of the test strategy.

4. Document integrated testing results

The final testing results are documented to summarize the findings (function working or not working), the issues for further analysis and recommendations. In addition number of test cases executed, number of defects found by severity, status of defects, etc. are also measured and recorded in the testing results.

5. Execute acceptance test plan

All acceptance test cases and/or scenarios are performed with the guideline of the test strategy.

6. Document acceptance testing results
The final testing results are documented to summarize the findings (function working or not working), the issues for further analysis and recommendations. In addition number of test cases executed, number of defects found by severity, status of defects, etc. are also measured and recorded in the testing results.

Team roles
The project manager would perform the task - determine critical business processes with the help of the customer who will have the final decision on which critical business process need be tested. Project manager or test manager would document details of testing. Tester will perform the tasks - execute integrated test plan and document integrated testing results. Customer will perform the tasks - execute acceptance test plan and document acceptance testing results.

Work Products
Required critical business processes would be determined and recorded with the customer; Test strategy, integrated and acceptance test plan and test results, which can be created and maintained with test management, are used and created for testing critical business processes.

Training process
Purpose
All the related members in the test project of custom software need to be educated or trained so that they possess the needed ability and qualification to ensure test quality. The test organization training and skill needs are identified, where different roles of the test project are educated. The education of above described process models or assets which support the whole process of the test project, new concept or technology adopted in the project, different test documentation, test standards and norms, test automation tools such as IBM Rational Tools and so on should be performed in this process.

Workflow
There are 5 tasks in the workflow, which are described based on the sequence in the following:
1. Establish the training program
The training policy is defined and recorded, where the training needs are identified such as standard set of processes of the project, skills appraisals. The training objectives and records are documented in the training program charter, as involves determining the skills needed to achieve the business objectives, identifying focus areas (critical skills) and evaluating needs against the organization's current skills. Training requirements will be documented in order to satisfy the training objectives: roles such as test manager, tester, etc and recommended training for each role, approved sources of training, training types and training facilities.

2. Create the training plan
Based on the training program charter the training plan is created to identify in detail all activities and responsibilities related to the development, delivery and evaluation of education and training for the test project. The training plan can be used to track the completion of the training. The cost for performing training plan would be determined.

3. Coordinate the delivery of training
The training requirements are provided to the training department for classes that will be delivered for the project members. For any new training courses, it is determined which roles and individuals within the organization must complete the training. If an individual has already completed the equivalent training before or his previous work experience renders the training of little value, this individual must not complete the mandated training. The sources of training are identified and approved, such as electronic, book learning or in-person. Completed training evaluation feedback would be collected to update the training plan.

4. Deliver the training
According to the schedule in the training plan, the training is delivered by the trainer. Changes to the courseware materials are recommended based on the completed training evaluation feedback. Normally for the large projects higher level mind-set regarding test processes will be trained.

5. Monitor the training program
The status of training program activities is reported and the effectiveness of the training is assessed by analyzing training evaluation feedback. By analyzing the training program status it can be determined if training goals are being met or some corrective actions need be taken as necessary. All the training documentation and materials will be managed under the configuration management with version control.

Team roles
Skills resource planner, who determines the resources that are needed in what quantities to perform project activities, would perform the tasks - establish the training program, create the training plan, coordinate the delivery of training and monitor the training program. The training would be delivered by the trainer.

IBM will normally provide the whole training service for customer (could be IBM internal) who will perform a test project of custom software. All the related members in the test project such as test manager, project manager, tester, configuration manager, developers and so on would attend the training and receive each corresponding education.

Work Products
The training program charter is a description of all the elements required by an organization in establishing and maintaining the training program, the training plan would identify in detail all activities and responsibilities related to the development, delivery, and evaluation of organization education and training for the test project. The completed training evaluation feedback and other reports would be used to improve the training plan and other courseware materials. All the training documentation and materials about the processes, assets, tools, technology, the to be tested custom software if necessary, etc. are needed in the training classes.

6 Realization of Virtual Testcenter (VT) for testing custom software

The last chapter has analyzed and elaborated general requirements, specific requirements for small or large projects, and particular requirements on VT. The two different solutions for scenarios of small projects and large projects have been given with the emphasis on analysis of process and service components of VT, so in this chapter the realization of the automation of these processes and other requirements on VT would be performed. At first the tools, which can realize these requirements, are to be selected, where in particular the appropriate IBM Rational Tools will be selected and analyzed, then the infrastructure is to be set up to provide the whole analysis, description and elaboration of realizing VT, also including two different scenarios for small and large projects.

So IBM Rational Tools, which are used to realize the Virtual Testcenter, will be introduced. Then the infrastructure will be elaborated to automate the processes, which have been determined in the last chapter, and provide the HW/SW infrastructure to satisfy the technical requirements on VT.

6.1 IBM Rational Tools

6.1.1 Rational ClearQuest (CQ)

Rational ClearQuest is a process and workflow automation tool designed for all sizes of development teams, the organization's specific processes and tasks in the dynamic environment of software development can be automated by customizing Rational ClearQuest, such as defect tracking, change tracking, etc. In addition the reporting and lifecycle traceability of the status of change activity created during the process are supported for better visibility and control of the software development lifecycle. The predefined E-mail notifications are able to help enhance team communication and coordination.

The concept - change request is used in Rational ClearQuest to record and track any type of change activity (such as defect, documentation modification) associated with the software development project. Change requests are stored as records in user databases and each record type for a change request defines the fields, forms, and state transition model associated with the change request, where actions can be taken to move the change request from one state to another.

The complete description of the process model for one type of change request is realized by a Rational ClearQuest schema. This includes a description of the states and actions of the model, the structure of the data that can be stored about the individual change request, hook code or scripts that can be used to implement business rules, and the forms and reports used to view and input information about the change request. [IRCQ]

The schema is a pattern or blueprint, which defines the way the data is stored and changed, for Rational ClearQuest user databases. When you create a database to hold user data, the database follows the blueprint defined in a schema. However, a schema is

not a database itself: it does not hold any user data about change requests, and it does not change when users add or modify data in the user databases. [IRCQ]

All the schemas can be saved in a special type of database called a schema repository. A schema repository can store multiple schemas for different types of change requests, and every schema can have multiple versions. The collection of user data for one process model associated with a specific version of a specific schema is in a user database, which is an instance of a version of a schema. As the change request moves through its lifecycle, the data stored in its record changes accordingly.

So the schema plays a very important role to realize the automation of the process and workflow specified in a project or organization. How to develop a schema to implement the required processes effectively is a very important task (see the next section).

The main roles that the customers perform while using Rational ClearQuest are user role, schema developer role and administrator and each role has its allowed tools to accomplish the tasks. It is possible for the users to access Ration ClearQuest across multiple platform by using a web browser submit, modify, and track change requests, and analyze project progress by creating queries and reports. Rational ClearQuest provides also possibilities to integrate with other products based on the requirements of the projects.

For the developers at different locations Rational ClearQuest Mulitsite can help them to use the same database set (a schema repository and its associated user databases): each site would have its own automated, error-free replica set (copy of the database set) and changes made in one replica will be sent in update packets to other replicas.

6.1.2 Rational ClearCase (CC)

Rational ClearCase is an enterprise software configuration management (SCM) tool designed for medium to large development teams, where requirements, models, source code, documentation, test scripts and other software development assets are controlled. It handles version control, parallel development, workspace management, process configurability, and build management. It also provides advanced build auditing and a Web interface for universal data access. Rational ClearCase remote client can be used to access Rational ClearCase from a variety of network connections. Through the use of scripted rules, ClearCase manages and enforces the organization's development process. [IRCC]

The following basic terminology is used in Rational ClearCase:

- **Element**: there are two types of elements: file element that is any file (e.g. source codes, test scripts) stored in a file system and directory element that contains file elements and other directory elements.
- **Version**: a specific revision of an element.
- **Versioned object base (VOB):** a secure repository that stores versions of file elements, directory elements, derived objects, and metadata associated with these objects.
- **Check out-edit-check in model**: enables team members to manage changes to the project. When an element is checked out, an editable copy of the element is

created in the view. When an element is checked in, a new version of it is added to the VOB.

- **View**: is represented as a directory and provides access to a specific version of one or more elements in a VOB. With a simple rule-based approach, a view selects a set of versions of elements without having to specify the versions explicitly.

Rational ClearCase supports two types of software configuration management: **Base ClearCase** and **Unified Change Management (UCM)**:

Base ClearCase [IRCC]
It covers the version control, configuration control, and part of the process management areas of the SCM domain. Base ClearCase allows a very high degree of flexibility to develop the own system of managing and tracking software resources, but it also means that more efforts has to be put into the design of the configuration control and the design of process workflows.
Base ClearCase terminology consists of:

- **Branch**: is an object that specifies a sequence of versions of an element. Each element has one main branch, which represents the principal line of development. Each element may also have multiple subbranches, each of which represents a separate line of development. Branches are used to implement parallel development.
- **Label**: is attached to any version of an element to identify that version in a meaningful way. Labels can be applied to a set of elements to mark important project milestones or to a specific version of an element to indicate the proposed starting point of a branch.
- **Configuration specification**: contains rules used by a view to select versions of elements. The rules are very flexible, and various criteria such latest created or label rule, can be used to indicate which versions of elements the view should display.

When base ClearCase is adapted for a project, a branching strategy, how to merge work between branches, creating standardized configuration specifications and labels, metadata to implement these project policies should be defined or performed.

Base ClearCase can be integrated with Rational ClearQuest to associate change requests with versions of elements. Versions associated with a change request constitute a *change set*.

UCM [IRCC]
It covers the version control, configuration control, process management and problem tracking areas of the SCM domain. UCM raises the level of abstraction to manage changes in terms of activities, rather than manually tracking individual files. UCM automatically associates an activity with its change set, which encapsulates all project artifact versions used to implement the activity. This enables project team members to easily identify activities included in each build and baseline. In using UCM the combination of CC and CQ provides full integration of activity-based development with process management and problem tracking.

UCM terminology consists of:

Project: a logical unit that is mapped to the development structure of an application or system and contains the configuration information (e.g. components, activities, and policies) needed to manage and track work on a product. A typical UCM project in Rational ClearCase consists of one shared work area and many private work areas (one for each developer).

Component: a group of file and directory elements (source code, test scripts and other relevant files) that are versioned together. A UCM component is developed, integrated and released as a unit. Components constitute parts of a project and can be shared by multiple projects. VOB can contain one or more components.

Activity: an object that records the set of files (change set) that a developer creates or modifies to complete and deliver a development task, such as a defect fix.

Stream: an object that maintains a list of activities and baselines, and determines which versions of elements are shown in the view.

Work area: a development area associated with a change and consists of a view and a stream in using UCM. A project contains one main *project integration stream* that records the project's baselines and enables access to the shared elements at the UCM project level. The integration stream and a corresponding integration view represent the project's primary shared work area. In most projects, each developer on a project has a private work area, which consists of a development stream and a corresponding development view. The development stream maintains a list of the developer's activities and determines which versions of elements appear in the developer's view.

Baseline: identifies one version of each element in a component that represents the integrated or merged work of team members. A baseline represents a version of a component at a particular stage in project development and will be created to reflect project milestones.

Composite baseline: a baseline that selects baselines in other components, when project team works on multiple components. By using a composite baseline in this manner, one baseline can be identified to represent the entire project.

The full functions of UCM can be provided through the combination of CC and CQ. When using UCM for a project, components, stream hierarchy, development policies and naming scheme for baselines need be decided.

For the developers at different locations Rational ClearCase Mulitsite can help to enable parallel development across them, each site can have its own automated, error-free replica of project databases (VOB) and it is transparent to access all software elements and artifacts. With update mechanisams changes made in one replica will be propagated to other replicas.

6.1.3 Rational RequisitePro (RP)

Rational RequisitePro is a requirements management tool that helps teams manage project requirements (can be IT projects or non-IT projects) comprehensively, promotes

communication and collaboration among team members, and reduces project risk, in order to improve the software development process and product quality. [IRRP]

Some predefined types of project requirements such as feature, supplementary, use case, and glossary terms are provided, where the specific requirement type and its attributes (e.g. Priority, Status, Cost, Difficulty, and Stability) can also be defined by team members. In Rational RequisitePro, a project includes a database, where document types (can also defined such glossary document, vision statement, use cases, test plan), requirement types, requirement attributes for description, discussions, and information about requirement traceability and user and group security are stored. All the requirements are stored in the project database. Documents can also be included in a project. Microsoft Word is used to create requirements documents, in which requirements can be described. Three kinds of views such as attribute matrix, traceability matrix and traceability tree are used to view requirements, set attributes (such as priority), and establish relationships between them.

Rational RequisitePro includes a web interface, which makes all project team members to be able to access requirements especially in remote locations or multiplatform environments. RequisiteWeb allows team members to access, query, modify, and create requirements by using a web browser.

Rational RequisitePro can be integrated with other Rational tools (e.g. CQ) to provide the association of RP requirements with other elements or records.

6.1.4 Rational Functional Tester (RFT)

Rational Functional Tester is an object-oriented automated functional test and regression test tool, where in particular Graphical User Interface (GUI) test and data-driven test are supported automatically.

Rational Functional Tester provides capabilities to generate scripts quickly by recording applications against the application-under-test. The scripts can be modified to enhance or make specific changes if required, and then the user can play back these scripts to repeat the same sequence of actions that was performed earlier during recording. So a test case will be realized through a script and performed to verify the functionality of the application based on the log of the scripts. In iterative development model regression test is supported for each iteration to ensure that both existing and new functionality work as expected.

Many types of applications such as Windows, .Net, Java, HTML, Siebel and SAP applications will be supported with test automation by RFT. In two integrated development environments and two scripting languages RFT is available; either Java in Eclipse or Microsoft Visual Basic .NET in Visual Studio .NET is able to used to author and customize test scripts.

Rational Functional Tester can be integrated with other Rational Tools to provide more efficient functions.

6.1.5 Rational Performance Tester (RPT)

Rational Performance Tester a performance testing tool used to identify the presence and cause of system performance bottlenecks. Multiple users can be emulated to test the performance of the application.

Rational Performance Tester provides capabilities to create tests by recording representative interactions with an application. The recorded actions constitute a test, which can be played back to inspect the results, in order to make sure that the tests are doing what as expected, otherwise the tests can be edited to enhance or make specific changes if required, and then a schedule can be created, which consists of user groups, appropriate tests assigned to each group and other schedule items such as loops, delays and think time settings, to emulate a heavy load. The schedule is executed, the reports during the execution are created and the results are evaluated.

The load testing against a broad range of applications such as HTTP, SAP, Siebel, SIP, TCP Socket and Citrix is supported in Rational Performance Tester. Rational Performance Tester can be integrated with other Rational Tools to provide more efficient functions.

6.1.6 Rational Manual Tester (RMT)

Rational Manual Tester is a manual test authoring and execution tool used to improve testing quality. Key-driven manual tests can be automated by Rational Manual Tester.

Test scripts can be created through the use of keyword libraries, where test statements or groups of statements can be shared. During the manual execution of scripts, the application being tested and the test script are shown at the same time, and each test instruction is performed and whether certain conditions pass or fail will be verified. The tester can type comments, take screen captures, and submit, resolve, or verify defects as the test progresses. In the end of tests the results are saved in a test log, where summary information about the test, detailed results for each test statement, information about associated defects, any comments typed during the testing, and any accessory files are contained. Results can be exported to a spreadsheet application for further analysis.

Rational Manual Tester can be integrated with other Rational Tool to provide more efficient functions.

6.2 Infrastructure of realizing VT

The infrastructure will be elaborated to realize Virtual Testcenter, where the appropriate Rational Tools will be selected and analyzed to implement the requirements on VT for two different project scenarios. Template concept, which can be of many types including documents, process models, physical data stores, should be here taken advantage of to realize particularly the automation of processes of VT in order to ensure not only the fast-start of software test projects but also the quality of the solutions. The mapping and terminology of the selected IBM Rational Tools to realize the specific processes need be analyzed and described, such as the implementation of the process models, in order to elaborate how the implementation with the tools can work properly in accordance with the processes. In addition the hard- and software infrastructure is to be set up to provide the technical environment to realize the technical requirements and process automation and make it possible to provide hosting and consulting services.

6.2.1 Scenario for small projects

In this scenario it will be elaborated how the specific processes defined in Section 5.3.1 for small projects are realized with Rational Tools, and then the hard- and software infrastructure is constructed to provide the technical environment to realize the requirements on VT for small projects.

Realization of process models for small projects

The process and service components for small projects have been defined in section 5.3.1 such as Test management, Defect management, Critical business process testing procedure and Training process.

As has been known that Rational ClearQuest is the process and workflow automation tool designed for all sizes of development teams, so the here required test management and defect management can be automated with CQ, and how a process can be automated with Rational ClearQuest would be analyzed in the following:

The schema in CQ is to be developed to realize the automation of the process and workflow specified in a project or organization, where the complete description of the process model for one type of change request is outlined. The schema in CQ and its elements can be defined and customized as templates to represent process models and the necessary work products during the processes so that the effectiveness and efficiency of process are ensured. So in order to develop a CQ schema, the process model of a specific change request should be designed at first, where its state transition model would be developed based on its properties. It should be defined that which roles have which privileges to take what actions to move the state to another in the state model in order that the change request is resolved or closed as expected. Each change request will be associated with a record type in CQ, which consists of fields and forms to describe the attributes or information of a specific change request in addition to above mentioned state transition model. At the same time a record type could be also stateless and comprises only fields and forms except the state transition model in order to represent the static element (necessary information) used in the process model and can be associated with state-based record types.

Hooks are able to written to customize the workflow of a record type and have four types such as field hooks can control how a field value can change at run time in the process, action hooks are associated with the events that affect the state of a record and can perform specific tasks (e.g. send email) when the action is complete, record scripts are specific to a record type and can perform specific tasks at run time, and global scripts are to define libraries of routines that all record types in the schema can share. Hooks can be written in VBScript (for Windows) and Perl (for the UNIX system and Windows) using CQ API if required. So with the help hooks it can be controlled which role can take what action to change the state of a record to another and what field value can or should be modified by which roles under what state.

A CQ schema could comprise more than one record type, which would work as a template for a change request, to represent more different change requests, where state-based or stateless records could be created as required. CQ provides the predefined package (schema) as the template for the schema developer so that the required schema

can be developed effectively and efficiently by customizing and applying the packages, so in order to give additional functionality to a CQ schema, the specified package, which includes a set of schema components such as record types, forms, fields, hook code, queries, charts and reports, can be added to it, and then a new schema with required functionality will be created. So packages would be helpful to develop an efficient and flexible schema to realize the processes.

The work products, which can be adopted in test management and defect management for small projects, will be regarded as different types of change requests in CQ such as test strategy, master test plan, detailed test plans (e.g. static test plan, unit test plan, integration test plan, system test plan, acceptance test plan and so on), test environment, test execution plan, test specification including test scenario, test matrices, test case, and test script, test results, test reports and defect management documentation. Some of these work products can also be represented with the same record type. All these work products can be template-based with the help of the schema.

So based on the above discussed principle for developing schema, each corresponding process model for each work product can be designed and then developed in a CQ schema. But to accelerate start up of the test project, a schema as a template for small test projects should be provided here. Another important point is required to describe how this specified schema can work properly in accordance with the defined processes, where the requirements on them should be satisfied.

So in order to realize the automation of test management and defect management for small projects, the needed schema would be created by applying CQTM package and Common package, which can function and also be customized as templates for specific process models, at the same time in order to improve the communication and traceability degree, some useful packages can be applied to the schema. (The Enterprise schema that consists of CQTM and Common packages could also be applied directly, but it also consists of many superfluous record types, fields and forms for this scenario so that unnecessary resources will be taken during the operation, e.g. for user databases). How to make advantage of the finally created schema will be discussed in the following:

Test management and Defect management
CQTM schema (in detail in Section 4.1) is suited to automate test management for small projects; as discussed detailed in Section 4.1, CQTM does not support CMMI and the state model (Table 4.1 and Figure 6.1) of state-based test plan, test case, configured test case, and test suite is not so complex that the project can be executed quickly, where only test plan need be reviewed, as it is relatively more complex and important than other objects, which will be derived directly or indirectly from test plan. The earlier described CQTM three-phase usage model (planning, authoring and execution) can function as the tasks of the workflow in test management in comparison with those for large projects.

So the test manager would create the necessary asset registry records to represent release or product and define the testing scope so that the test planning hierarchy is set up, and then test plan can be defined. The tester would create test case, configuration, configured test case, and test suite records. For each configured test case test scripts are developed by testers using testing tools such as RPT, RMT, and RFT. The configured

test case and test suite records are executed and the test results are reviewed and committed to the database.

In addition to state-based record types such as test plan, test case, configured test case, and test suite, the other stateless record types are asset registry, configuration, configuration attribute, configuration value, iteration, test log, suite log, test type, computer, computer group and file location, which record important, necessary information about the process and can be associated with state-based objects. The relationship between these record types that is shown in Figure 4.1 would be implemented in CQTM. In order to represent its necessary information and attributes (see Section 4.1), each record type should have the appropriate form and fields to record them. Each record type's form and fields can be defined and customized manually based on its properties. Normally each record type's form will have one or more tabs or pages to group related sets of fields and controls, e.g. main tab would record the necessary attributes and references with other record types e.g. ID, state, headline, owner, priority, description, each of which should be of the appropriate data type such as Int, (Short-, Multiline-)String, Date_Time, Reference(_List) and so on.

The test plan record type's Main tab is displayed in Figure 6.2, in addition to above mentioned attributes, a file (e.g. with use cases) can be associated with a test plan as the test motivator file, and which iteration and asset registry can also selected with a test plan. Its Test Plans / Test Cases tab provides the hierarchy of test planning assets related to this test plan record, e.g. associated with its parent test plan, child plans and test cases, so the test plan record types can be of parent/child relationship with each other (Figure 6.3), so a test plan record type that works as test strategy or master test plan can have several child test plan records to represent detailed test plans such as unit test plan, integration test plan, system test plan and acceptance test plan, each of which would be associated with its (configured) test cases records.

The test case and configured test case record types can realize the test specification work product, Test case form's Main tab records the basic information such as ID, state, headline, owner, priority, description, associated test plan and iteration, then its Execution tab will associate a test script from a file location and executable configured test cases, which is associated with a configuration so that it can be executed.

A test suite record that orders the associated configured test cases records can be used to realize the test execution plan discussed in the process. The test results of configured test cases or test suit can be stored in a test log record that shows which configured test case or test suite has failed or successful under what configuration, and any defect found in test results can be submitted or associated with existing defects from test log record form in accordance with defect management.

The other record types in CQTM will have the similar way to record the own information and might have more tabs based on its properties, their forms are referenced in the Appendix (Legacy Data tab, which many record types have, is predefined and designed for migration from Rational Test Manager assets to Rational CQTM and so will not be here used in the VT.)

For defect management the Common package can be applied and its corresponding state model shown in Figure 4.2 is also not so complex and suited enough for small projects.

The main tab of defect record can be used for defect management documentation (see Figure 6.4), which consists of the basic necessary information (ID, state, headline, owner, priority, severity, description, key words) about each defect.

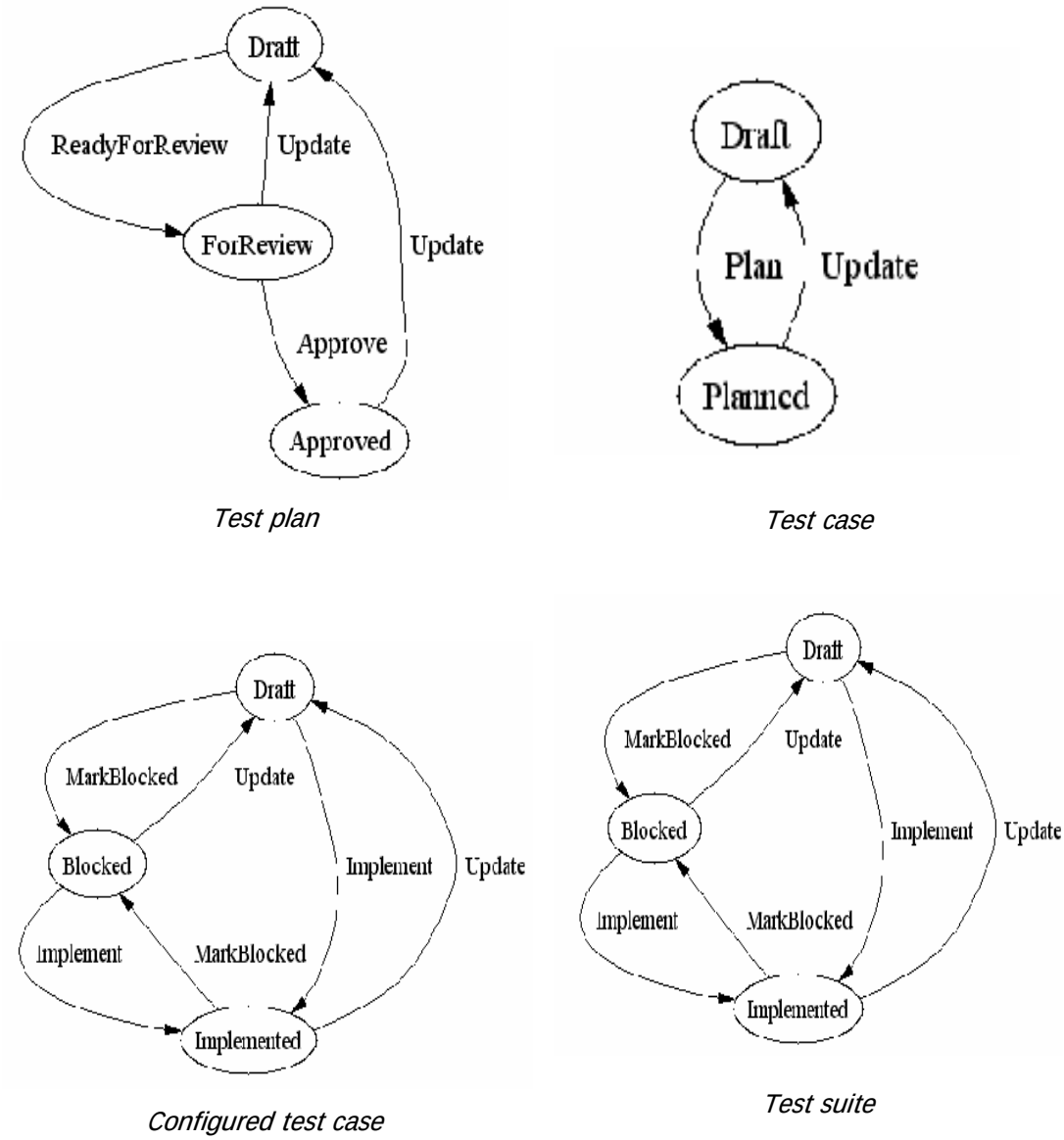


Figure 6.1 CQTM – state models of state-based objects

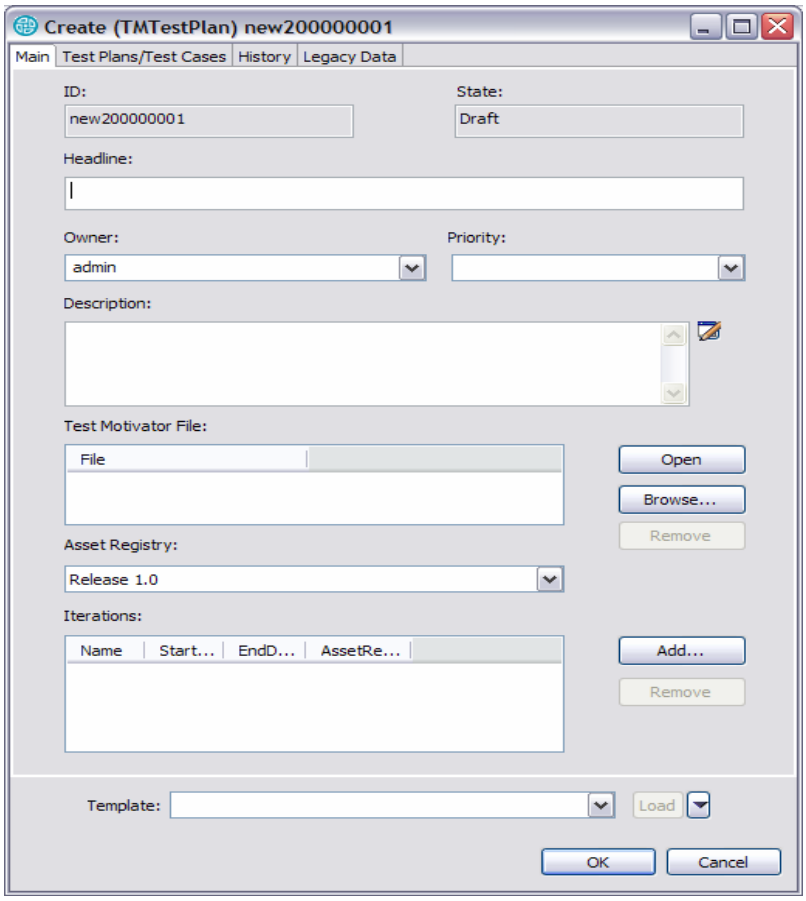


Figure 6.2 CQTM Schema – Main tab of Test Plan

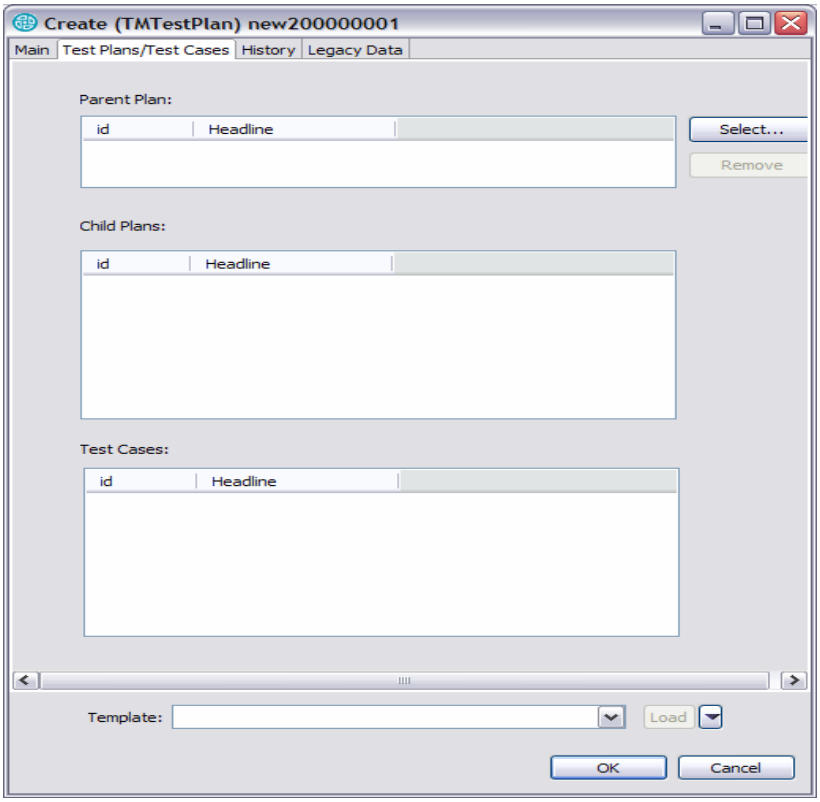


Figure 6.3 CQTM Schema – Test Plans/Test Cases tab of Test Plan

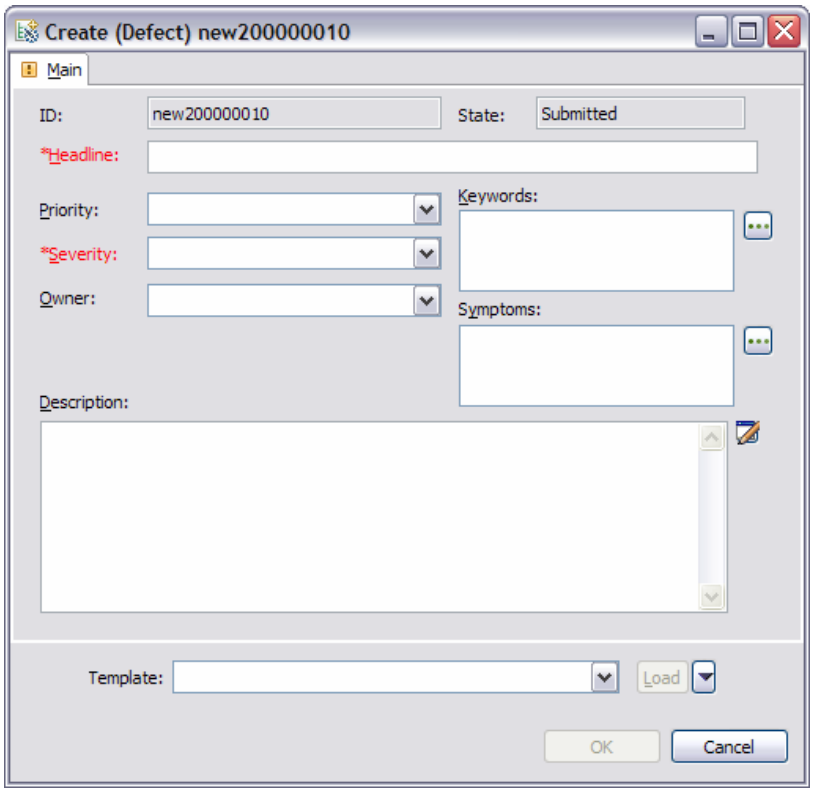


Figure 6.4 Common Schema – Main tab of Defect

Many other service components for test management and defect management can be realized by applying the CQ packages to the above created schema. Customer package is used to add a new tab named Customer to the existing record types such as defect, test plan, test case record) in order to record information about the customer who required the custom software, defined the test strategy and reported a defect. Resolution package can add a Resolution tab to defect record to record the information about the resolution for this defect. E-Mail package provides a good communication between the team members and make it possible that the corresponding responsibilities will get an E-mail when the information of a record was changed or created. Audit Trail package can provide an audit trail for any changes to a record, e.g. who, when, and what changes. Attachments package can make the test member attach an external file to a record. History package is able to record the history of a record, beginning with its submission and including all modifications to this record. Notes package is able to make the test member add any useful notes to a record. Any package will never destroy the existing form and fields of a record but add a new tab with its particular fields for providing new functionality to the form of the record. Not all of above mentioned packages are necessary to all types of small projects, but they do help when they are required in a specific small project.

Another important point is about the mastership of the record, which identifies the replica in the CQ Multisite environment which masters this record, in order to allow users to change the mastership of a record, a system field called ratl_mastership should be added to the form of the record type. In this scenario all the record types of the schema will have this field.

The finally created schema, which consists of all the above discussed record types of test management and defect management for small projects, can be associated with a user database, where all of records needed in the test and defect management will be collected, then from the CQ user database the queries can be defined by the filter expressions to search the database in order to select specific records. In Figure 6.5 the test status will be shown, where the numbers of test cases planned, implemented, scheduled and executed are calculated.

In this way reports for all types of record types can be got from the query result set and the specific report format. In addition with CQ a graphical view of query results can be made by charts, which can be distribution, trend or aging. Then the actual test project status is able to be got through reports and charts and be used for further analysis

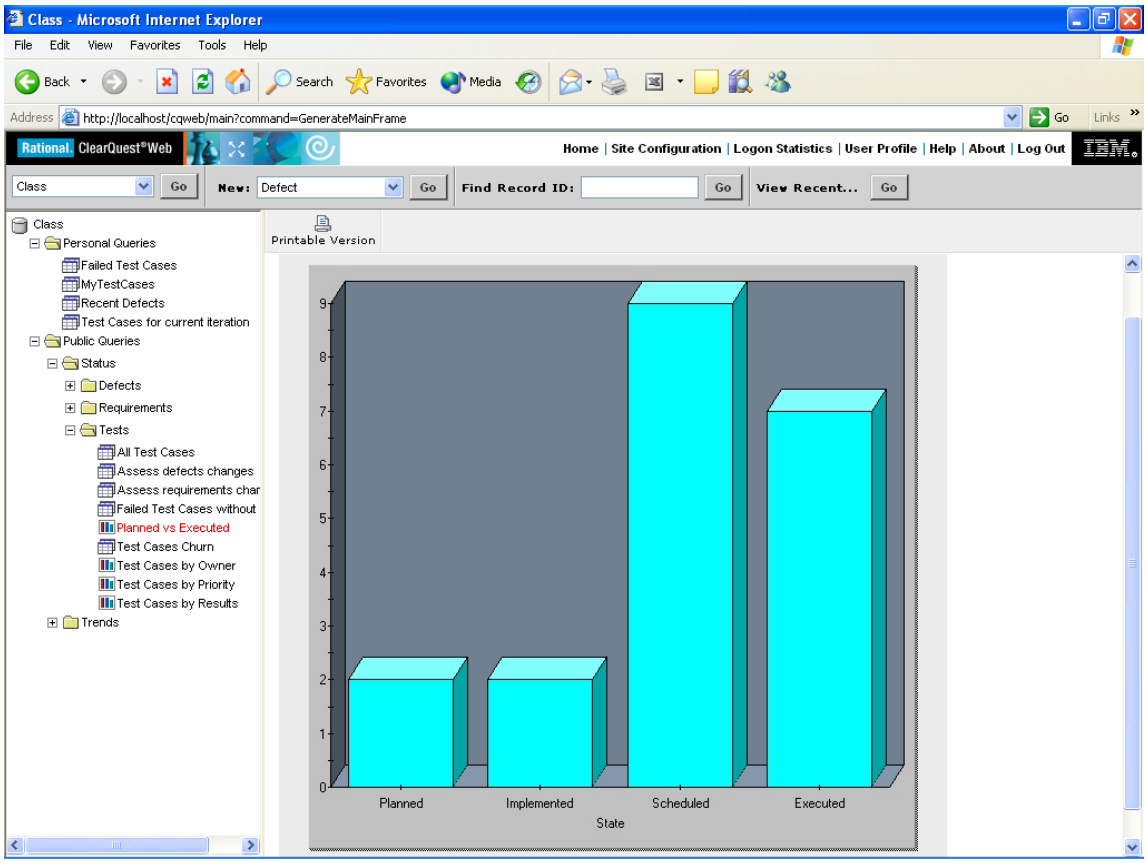


Figure 6.5 Test reporting and metrics

Critical business process testing

In this process tasks have been described in the last chapter, which can be automated with the help of above mentioned test management and defect management. The test motivator file of test plan record can be associated with the specific file with the customer's requirements on critical business process. Test strategy, integrated and acceptance test plans, test cases and results, which are needed in the critical business process testing, would be created and maintained in the test management. Any defect found in this process will be delivered in the defect management. Reports about the detailed information are created to show the status of the critical business process testing so that it can be determined if the custom software can be moved into the customer's production environment.

Training process
The above discussed processes such as test management, defect management and critical business process testing will be trained in this service. Tools such as CQ, RFT, RMT, RPT etc. and the used CQ schema and other newly defined or customized templates for Virtual Testcenter will be included in the training process. This process is flexible for small projects, where normally basic mind-set about testing processes for the small test project will be defined and trained for project members. The required work products such as training program charter, training plan, training evaluation and report etc. can be created based on the document templates e.g. using Microsoft Word order Excel, web sites. All the training documentation and materials for a specific test project will be provided.

Hard- and software infrastructure for small projects
In order to realize all these requirements on VT the hard- and software infrastructure should be set up to provide the technical environment, where in particular the general requirements on VT will be satisfied.

In Figure 6.6 the infrastructure for small projects is shown, the final schema, which was created earlier and supports test management and defect management, is stored in the schema repository (a special type of database); the user database associated with this schema resides on the CQ DB server, where DB2, Oracle, SQL Server and MS Access are supported.

On the CQ server the CQ application is running and communicates with other servers such as mail server, which enables the test project members to communicate with each other through E-mails and supports the Simple Mail Transfer Protocol (SMTP) or the Messaging Application Programming Interface (MAPI), license server, where the CQ licenses or CQ Multisite licenses are managed by license management software (e.g. FLEXlm), and the number of licenses will be calculated by determining how many project members will access the CQ databases (for CQ licenses) or the replicated CQ databases (for CQ Multisite licenses). Normally for small projects the number of licenses will not be very high e.g. 5-30 licenses.

The test server enables the remote test execution required by the test management (when performing the configured test case or test suite records) and is identified by the Computer and Computer group record types in the CQTM schema. Many Rational testing tool such as RFT, RPT, RMT and also the Eclipse Test and Performance Tools Platform (TPTP) test tools such as Junit, Manual test and URL test can run on the test server and their licenses will be managed on the license server. Test server can also be identical with the CQ server, and then tests will be executed locally.

As described in the processes for small projects, different roles with different responsibilities are required to perform the own tasks defined in the test project. So test manager, project manager, tester, and customer and their privileges can be defined using the CQ administration client, at the same time they would also perform the corresponding privileges based on the rules defined in the schema during the test project, as can be got by using CQ windows clients, which will have full functionality not like CQ Web client.

The CQ Web server is served by the CQ server, which provides a platform to execute the transactions the user wants, and then the user can access the CQ Web server remotely by using a web browser to submit, modify, and track change requests in the test management and defect management, and analyze the test project progress by creating queries and reports. It is very helpful and convenient for the project members to use CQ Web client if they work distributed geographically, although the CQ Web client has the limited functions e.g. it cannot execute the configured test case records and modify the schema.

All above discussed points will also function in a replica at anbother site, e.g. in Figure 6.6 Site A for IBM or some project members, who work at the same site, but there are also the customer or other project members, who work distributed geographically at another site B, which would have the similar infrastructure as Site A, so in order to make it possible that both sites have the same test project data and full functionality timely, so CQ Multisite will be adopted, in order that the database set (the schema and its associated user database) in the production environment is replicated on both sites synchronously, and then CQ shipping server is required to work as a synchronization server and create and transport the packets between the two sites. In many cases the firewall is set up for the most secure implementation, so the gateway shipping server will be configured with a receipt handler to pass packets from CQ shipping server across firewalls to similarly configured gateway shipping server at another site, which passes packets to CQ shipping server for updating the database set including schema repository and CQ DB server.

The update of the database set can happen at any site, but only such a site, which owns the mastership of the to be changed records in the user database, can do it. Mastership changes are communicated among replicas at different sites by the above described standard synchronization mechanism.

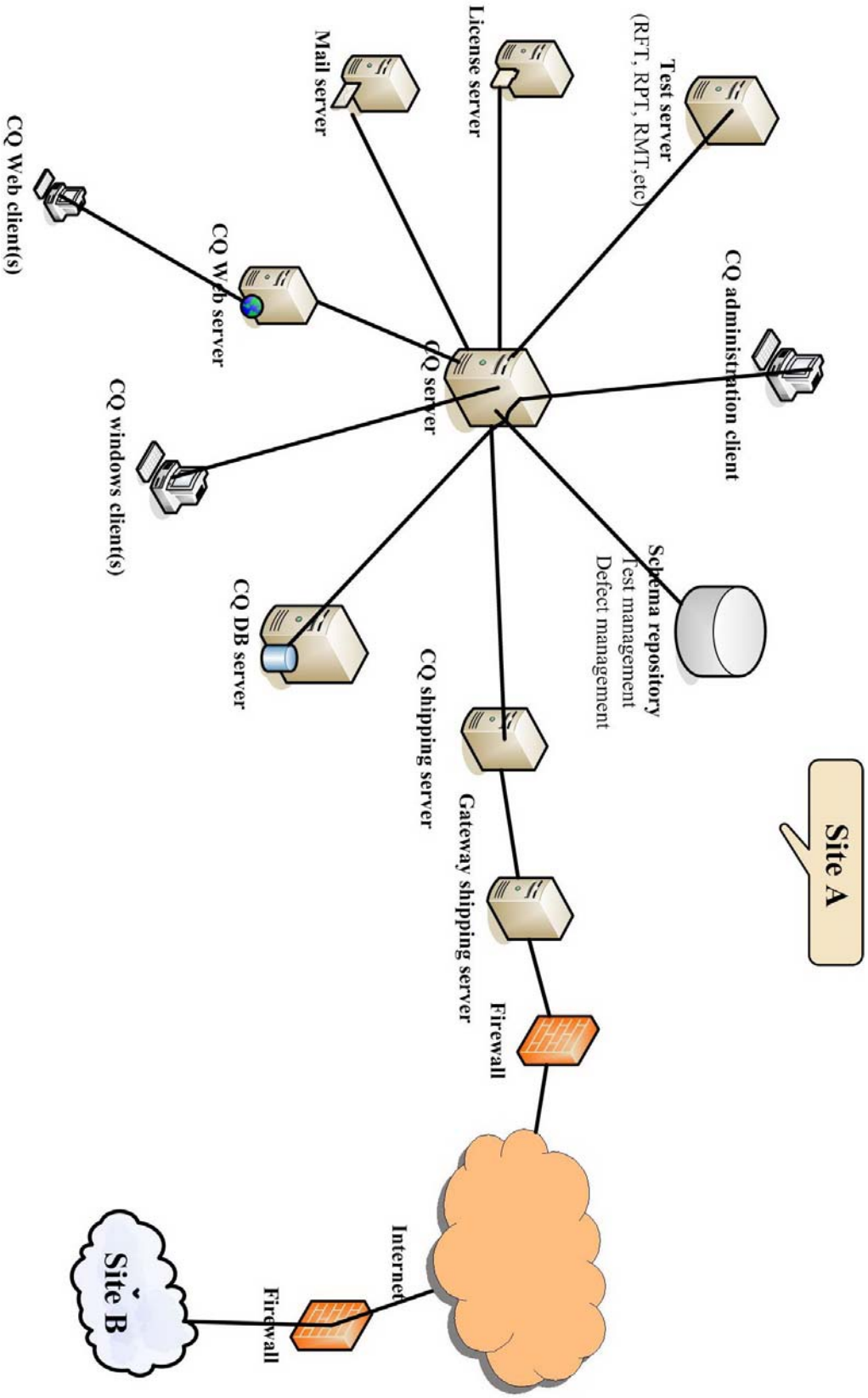


Figure 6.6 Hardware- and software infrastructure for small projects in VT

So with the CQ Multisite a good backup strategy can be got e.g. IBM and customer would have the test project data backed up at the site of each other. At the same time the schema in the schema repository can be exported into a text file, which can be imported into the schema repository of another system later; the data of CQ DB can also exported into a group of text files, which can be imported into the CQ DB of another system, so that the test project data can be reused effectively and efficiently.

All or parts of the components (servers) at one site can run on the same host in the project praxis, in particular in this scenario for small projects, as the data traffic is not very high. The appropriate type of operating system (e.g. Linux, UNIX, and Windows) and hardware parameters (e.g. CPU, memory, disk, network) can be decided based on the concrete test project situation.

6.2.2 Scenario for large projects

In this scenario it will be elaborated how the specific processes defined in Section 5.3.2 for large projects are realized with Rational Tools, and then the hard- and software infrastructure is constructed to provide the technical environment to realize the requirements on VT for large projects.

Realization of process models for large projects

The process and service components for large projects have been defined in section 5.3.2 such as Test management, Work product inspection, Defect management, Issue management, Change management, Configuration management, Requirements management, Critical business process testing procedure and Training process.

In Section 6.2.1 it has been analyzed how the processes can be automated with ClearQuest, these methods and principles function also for processes in this scenario, in particular the template concept would be very useful to realize these processes. But in comparison with the scenario for small projects there are many differences in this scenario, as the processes are much more complex than for small projects: every process guide is stringent and much more complex work products are to be created during the execution of the tasks in the process. In the following it will be described sequently: test management, work product inspection, defect management, issue management and change management are automated with Rational ClearQuest, configuration management with Rational ClearCase and requirements management with Rational RequisitePro, so in Figure 6.7 ClearQuest would function as the hub to link to RequsitePro, ClearCase and Rational Test Tools to realize this scenario, detailed information will be described later. At same time reporting and metrics about these work products can be created through ClearQuest just in the same way as stated in Figure 6.5.

Based on the process description there can be many possibilities of the state transition models, if these models can satisfy the workflow of the process, so the state transition models in the OPAL Schema (see Section 4.2) can be customized here for large projects, the lifecycle of each work product is complex enough to satisfy the workflow of the process, but the roles (e.g. customer role) should be defined clearly according to the process description to have appropriate privileges to take the action and change the correspond fields of the work product form.

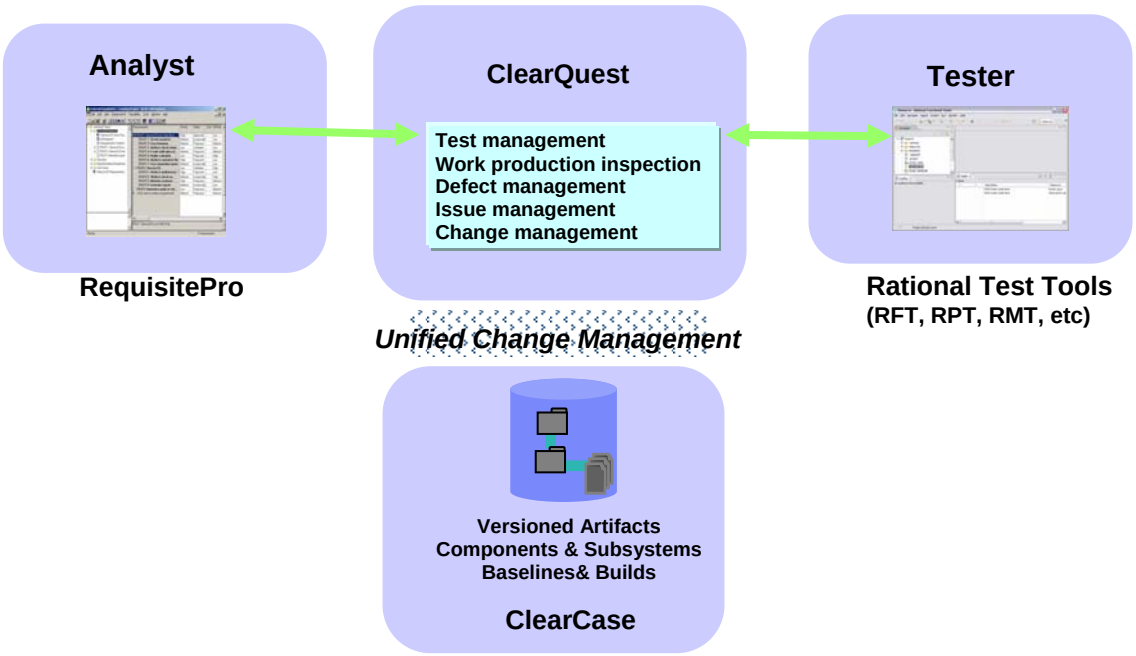


Figure 6.7 ClearQuest as the HUB in the scenario for large projects

Test management

In principle the structure of CQTM can be used as the template here, but the form and the state transition model of each record type should be customized more complex to record much more detailed information for large projects and satisfy the requirements on the process, so the state-based record types such as test plan, test case, configured test case, test suite and test report, and the stateless record types such as asset registry, configuration, configuration attribute, configuration value, iteration, test log, suite log, test type, computer, computer group and file location must be tailored and improved for large projects, so that these record types can represent all the work products needed in the process such as test strategy, master test plan, detailed test plans (e.g. static test plan, unit test plan, integration test plan, system test plan, acceptance test plan and so on), test environment, test execution plan, test specification (including test scenario, test matrices, test case, and test script), test results and test measurements.

In order to take advantage of templates, the state transition models discussed in the OPAL CQTM Schema (in Section 4.2) can be used for these record types, as they are compliant with OPAL, so the process quality can be ensured. At the same time the form of each record type will be customized to have more tabs to record more information to represent its relevant work product for large projects, in the following it will be discussed how these work products are realized with these record types:

Test plan record type can represent test strategy, master test plan, detailed test plans (e.g. static test plan, unit test plan, integration test plan, system test plan, acceptance test plan and so on) and test environment, so a field should be added to its Main tab to specify the plan type, which has the basic necessary information as described in Section 6.2.1, such as ID, date, state, priority, test manager, project manager, application, headline, description and purpose. The Background tab can provide brief overview of the project or release, current system environment with a short description of the current technical and operational environment, project goals, release objectives and any other references.

In the Strategy tab business functions (use cases from use case model), structural functions (non-functional requirements), risk assessment, contingency plans and test focus areas (based on the quality goals) can be recorded. The Associated Matrices tab consists of test matrices, which specify functional and non-functional tests needed in the test project for a specific test software, and organizational responsibilities, which describe who (roles) will be responsible for test planning, detailed test plans, test preparation, test execution and test results reporting.

The Planned Activities tab will give an overview about test objectives which can be for varying test levels, test scope which defines the boundaries of testing, assumptions, approach which could reference to metrics to be used in the test project or requirements on test management and reporting procedures, and functions to be tested and functions not to be tested because of its priority, test effort and test ability. In the Criteria tab the entry, exit, suspension and resumption criteria will be defined for each of the detailed test levels within the project scope.

The test environment associated with different types of test plans (test levels) can be described through Test Environment tab, where the test environment build strategy and requirements are defined to ensure that the necessary hardware, software, tools, and other infrastructure components are able to be assembled to create the test environment, test data strategy is defined to provide an approach for test data creation, setup, and maintenance, test tools build is to define tools specifications, evaluation, selection, acquisition, and implementation planning.

The Schedule tab will specify test planning workshops, major test milestones, resource requirements, personnel responsible for test activities and iterations. Related Docs tab will specify its parent/child plans and the test cases created from this test plan.

When creating a specific type of test plan such as test strategy, master test plan, detailed test plans (e.g. static test plan, unit test plan, integration test plan, system test plan, acceptance test plan and so on), a set of relevant tabs and fields can be selected and filled.

Then test case and configured test case record types will be used to represent the test specification. The Main tab of the test case record type will record the basic necessary information such as ID, date, state, priority, project name, reviewer, application, headline, description, the associated test plan and test level. The Preparation tab will record the estimated test effort, tools description, test case dependencies, test data preparation and special procedural requirements for a specific test case. The Execution tab will reference to the test script, its associated configured test case records, which can be executed, and the associated iteration. In the configured test case form the Main tab consists of the configuration and its associated test case in addition to other normal necessary information. The Execution tab will reference to the test script, which can be copied directly from its associated test case. After the configured test case is executed, the test results will be recorded in the test log, where its Main tab will record which test script from which configured test case failed or passed and its Configuration tab records which configuration, iteration or build was associated with this test, if any defect has been found, it can be submitted as a new defect or associated with an existing defect through the Generated Defects tab.

The test execution plan can be represented by a test suite record that orders the associated configured test cases records so that the test cases will be executed in a specific order.

In the end of testing normally a test report will be created to show the test status and can be created in the way like in Figure 6.5, but for large projects the test report record type is customized specially, its Overview tab records the basic necessary information such as ID, date, state, project name, test manager, project manager, application, brief description and testing overview. The Test Result Summary will have a test case report to show the number of planned, executed, successful and failed test cases and the number of defects found for each test level. Defects are also summarized to show the number of defects with different severities and different states such as open, pending and close. The Analysis and Conclusion tab will analyze the testing status and have a conclusion based on the test criteria in the test plan.

In addition Requirements tab makes it possible that a newly created requirement or the existing relevant requirements are associated with the test plan, test case, configured test case record , see Requirements management for detailed information.

The other record types will function as those in the scenario for small projects. All these forms can be changed and tailored based on the concrete requirements of the project praxis. As all the work products needed in the process can be realized with these record types, test management can be performed based on the process guideline.

Work product inspection

The work production inspection (WPI) form is the most important work product during this process to record all the necessary information and so will be realized as a CQ record type, which would have many tabs to group and classify the related information. The Overview tab of this form will record project information (e.g. ID, status, date, project name, project manager, application name and version, submitter, status), work product information (e.g. description, version, type such as requirements, design, code, test plan, test case and so on, reference documentations used during the review) and the inspection phase of the development process (e.g. requirements, design, code, test, deploy, maintain). Meeting tab will show the meeting type is formal or informal, when and where the meeting will take place and the list of the primary and additional participants and their responsibilities. Defects tab will record any defect identified during the work product inspection, which can be submitted through the defect management. Issue tab will record any issue identified during the work product inspection, which can be submitted through the issue management. Notes tab will provide identified risks that the defects can lead to, the suggestions for added value and meeting minutes. Resolution tab will give an exit decision of this WPI such as accept, reject or reinspect, the history of WPI changes is also saved for the late analysis. The metrics for this WPI should also be recorded such as work product size, meeting preparation effort, rework effort and number of the defects identified with different severities. With a WPI form a work product inspection record can be created for the to be reviewed work product and project members can perform WPI with the process guide.

Defect management

The defect management documentation used during the whole process should be realized through a defect record type, which will have more tabs to record more necessary information in comparison to the one for small projects.

The Overview tab records the basic necessary information such as ID, date, project name, application name and version, name of the project organizational unit, headline, submitter who raises this defect, originator who actually identified this defect, status, project manager, severity, resolution owner, priority level, date targeted for resolution of defect and description.

The Problem detail tab provides detailed description of the defect, the type of the defect will be identified. During which activity (e.g. inspection, unit test, integration test, system, test, acceptance test, etc.) of the software test the defect was found. In which phase (e.g. requirements, design, code, test, deploy and maintain) of the software development lifecycle the defect has entered or was detected. The root cause of the defect is analyzed to identify the reasons why defect has happened during a process or in a work product. The process during which the defect was injected or detected is also to be identified. The technique (e.g. inspection, component, system) from which the defect escaped or was detected can be mentioned here. Normally the defect need be able to be reproduced so that the resolution owner can analyze it effectively to find a good resolution, so detailed steps and information necessary should be given to reproduce the defect. Any suggestion or additional comments related to the defect can also be written in this tab.

The Supporting Documentation tab makes it possible to add any related attachment which is useful to describe or resolve the defect. The Analysis tab would analyze the impact of the defect and recommend the planned action to fix it. The date, duration and effort will be planned to fix the defect. The planned actions including skill and resources required to implement the fix, will be listed to address the defect. Any comment related to actions and affected areas (e.g. work products, codes) can be recorded during the lifecycle of the defect.

The Resolution tab will have necessary information about the resolution of the defect. The resolution code can be set as e.g. fixed, change, issue, invalid, fixed indirectly, fixed in the future release. The actual date, duration, effort, number of build cycles and test cycles for resolving the defect will be recorded. If the defect is a duplicate of another existing defect or a group of related existing defects, then link(s) can be referenced to associate the defect. The history of state transition during the defect lifecycle is also logged in this tab. The Test Logs tab can help the project members to check which test logs the defect is related to.

Unified Change Management tab associates this defect with its change set including a list of work products that are to be created or modified to resolve this defect in a UCM project, see Configuration management for detailed information.

Requirements tab makes it possible that a newly created requirement or the existing relevant requirements are associated with the defect record, see Requirements management for detailed information.

With defect record type, any defect found during the test project can be managed based on the process guide.

Issue Management
The issue management documentation used during the whole process would be realized through an issue record type, which has several tabs to record all information.

The Overview tab will record the basic necessary information such as ID, date, project name, application name and version, name of the project organizational unit, headline, submitter who raises this issue, originator who actually identified this issue, status, project manager, severity, resolution owner, priority level, date targeted for resolution of issue and description. The phase (e.g. requirements, design, code, test, deploy and maintain) during which the issue was identified is selected. The root cause of the issue is analyzed to identify the reasons why issue has happened during a process or in a work product. Whether the issue is major or minor can be determined by its type. Whether the issue is recurrent would be recorded. Areas (e.g. work products, codes) affected by the issue is summarized. Actions to address the issue or any comment about the issue or its resolution are able to be added in this tab.

The Resolution tab would provide information about the resolution of the issue. The resolution code can be set as e.g. change, defect, transferred or resolved. The planned actions including skill and resources required to resolve the issue, will be listed. The actual date, effort and actions for resolving the defect can be recorded here. The history of state transition during the issue lifecycle is also logged in this tab.

Unified Change Management tab associates this issue with its change set including a list of work products that are to be created or modified to resolve this issue in a UCM project, see Configuration management for detailed information.

With issue record type, any issue found during the test project can be managed based on the process guide.

Change management
The change management documentation used during the whole process would be realized through a change request record type, which also has several tabs to record all information.

Overview tab will record the basic necessary information such as ID, date, project name, application name and version, name of the project organizational unit, headline, submitter who raises this issue, originator who actually identified this issue, status, project manager, severity, resolution owner, priority level, date targeted for resolution of issue and description. Its size determines whether the change is large or small. The first affected phase (e.g. solution definition, design, solution generation, solution validation, solution deployment, training or other) is selected. A high-level estimate of cost and effort to estimate the change request will be shown in this tab.

Supporting Documentation tab makes it possible to add any related attachment which is useful to describe or resolve the change request. In Analysis tab the estimate of start-date, finish date, cost and effort to implement the change request is performed. Areas (e.g. business functions, work products or codes) which are affected by the change request will be identified. Alternative solutions considered to meet the change request are

documented, where the selected one should be clearly identified. Any other comment related to actions about the change request and its resolution can be described as well.

Implementation tab would determine the approved start date, finish date, cost and effort to implement the change request. Specific criteria that detail the requirements for the completion of the change request are defined. If the change request is so large that the original schedule is impacted heavily and must be changed, the new schedule delivery date will be made.

In Resolution tab the information about the resolution of the change request is provided. The resolution code can be set as e.g. defect, issue, complete, invalid, and implemented in next release. If the change request is a duplicate of another existing change request or a group of related existing change requests, then link(s) can be referenced to associate the change request. If the change request has resulted in the change of some plans and agreements amendment, they will be recorded and their delivery date is updated. The actual start date, finish date, effort and actions for resolving the defect can be recorded here.

Unified Change Management tab associates this change request with its change set including a list of work products that are to be created or modified to implement the change request in a UCM project, see Configuration management for detailed information.

Requirements tab makes it possible that a new requirement is created or the existing relevant requirements are modified from a change request; see Requirements management for detailed information.

The history of state transition during the change request lifecycle is also logged. A change request record can be created to manage any change required during the project in accordance with the process guideline.

Configuration management

In the test project the integrity of all the work products, which are subject to change, is established and maintained under the configuration management, so that any version of a work product can be recreated at any time to define and organize the elements of a system for testing software.

Test scripts, source code, system/component requirements recorded in the documents, builds, all work products needed in the test environment, even the text file exported from a CQ schema and other work products which need to be tested or required during the software test will be managed in the configuration management system. Virtual Testcenter would enable that the test strategy, test plans, test cases, test execution plan would be managed in test management and can be versioned by associating an iteration record, as makes it possible to get a specific versioned group of test strategy, test plans and test cases for testing a specific version of iteration.

The configuration management plan mentioned in the process description will be made based on the concrete project situation. It has been discussed that Rational ClearCase (CC) is an enterprise software configuration management tool suited for large projects, so the library management system will be realized by Rational ClearCase (CC), which

has defined concepts such as element, version, VOB, check out-edit-check in model and view to control configuration items.

CC supports two types of software configuration management: base ClearCase and Unified Change Management (UCM), so here particularly UCM will be selected for Virtual Testcenter, because the combination of CC and CQ used in UCM provides full integration of activity-based development with process management and problem tracking, so that the configuration management is able to cooperate very well with other processes such as defect management, change management, issue management, as each change to an element under configuration management could always originate from the test project's change management, defect management, issue management and requirements management.

With the help of UCM any activity such as task or requirement (e.g. defect fix, change implementation, issue resolution) from defect management documentation, change management documentation and issue management documentation should be able to be associated with the correspondent view that is represented as a directory for accessing to a specific version of one or more elements in a VOB and change set including a list of work products that are to be created or modified to complete this task. Which versions of work products are shown in the view is determined in a stream which maintains also many change sets and baselines. All of them are managed and maintained in a UCM project, which is mapped to the development structure of an application or system in the test project.

So in order to implement this process, the forms of defect, change and issue record types of the CQ schema will need a new tab named Unified Change Management by applying packages such as AMStatesTypes, BaseCMActivity, UCMPolicyScripts and UnifiedChangeManagement. The new created tab is shown in Figure 6.8. The defect, change or issue record will be referenced in CC as an activity, and then the responsible role (e.g. test manager, tester, developer) of test project team can create, modify and develop the work products (e.g. test scripts, soft code, builds, other documents) of change set in the work area defined by view and stream shown in the record to resolve the tasks (e.g. resolving defect, issue and change request) defined in the record stored in the CQ database.

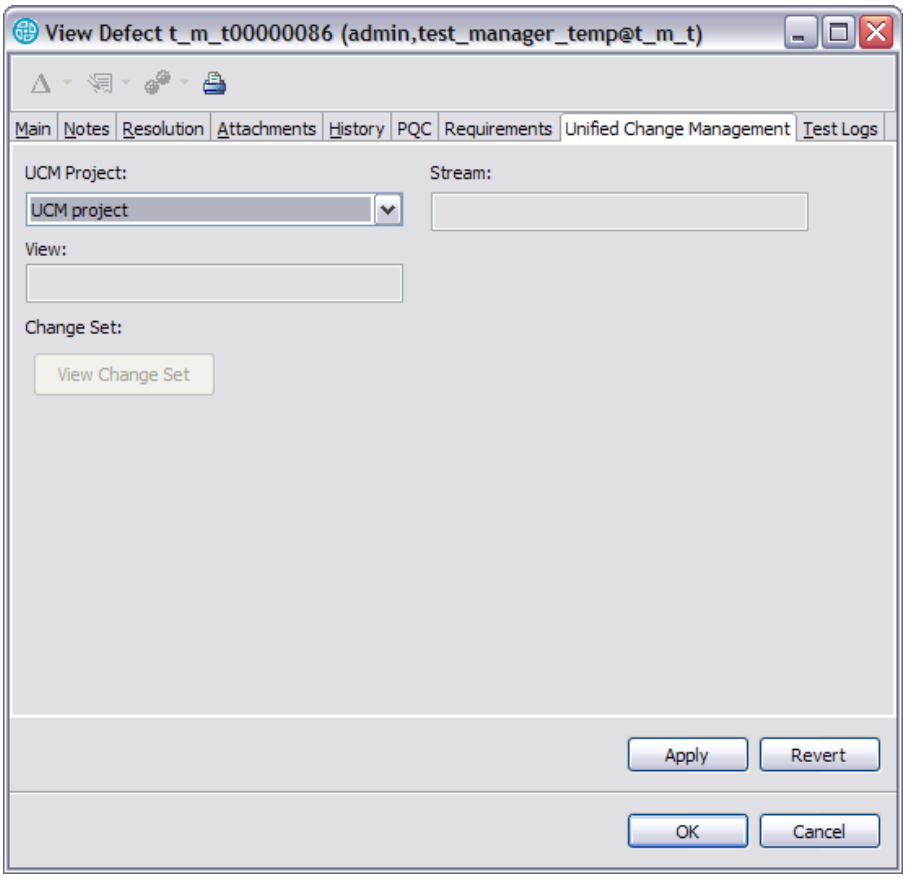


Figure 6.8 Unified Change Management tab

With CC a UCM project will be created to represent the whole structure of the application or system in the test project, where components including a group of file and directory elements (source code, relevant documents or standards) can be used for parts of this UCM project, and then streams would maintain a list of activities and baselines. In addition to above mentioned streams associated with specific defects, change requests and issues. There can be many streams for different tasks, such as test, development, build, deploy. In particular in the test project test stream would be created for test team to work on test scripts of e.g. RFT, RPT, RMT, work products needed in the test environment, applicable standards (see Section 3.2.8) used for reviewing work products and the baselines of to be tested work products (e.g. software components, integrated systems) for all levels of tests.

With UCM it makes it possible to test any version of the work products, which are created or modified for defects, issue, change requests or other requirements. UCM enables that these defects, issues, change requests and other requirements can be resolved more efficiently by associating each of them with its change set and tracking problems in order that the corresponding tests can pass finally.

Requirements management
In the test project requirements of the customer and project need be identified, tested, refined and traced, so requirements management need cooperate efficiently with other processes such as change management, defect management and test management.

As has been discussed that Rational RequisitePro (RP) is a requirements management tool, where a project template can be used, so in Virtual Testcenter a new project template will be defined for large test projects and in this RP project template the four document types with the appropriate format, such as business requirement specification, system requirement specification, component requirement specification and use case specification, and four requirements types with the appropriate format, such as business requirement, system requirement, component requirement and use case, will be defined. For each requirement type its corresponding requirement attributes will also be defined such as priority, status, acceptance criteria, applicable test type(s), release, notes and so on, each of which would be of the appropriate data type. A RP requirement can be created using the newly defined template (see Figure 6.9) for Virtual Testcenter, where requirement type, name, text description can be determined and modified, in addition to revision information (date, author, change description), requirement attributes, traceability with other requirements, hierarchy (parent/child relationship) with other requirements and other discussion information about this requirement.

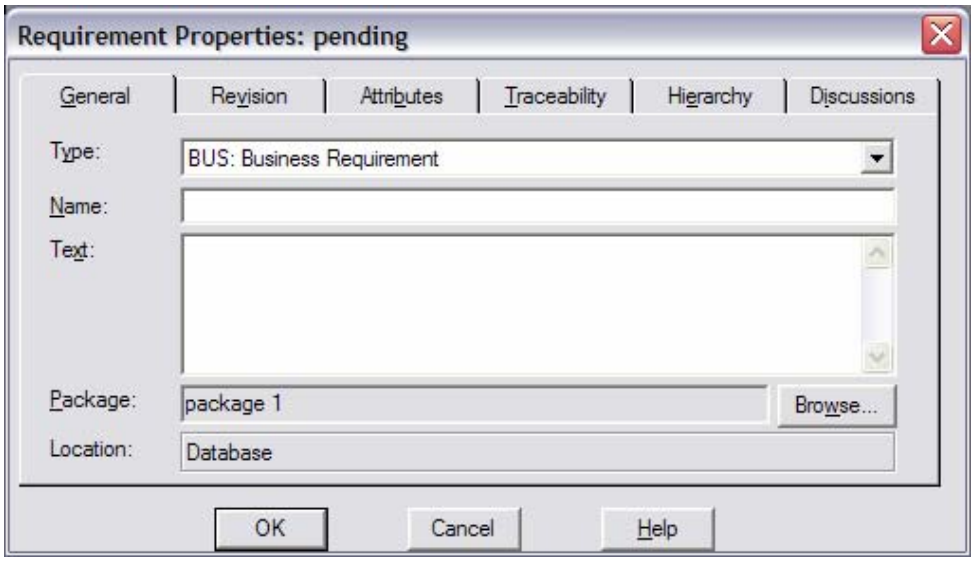


Figure 6.9 RP requirement

Normally during the test project in Virtual Testcenter unit tests will be based on component requirements, integration tests will be based on component or system requirements, system tests including functional and non-functional requirements will be based on system requirements, where functional tests can also be described specially based on use cases, acceptance tests will be based on business or system requirements. In principle component requirements and use cases will be derived from system requirements, which again result from business requirements.

During this process a requirement document of a specified type can at first be created using Microsoft Word to describe requirements, and then in the document requirements of the appropriate type can be created, these requirements will be stored in the test project database. RP views can be created to display and manage requirements, their attributes, and their relationships with other requirements. So the requirements traceability and verification matrix mentioned in the process description will be created by a specified view, which records the overall relationship and the chain traceability of all requirements in the test project, where the business requirements will always stand on the

highest level. From it all requirements are traced and the verification situation of them is shown and updated (if all associated tests of a requirement have passed, its status will be updated as implemented.), then the test coverage and results of all these requirements in the project will be got, so it can be determined if the test project is finished and the custom software product has satisfied the customer's requirements with an arranged degree and can be delivered into the productive environment of the customer.

The requirements management should have a good cooperation with other processes, as e.g. a requirement can be submitted newly or modified from a change request, a defect would be associated with a requirement to determine the status of the requirement or specify the expected results, test plans or test cases will be associated with the requirements to show when and how each requirement will be tested during the test project, so it is required that RP requirements should be able to be associated with CQ record types of the earlier created schema such as change, defect, test plan, test case and configured test case records.

In order to realize the association, a RP requirement attribute type named ClearQuest integration for the integration between a RP requirement type and another CQ record type would be used, so any RP requirement type, which would be associated with a CQ record type, should have a new attribute of ClearQuest integration type added, at the same time the CQ record type will have a new tab with the name Requirements (see Figure 6.10) added in its form by applying Rational RequisitePro package to the earlier created schema.

In order to configure the integration between RP project and CQ user database, a Rational Administrator Project (RAP) need be created by using IBM Rational Administrator, where the RP project and CQ user database will be specified and the association relationship should be defined rightly to enable that each of the above mentioned RP requirement types, such as business requirement, system requirement, component requirement and use case, can be associated with each of the CQ record types such as change, defect, test plan, test case, configured test case and CQ requirement records.

If a RP requirement is created, then a corresponding CQ requirement, which consists of a set of attributes (see Figure 6.11) such as name, requirement type, RAP, RP project and revision information to associate this RP requirement in the RP database, can be created in the CQ database through the RAP, so that every RP requirement will have a unique mapped CQ requirement (also called proxy requirement). From its Requirements tab a defect, change request, test plan, test case or configured test case record can be associated or disassociated with specific RP requirements, which could have already existed or be created newly using RP, or CQ requirements, which are associated with existing RP requirements.

With RP the baseline of all types of requirements for specific releases or important milestones can be made and then be managed under the configuration management, so that software tests can be performed based on the appropriate requirements.

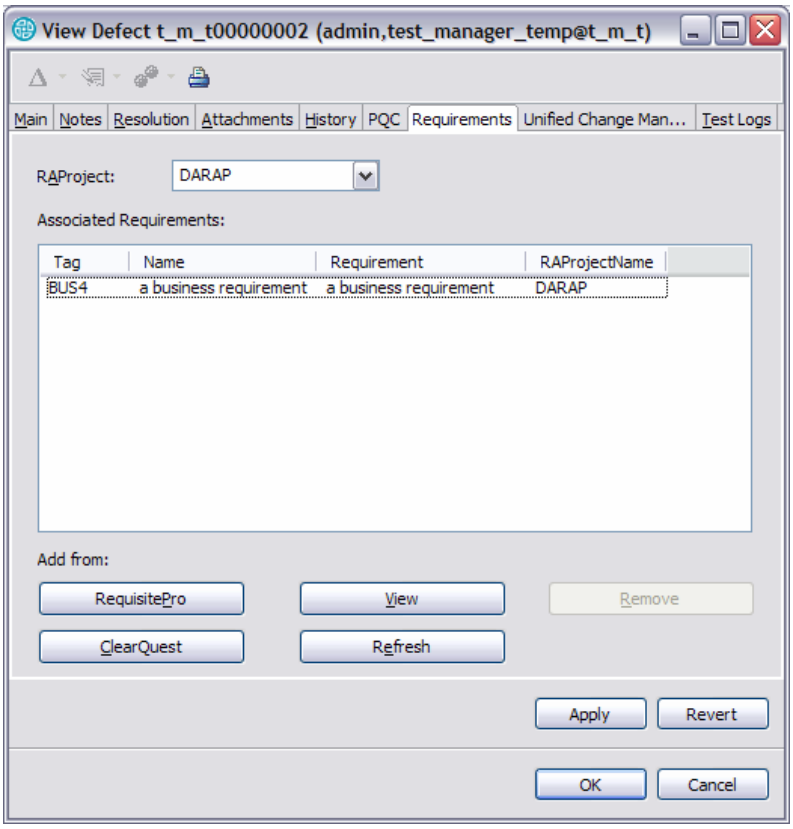


Figure 6.10 Requirements tab of customized CQ record type form

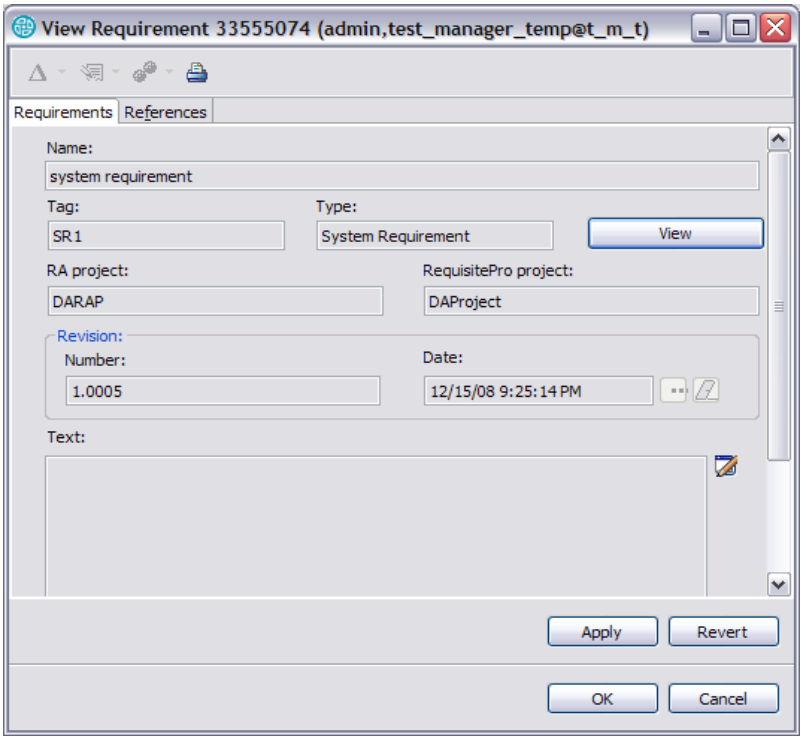


Figure 6.11 CQ requirementt

Critical business process testing procedure

In this process tasks have been described in the last chapter, which can be automated with the help of above mentioned requirements management, test management, defect

management and issue management. The customer's requirements on critical business process will be created and maintained in the requirements management, which are normally of business requirement type. Then test strategy, appropriate integrated and acceptance test plans and test cases will be selected and maintained by using the association with the requirements on critical business process through the integration between test management and requirements management. Any defect and issue found during this process will be delivered and dealt with in accordance with the defect management and issue management. Reports about the detailed information are created to show the status of the critical business process testing so that it can be determined if the custom software can be moved into the customer's production environment.

Training process
The above discussed processes such as test management, work product inspection, defect management, issue management, change management, configuration management, requirements management and critical business process testing procedure will be trained in this service. Tools such as CQ, CC, RP, RFT, RMT, RPT etc. and the used CQ schema and other newly defined or customized templates for Virtual Testcenter will be included in the training process. This process is relatively complex and could last long for large projects, where normally advanced mind-set about testing processes for the large test project will be defined and trained for project members. The required work products such as training program charter, training plan, training evaluation and report etc. can be created based on the document templates e.g. using Microsoft Word order Excel, web sites. All the training documentation and materials for a specific test project will be provided and controlled under the configuration management of this scenario.

Hard- and software infrastructure for large projects
In order to realize all these requirements on VT the hard- and software infrastructure should be set up to provide the technical environment, where in particular the general requirements on VT will be satisfied.

In Figure 6.10 the infrastructure for large projects is shown, where the part for CQ infrastructure functions similar to the infrastructure of the scenario for small projects (see Figure 6.6), the final CQ schema, which was created and customized earlier and supports test management, defect management, word product inspection, issue management and change management, is stored in the schema repository (a special type of database), other servers such as CQ server, CQ DB server, test server, CQ web server, license server and mail server will work just like described in Section 6.2.1, where the license server would manage the licenses of all Rational Tools used here such as CQ, CC, RP, RFT, RPT, RMT, etc. The number of licences for specific Rational Tools for large test projects should normally be high, e.g. more than 30.

The main difference to the scenario for small projects is that Rational ClearCase and RequisitePro need be set up to provide configuration management and requirements management during the large test project.

In the configuration management CC registry server would record the information of shared CC resources in the ClearCase environment of the test project, such as which VOB is served by which host, which views are available, where storage directories are, which regions are available. All the views will be created on the CC view server and could work as a temporary working area for the project members to resolve a defect,

change request or issue managed by the processes realized through CQ, the modifications in the views should be checked in timely if finished. All versions of data of a UCM project will be stored and managed in the CC VOB server, so it is very important to backup the VOB of the test project.

There are two possibilities for the project members to access to the streams and views of the UCM project, one is got through CC clients locally; the other is got by accessing to the CC web server remotely through CC web client (web browser) or CC remote client, where CC web server will provide the web interface to work on streams and views of the VOB in order to resolve a UCM activity.

In the requirements management on the RP server the RP application is running to enable that the project members are able to create a RP project from the above designed project template, and then test project members can work on RP requirements through the RP clients locally or through RP web clients (web browser) remotely by accessing to the RP web server. All the RP requirements will be stored on the RP database server.

It is very helpful and convenient for the test project members to use web clients for CQ, CC and RP if they work distributed geographically, as common functionalities can be got in this way, but at the same time full functionalities can be performed through their thick clients (locally) if required.

As can also be seen in the Figure 6.12 that ClearQuest would be integrated with ClearCase through a UCM project and also with RequsitePro through a RAP to provide the efficient communication mechanism for all of the processes required for the large test projects.

All above discussed points will function at one site e.g. Site A in Figure 6.12 for IBM or some project members, who work at the same site, but there are also the customers or other project members, who work distributed geographically at another site B, which would have the similar infrastructure as Site A, so in order to make it possible that both sites have the same test project data and full functionality timely, and then CQ Multisite and CC Multisite will be used here, in order that the database set (the CQ schema and its associated user database, VOB data) in the production environment is replicated on both sites synchronously.

But at the site B RP will not run, as RP has no Multisite version, but as designed earlier, every RP requirement will have a unique mapped CQ requirement record in the CQ user database, so that all the requirements information will be synchronized with the CQ Multisite.

The whole synchronization process would work similarly as the scenario for small projects; the main difference is that an additional CC shipping server would be used to manage packet transfers between CC VOB server and gateway shipping server.

With the CQ Multisite and CC Mulitisite a good backup strategy can be got e.g. IBM and customer would have the test project data backed up at the site of each other.

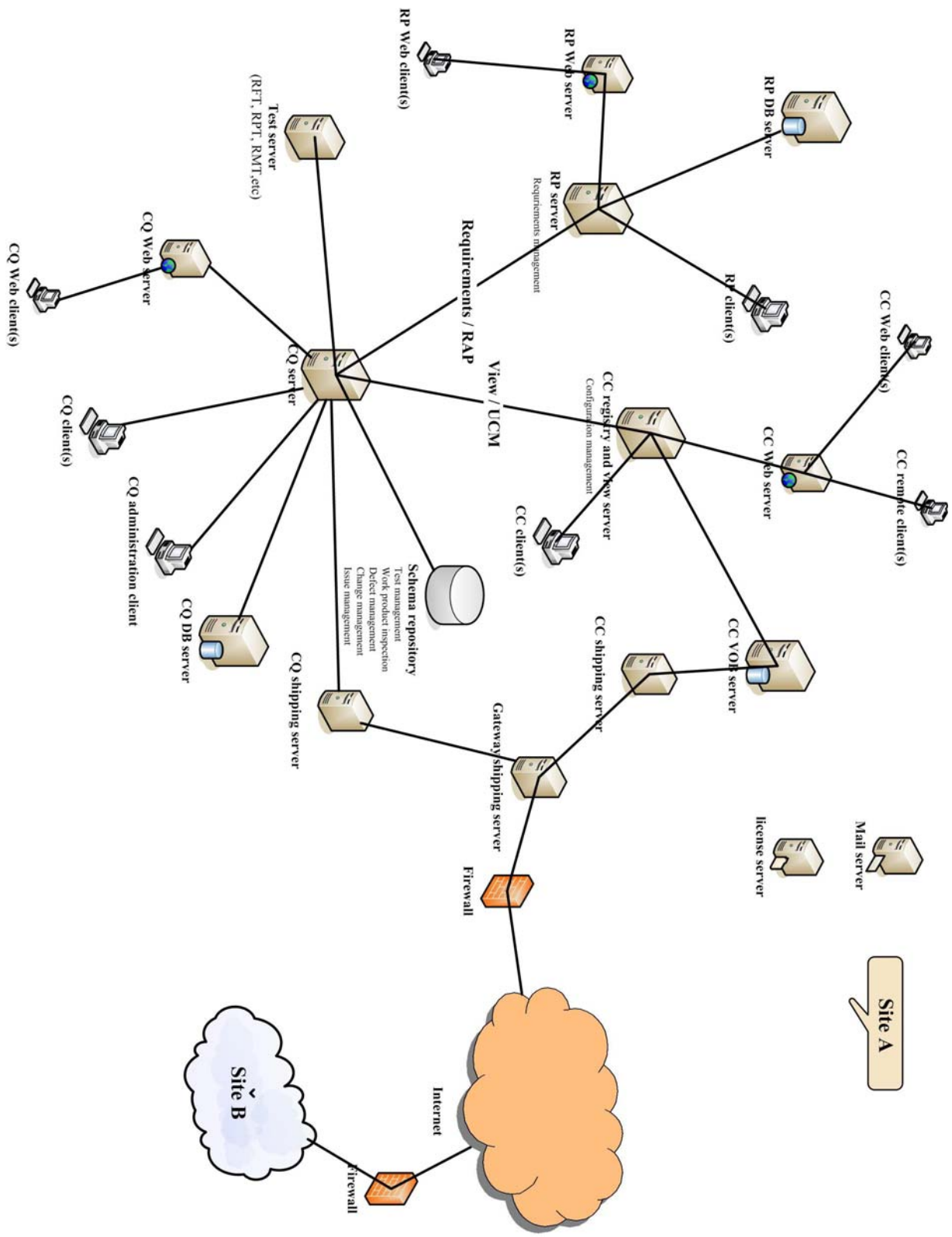


Figure 6.12 Hardware- and software infrastructure for large projects in VT

In this scenario for large test projects the data traffic should be high, so the host performance should be ensured, in particular for ClearCase environment, nearly all of CC operations involve access to data in the VOB, so the host performance of the CC VOB server should be ensured, where a high network high-bandwidth (e.g. 100 MB/second or greater) network connection to the VOB host should be used. CC registry and view server can run on a separate host and CC web server will also run on a standalone host in order to provide good performance for working on views of a UCM project for test project. CQ server, RP server and DB server (for CQ user database, CQ schema repository and RP database) will each reside on a separate host. CQ web server and RP web server can share one host, and then each host component can select the appropriate type of operating systems.

6.2.3 Summary of scenarios

The above two sections have discussed the scenarios separately for small and large test projects, in practice these two solutions would be provided in a infrastructure in Figure 6.13, the hardware- and software infrastructure shown in Figure 6.12 will be set up on IBM site to provide a central management for all test projects, where only ClearQuest environment will be used for small test projects. The CQ schema repository will store two earlier created schemas for small and large test projects, and then for each customer a separate CQ user database will be created associated with the appropriate CQ schema based on the test project type. A specific ClearCase environment (particularly for CC VOB and view servers) will be established for each customer, as each test project would have its specific products and materials to be managed under configuration management. In the RP environment all large test projects can create their specific RP projects from the earlier designed RP project template, and then a separate RP database will be created for each customer. At the sites for customers the appropriate infrastructure for small or large project will be selected based on their own test project type.

The services for web servers (CQ, CC or RP web server) can be provided mainly by IBM so that all the customers can access to their own test project data conveniently and remotely through web clients (web browser) or remote clients. In order to have full functionalities, all the test project members including IBM members and customers, who work distributed geographically, can use Virtual Private Network (VPN) to log on specific hosts (e.g. CQ server, RP server) on IBM site e.g. with Secure Shell (SSH) for Unix and Remote Desktop Protocol (RDP) for Windows, at the same time (if necessary) the project members working locally at their own sites, will also access to the hosts with full functionality without having to connect to hosts at IBM site, so it is very flexible for the test project members to select the most appropriate method to perform test project under different situations.

Although the CQ and CC Multisite has provided a good backup strategy for test project data, a backup server such as Tivoli Storage Manager (TSM) had better be used on the IBM site to backup the important data for all the test projects in order to get a higher security of data protection.

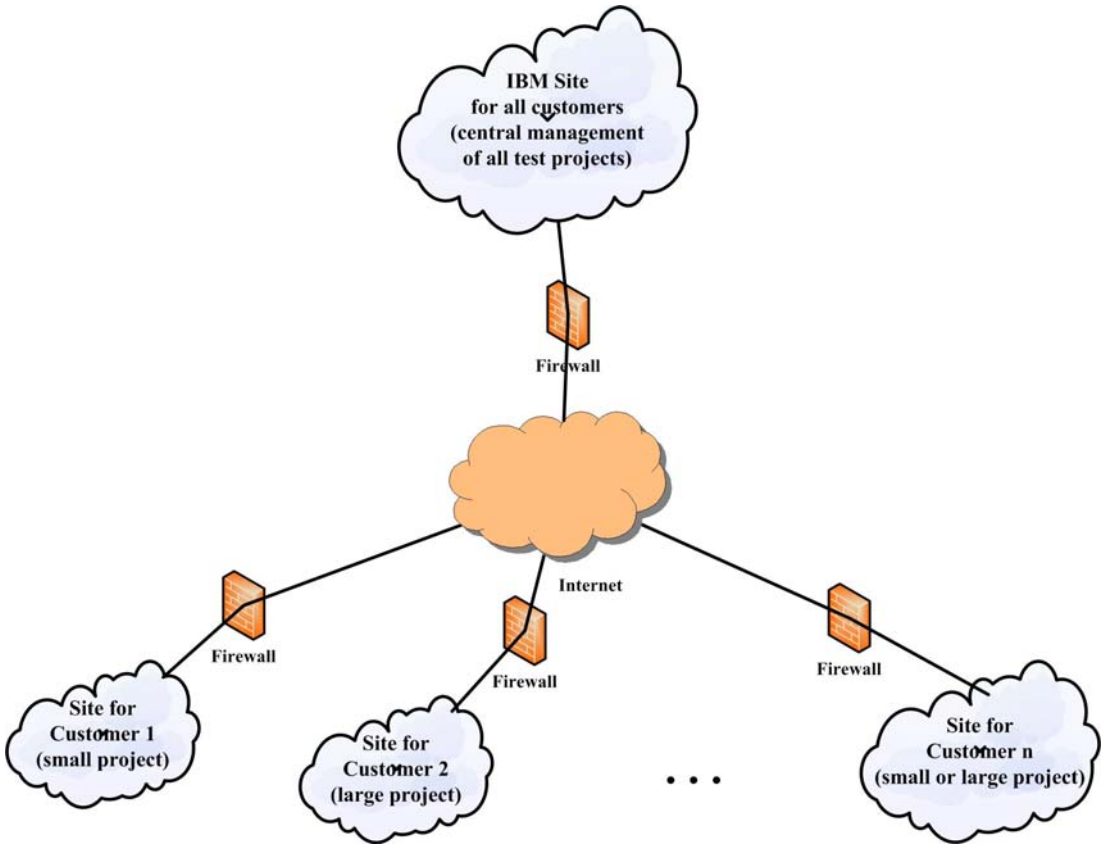


Figure 6.13 Summary of scenarios in VT

7 Discussion

In this thesis a concept named Virtual Testcenter, which functions as a new complete software test solution, has been elaborated, where the necessary components have been included and discussed. Virtual Testcenter would be used to provide custom software test solution or service for IBM internal or external customers by making ready the basic environment for the execution of test projects. In the following the important points about Virtual Testcenter will be discussed.

Requirements on VT from the results of the survey
In order to get requirements on VT a survey has been performed based on a catalog of criterions or influence factors, which define typical project situations, and then its results from industry project praxis has been analyzed to conclude general requirements and two different scenarios (small and large test projects) of specific requirements, where the earlier theoretical analysis about software test, IBM best practices, is-analysis results of available IBM test Assets and the property of custom software have been taken into account, so VT is able to better satisfy the requirements of IBM and its external customers for testing custom software. Particularly the process requirements have been stated in VT, as the processes and their quality and complexity are so important to ensure the test project quality and finally the custom software product quality, so different scenarios of process requirements are defined for different types of test projects and included in this test solution. The communication mechanism between these processes has been also described in each separate process to show the logical and efficient workflow during the whole test project.

Realization of VT through IBM Rational Tools
The requirements on VT need to be realized by tools so that the test project can be performed effectively, thus appropriate IBM Rational Tools such as CQ, CC, RP, RFT, RPT, RMT, etc have been selected and analyzed in order to realize the requirements on VT, where the integration between these tools is analyzed and customized to have these processes communicate with each other very well during the test project. The cooperation and customization of these Rational Tools has proven that the requirements defined in the VT can be implemented and automated, so the test project members should be able to perform custom software test efficiently with this automation solution when they have received the training services mentioned in VT.

Fast start-up of the test projects
As two test project templates for small and large test projects in VT have been defined and designed and the technical environment of hard- and software infrastructure to automate it is set up, so each new test project can have an accelerated start-up from an appropriate test project template without having to make it up by self and much effort can be saved. The test project members can mainly concentrate on their assigned responsibilities for testing their custom software product and better teaming and understanding between members can be got.

Better test project management
VT has established baselines of activities, tasks, work products and roles for necessary processes of the test project and these baselines can be managed and measured during the test project, so a better test project management can be gained. At the same time all

of the test projects will be centrally managed on the IBM site, where hosting and consulting services will be provided to enable IBM internal or external customers to perform each own test project in particular when team members work distributed in different locations. The test project data will be stored and managed on both the central IBM site and each customer's own site, so lower risk and more effective test project control would be ensured through the use of established processes in VT.

Template-based VT
Template concept has been taken advantage of in elaborating this test solution so as to ensure not only the quality of testing custom software products but also solution quality. Many types of templates have been created in this thesis, such as two test project templates have been analyzed and defined for small and large test projects in VT, where processes and their activities, tasks and work products can be templates to describe and guide how team members should work while testing custom software. In particular in the realization of VT the appropriate CQ schema will be designed and customized to work as templates to represent work products and implement the process automation. A RP project template would also be created to automate requirements management for VT. There could be many possibilities of these CQ schemas and RP project template to implement the same requirements on VT, two prototype-like CQ schemas are created by customizing the standard schemas for different process requirements. In addition needed documents and user databases, which are created to record specific test project data, are also used as templates. Template has a very great advantage that it can be reused and further developed or customized to comply with new more specific requirements or functionalities of another very specific test project if required in the future. VT itself is also able to be regarded as template on the level of providing test solution or service.

IBM best practices and industry standards
As VT is a test solution provided by IBM and also for IBM internal, so IBM best practices would be adopted while elaborating some points of this concept to ensure solution quality and better satisfy the requirements of IBM and its external customers, such as OPAL would be referenced and customized for some processes required in VT, and also an industry standard - CMMI has been referenced to define the process requirement about quality and complexity for different scenarios of solutions in VT. At the same time some industry or branch standards need also be used during the testing of so special custom software such as medicine software, security software.

Terminology unified within a test project
During theoretical analysis of testing software, it can be found that in the field of software test theory and industry praxis the terminology is not always unified completely, it cannot be easy to decide which terminology is better, but inside a test project a unified terminology should be got to ensure not only the efficient communication and clearer understanding between team members but also high coverage degree of complete activities, tasks and work products of software test, in order to reach a general goal that defects of software are found and resolved and software quality is improved. VT has regarded different aspects of testing custom software and given a basis for a unified terminology to use across all of test projects particularly for processes, which support the effective execution of test projects. The training service should explain the terminology to each member before software test really begins.

8 Summary

This thesis has elaborated a new complete test solution by bundling the activities for testing custom software with the concept – Virtual Testcenter for IBM Deutschland EAS Rational CoC. VT will include all necessary components such as requirements, tools, processes, hosting, hard- and software infrastructure, training services, etc, to provide test services. VT is able to accelerate the start-up of a new test project and improve test project management control by managing all of test projects centrally. With the help of VT risk will be reduced, productivity will be increased and estimating accuracy will be improved within the test project so that overall project effort will be reduced through the early identification and resolution of defects to improve custom software quality. In the last chapter many other characteristic about VT have also been concluded during the discussion.

Furthermore some aspects for further development about VT will be pointed in the following.

Throughout the whole elaboration process of VT it can be concluded that VT is able to satisfy different kinds of requirements while testing custom software, but in order to get a validation or more accurate feedback and improvement, a survey can be performed from project praxis about CQ schemas, RP project template and other components of VT, after many test projects from different industry fields, with different types and even from different nations have been performed with the help of VT for some time, and then new requirements and improvement factors will be summarized and used to improve test services continuously.

At the same time it can be possible that one customer has at first selected the scenario for small projects as its test project template in the early phase, and then with the time the project could become larger because of some factors such as continuous new more functionalities required on this custom software product or test project group expanding, so that the customer would like to migrate the actual whole test project to the scenario for large projects in VT in order to comply with the high project complexity and ensure test project quality, so the focus point lies in how to move actual test project data e.g. test strategy, test plans, test cases, etc. in test management and defect information in defect management into the corresponding test management and defect management of the new scenario, so an automation mechanism can be developed to realize this requirement, where data structures of records stored in user databases for different scenarios would be compared to elaborate an efficient method for it.

List of tables

List of figures

List of abbreviations

AM Agile Modeling
CC ClearCase
CCM Change & Configuration Management
CCRC ClearCase Romote Client
CM Configuration Management
CMMI Capability Maturity Model Integration
CoC Center of Competence
CQ ClearQuest
CQTM ClearQuest Test Manager
CRC Class, Responsibilities and Collaboration
DB Database
GTS Global Technology Services
GBS Global Business Services
GHz Gigahertz
GUI Graphical User Interface
HW Hardware
HTML Hyper Text Markup Language
IBM International Business Machines
IBM Deutschland EAS IBM Deutschland Enterprise Application Solutions GmbH
IEEE Institute of Electrical and Electronics Engineers
IOC Initial Operational Capability
ISO International Standards Organization
LCA Lifecycle Architecture
LCO Lifecycle Objective
MB Mega Byte
Message Application Programming Interface (MAPI)
MS Microsoft
NGS Nordic Generic Solution
OPAL OnDemand Process Asset Library
PM Project Management
PR Product Release
QA Quality Assurance
RAP Rational Administrator Project
RDP Remote Desktop Protocol
RFT Rational Functional Tester
RMT Rational Manual Tester
ROI Return on Investment
RP RequisitePro
RPT Rational Performance Tester
RUP Rational Unified Process
SCM Software Configuration Management
SD System Development
SMTP Simple Mail Transfer Protocol
SMT Service Management Tool
SOA Service-Oriented Architecture
SPICE Software Process Improvement and Capability Determination
SQA Software Quality Assurance

SQM Software Quality Management
SSH Secure Shell
SW Software
TAS Test Automation Starter
TMM Testing Maturity Model
TPTP Test & Performance Tools Platform
TSM Tivoli Storage Manager
UAT User Acceptance Test
UCM Unified Change Management
UML Unified Modelling Language
VOB Versioned Object Base
VPN Virtual Private Network
VT Virtual Testcenter
XP eXtreme Programming

Glossary

Acceptance criteria: the definition of the results expected from the test cases for acceptance test. The component or system must meet these criteria before it is accepted by a user, a customer, or other authorized entity and implementation can be approved.

Acceptance test: formal testing with respect to customer needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable a user, customer or other authorized entity to determine whether or not to accept the system.

Audit trail: a path by which the original input to a process (e.g. data) can be traced back through the process, taking the process output as a starting point.

Build: an operational version of a system or component that incorporates a specified subset of the capabilities that the final product will provide. Builds are defined whenever the complete system cannot be developed and delivered in a single increment.

Black-box test: Testing, which is either functional or non-functional, is performed based on the test object specification without reference to the internal structure of the component or system.

Boundary value analysis: a black box test design technique in which test cases are designed based on test data that lie along boundaries or extremes of input and output possibilities.

Branch coverage: the percentage of branches that have been exercised by a test suite.

Causal analysis: the evaluation of the cause of major errors, to determine actions that will prevent reoccurrence of similar errors.

Change request: a documented proposal for a change of one or more work item or work item parts.

Control flow analysis: a form of static analysis based on a sequence of events (paths) in the execution through a component or system.

Condition coverage: the percentage of condition outcomes that have been exercised by a test suite.

Custom software: software developed specifically for a set of users or customers. The opposite is standard software.

CQ schema: a pattern or blueprint used in ClearQuest to define the way the data is stored and changed for ClearQuest user database. Process(es) can be automated by creating an appropriate CQ schema.

Data flow analysis: a form of static analysis based on variable usage within the code.

Defect: a variance from expectations, which can cause a component or system to fail to perform its required function.

Detailed test plan: a detailed test plan for a specific level of dynamic testing. It defines what is to be tested and how it is to be tested. The plan typically identified the items to be tested, the test objectives, the testing to be performed, test schedules, personnel requirements, reporting requirements, evaluation criteria, and any risks requiring contingency planning. It also includes the testing tools and techniques, test environment set up, entry and exit criteria, and administrative procedures and controls.

Dynamic test: testing involves the execution of the software of a component or system. Dynamic test is a process of validation by exercising a work product and observing the behaviour of its logic and its response to inputs.

Equivalence partitioning: portion of the component's input or output domains for which the component's behaviour is assumed to be the same from the component's specification.

Entry criteria: a checklist of activities or work items that must be complete or exist, respectively, before the start of a given task within an activity or sub-activity.

Exit criteria: a checklist of activities or work items that must be complete or exist, respectively, prior to the end of a given task within an activity or sub-activity.

Expected results: predicted output data and file conditions associated with a particular test case. Expected results, if achieved, will indicate whether the test was successful or not. Generated and documented with the test case prior to execution of the test.

Functional test: test used to assure that the system meets the business requirements, including business functions, interfaces, usability, audit & controls, and error handling etc.

Inspection: an evaluation technique in which software requirements, design, or code are examined in detail by a person or group other than the author to detect faults, violations of development standards, and other problems.

Integration test: a level of dynamic testing that verifies the proper execution in the interfaces and in the interactions between integrated components or systems.

Master test plan: a plan that addresses testing from a high-level system viewpoint. It ties together all levels of testing (unit test, integration test, system test, acceptance test). It includes test objectives, test team organization and responsibilities, high-level schedule, test scope, test focus, test levels and types, test facility requirements, and test management procedures and controls.

McCabe's cyclomatic number: software metric used to measure the complexity of a program. It directly measures the number of linearly independent paths through a program's source code.

Non-functional test: test of the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, security, usability, maintainability and portability.

Process: a set of interrelated activities that is carried out to produce a valuable result from inputs.

Random test: a black box test design technique where test cases are selected by using a pseudo-random generation algorithm, to match an operational profile.

RDP: Remote Desktop Protocol (RDP) is a multi-channel protocol that allows a user to connect to a networked computer, e.g. Windows refers to RDP client software as either Remote Desktop Connection (RDC) or Terminal Services Client (TSC).

Regression test: a functional type of test, which verifies that changes to one part of the system have not caused unintended adverse effects to other parts.

Requirement: a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. A requirement can be either functional or non-functional. The set of all requirements forms the basis for subsequent development and test of the system or system component.

Retest: testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions.

Risk: a factor that could result in future negative consequences.

Smoke test: a subset of all defined/planned test cases that cover the main functionality of a component or system.

SSH: Secure Shell (SSH) is a network protocol that allows data to be exchanged using a secure channel between two networked devices. SSH is used primarily on Linux and Unix based systems and typically used to log into a remote machine and execute commands.

Statement coverage: the percentage of executable statements that have been exercised by a test suite.

Static test: the process of evaluating a program without executing the program or the detailed manual examination of a work product's characteristics to an expected set of attributes, experiences and standards.

System: a collection of components organized to accomplish a specific function or set of functions.

System test: a dynamic level of testing in which all the components that comprise a system are tested to verify that the system functions together as a whole.

Template: a specific type of guidance defined and elaborated in this thesis that can provide a work product with a predefined table of contents, sections, packages, and/or headings, a standardized format, or define a project type with specific components, or describe process models.

Test coverage matrix: a worksheet used to plan and cross check to ensure all requirements and functions are covered adequately by test cases.

Test case: a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

Test data: the input data and file conditions associated with a specific test case.

Test level: a group of test activities that are organized and managed together. A test level is linked to the responsibilities in a project. Examples of test levels are uni test, integration test, system test and acceptance test.

Test log: a chronological record of relevant details about the execution of tests.

Test of state transition tables: a black box test design technique, where test cases are designed to execute valid and invalid state transitions of the component or system.

Test plan: a document prescribing the approach to be taken for intended testing activities. The plan typically identifies the items to be tested, the test objectives, the testing to be performed, test schedules, entry / exit criteria, personnel requirements, reporting requirements, evaluation criteria, and any risks requiring contingency planning.

Test report: a document describing the conduct and results of the testing carried out for a system or system component.

Test script: a sequence of actions that executes a test case. Test scripts include detailed instructions for set up, execution, and evaluation of results for a given test case.

Test strategy: A high level description of major system-wide activities which collectively achieve the overall desired result as expressed by the testing objectives, given the constraints of time and money and the target level of quality. It outlines the approach to be used to ensure that the critical attributes of the system are tested adequately.

Test type: tests a functional or structural (technical) attribute of the system.

Unit test: the first level of dynamic testing and is the verification of new or changed code in a module to determine whether all new or modified paths function correctly.

Use Case: a sequence of transactions in a dialogue between a user and the system with a tangible result.

Virtual Testcenter: new concept elaborated in this thesis for IBM EAS, in order to realize that the basis environment can be made to ensure fast start of software test projects, which can be controlled and managed centrally as well. All necessary components

needed in Virtual Testcenter such as requirements, tools, processes, hosting, hard- and software infrastructure, training services, etc, used to provide test services are analyzed and outlined.

White-box test: Evaluation techniques that are executed with the knowledge of the implementation of the program. The objective of white box testing is to test the program's statements, code paths, conditions, or data flow paths.

Work product: the result produced by performing a single task or many tasks. A work product, also known as a project artifact, is part of a major deliverable that is visible to the customer. Work products may be internal or external. An internal work product may be produced as an intermediate step for future use within the project, while an external work product is produced for use outside the project as part of a major deliverable. As related to test, a software deliverable that is the object of a test, a test work item.

Bibliography

[AgileModeling] Agile modeling http://www.agilemodeling.com/ (10 July 2008)

[Aldebaran08] http://www.aldebaran.de/Individualsoftware.52.0.html (11 July 2008)

[Balzert98] Helmut Balzert.: Lehrbuch der Software-Technik; Band 1 und 2; Spektrum Akademischer Verlag, 1998.

[Boehm88] Barry W. Boehm : „A Spiral Model of Software Development and Enhancement", IEEE 1988 , http://www.cs.usu.edu/~supratik/CS%205370/r5061.pdf (27 June 2008)

[Bucanac99] Christian Bucanac: The V-Model http://www.bucanac.com/documents/The_V-Model.pdf (05 July 2008)

[CC] Coley consulting: Types of testing http://www.coleyconsulting.co.uk/testtype.htm (15 July 2008)

[CMMI06] Carnegie Mellon Software Engineering Institute – „CMMI for Development Version 1.2", August 2006, http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr008.pdf (08.July.2008)

[DAS08] Improve business value with Defect Analysis Starter, 24 June 2008 (IBM-internal) http://w3.ibm.com/news/w3news/top_stories/2008/06/gbs_testing.html (27 June 2008)

[Dustin02] Elfriede Dustin : Evaluating a Tester's Effectiveness, 03 November 2002, http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=3281&tth=DYN&tt=siteemail&iDyn=2 (01 July 2008)

[Dustin03] Elfriede Dustin : Effective software testing : 50 Specific Ways to Improve Your Testing, Addison Wesley. 2003

[EM07] Gerald D. Everett , Raymond, Jr. McLeod :Software Testing: Testing Across the Entire Software Development Life Cycle, Wiley-IEEE Computer Society Pr, 2007

[FG99] Fewster, M.; Graham, D.: Software Test Automation; Effective use of test execution tools, Addison Wesley, 1999

[IBMGermany] IBM in Deutschland http://www05.ibm.com/de/news/oneibm/index.html?ca=one_ibm_de&me=w&met=de_hplg (15 June 2008)

[IEEE] IEEE Software Engineering Standards Zone http://standards.ieee.org/software/ (25 June 2008)

[IEEE1028] IEEE Standard for Software Reviews, IEEE Std 1028-1997 http://en.wikipedia.org/wiki/Software_review

[IEEE90] Institute of Electrical and Electronics Engineers : „IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries", 1990, ISBN 978-1559370790

[IEEE98] IEEE Standard 829-1998 – „IEEE Standard for Software Test Documentation" http://se.inf.ethz.ch/teaching/ss2005/0050/exercises/REMOVED/IEEE%20Std%20829-1998%20test.pdf (08 July 2008))

[Imbus08] http://www.imbus.de/index.shtml (11 July 2008)

[Infodat] Ist Individual-Software noch zeitgemäß ? http://www.infodat.org/html/individual_software.html (23 June 2008)

[IRCC] IBM Rational ClearCase Help Documentatin 2008

[IRCQ] IBM Rational ClearQuest Help Documentatin 2008

[IRFT] IBM Rational Functional Tester Help Documentatin 2008

[IRMT] IBM Rational Manual Tester Help Documentatin 2008

[IRRP] IBM Rational RequisitePro Help Documentatin 2008

[IRPT] IBM Rational Performance Tester Help Documentatin 2008

[ISO9126] ISO Standard 9126 „Software engineering – product quality", International Standard Organization, 2001-2004

[Jungmayr04] Stefan Jungmayr : Improving Testability of object-oriented systems, http://www.dissertation.de/FDP/sj929.pdf, 2004, ISBN 3-89825-781-9

[Jungmayr05] Stefan Jungmayr : Testbarkeit im Entwicklungsprozess, 2005, http://www.testability.de/Publikationen/TAE05_Artikel_jungmayr.pdf (07 July 2008)

[Kahlbrandt 98] Bernd Kahlbrandt. Software-Engineering: objektorientierte Software-Entwicklung mit der Unified modeling language. Springer, 1998.

[Konda05] Kalyana Rao Konda : Measuring Defect Removal Accurately, http://www.stpmag.com/downloads/stp-0507_testmetrics.htm (05 August 2008)

[KR05] Per Kroll, Walker Royce: „Key principles for business-driven development", IBM DeveloperWorks, Oktober 2005 http://www-128.ibm.com/developerworks/rational/library/oct05/kroll/ (11 July 2008)

[Kruchten03] Philippe Kruchten: The Rational Unified Process -- An Introduction, Third Edition, Addison Wesley Longman, 2003.

[LFH08] http://www.lessons-from-history.com/Level%203/functional%20vs%20non-functional.html, (2 August 2008)

[MSBT04] Glenford J. Myers, Corey Sandler (Revised by), Tom Badgett (Re vised by), Todd M. Thomas (Revised by): The Art of Software Testing, 2nd Edition, John Wiley & Sons, Inc. 2004

[OST08] http://www.onestoptesting.com/test-metrics/types.asp (05 August 2008)

[Royce70] Winston W. Royce – „Managing the Development of Large Software Systems", Proceedings IEEE WESCON, 1970, http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf (02 July 2008)

[RUPIBM] Rational Unified Process http://www-306.ibm.com/software/awdtools/rup/ (03 July 2008)

[SEICMMI08] the CMMI web site http://www.sei.cmu.edu/cmmi/index.html (15 August 2008)

[SL02] Andreas Spillner, Tilo Linz: Basiswissen Softwaretest, dpunkt.verlag, 2002

[SR] Kevin Yeung-Kuen See, Pierre Regazzoni: Test Management and Tracking with the IBM Rational ClearQuest Test Management http://www.ibm.com/developerworks/rational/library/07/0306_see_regazzoni/ (20 August 2008)

[Sullivan07] Angela L. Sullivan : IBM Testing Services Takes Off , November. 2007 http://w3.ibm.com/news/w3news/top_stories/2007/11/gbs_testing_services.html (27 June 2008)

[Testbarkeit08] http://www.testbarkeit.de (15 June 2008)

[TSE] The-Software-Experts, Software process models, http://www.the-software-experts.de/e_dta-sw-process.htm (15 July 2008)

[TSWP] The Testing Standards Working Party http://www.testingstandards.co.uk/ (01 July 2008)

[Vaher04] Liina Vaher : Potenziale und Risiken von Standard- und Individualsoftware, http://_ws03_04/www/Vaher/Homepage/index.htm (5 June 2008)

[Wiki08a] http://en.wikipedia.org/wiki/ISO_9126 (14 July 2008)

[Wiki08b] http://de.wikipedia.org/wiki/Individuall%C3%B6sung (21 July 2008)

[Wiki08c] http://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration (25 July 2008)

[XPHP08] http://www.extremeprogramming.org/index.html (13 June 2008)

Appendix

Capability Maturity Model Integration (CMMI)

Capability Maturity Model Integration (CMMI) is a process improvement approach that provides organizations with the essential elements of effective processes. It can be used to guide process improvement across a project, a division, or an entire organization. [SEICMMI08] CMMI is accepted today world-wide as the Industry Standard Software Development Model.

CMMI best practices are published in documents called models, which each address a different area of interest. There are now two areas of interest covered by CMMI models: Development and Acquisition.

CMMI model defines a set of process areas (PA) (e.g. Project Planning, Requirements Management, Organizational Training), which describe the aspects of product development that are to be covered by organizational processes. Each process area has 1 to 4 *goals*, and each goal is comprised of *practices*. These goals and practices are called *specific* goals and practices, as they describe activities that are specific to a single process area. An additional set of goals and practices applies across all of the process areas; this set is called *generic* goals and practices. CMMI V1.2 for development model contains 22 *process areas*, where four process area categories used: Project Management, Process Management, Engineering, and Support.

There are 2 different representations in CMMI: staged and continuous:

Staged representation:

Process improvement is measured using maturity levels. Maturity level is the degree of process improvement across a predefined set of process areas. Organizational maturity pertains to the “maturity” of a set of processes across an organization.

There are five maturity levels (1-5):

Maturity level 1: Initial

Processes are unpredictable, poorly controlled and reactive.

Maturity level 2: Managed

Processes are characterized for projects and actions are often reactive.

Maturity level 3: Defined

Processes are characterized for the organization and actions are proactive.

Maturity level 4: Quantitatively managed

Processes are both measured and controlled.

Maturity level 5: Optimizing

Process improvement is a continuous focus.

An organization can be appraised using an appraisal method like *Standard CMMI Appraisal Method for Process Improvement (SCAMPI)* and awarded a 1-5 level rating. The rating results of such an appraisal can be published if released by the appraised organization. [Wiki08c]

Continuous representation:

Process improvement is measured using capability levels. Capability level is the achievement of process improvement within an individual process area. Process area capability pertains to the “maturity” of a particular process across an organization. There are six capability levels (0 to 5). For capability levels 1-5, there is an associated generic goal. Each level is a layer in the foundation for continuous process improvement. Thus, capability levels are cumulative, i.e., a higher capability level includes the attributes of the lower levels.

Both representations provide ways of implementing process improvement to achieve business goals. Essentially the same content is provided in both representations, which are yet organized in different ways. However, it is crucial to choose the right representation depending on what is trying to be achieved:

Continuous Representation is used for improving specific process areas – (Example: if parts of the lifecycle is not contracted to IBM e.g. customer retained Requirements Development and User Acceptance testing activities)

Staged representation used for improving overall organizational process maturity – (Example: IBM has contractual responsibility for all lifecycle activities and the requirement to move up in process maturity (ML2 to ML3) by a target date)

CMMI conformance can be attained by following either staged or continuous representation of the model.

CMMI only defines the fundamental practices – ”What”, but not the concrete steps – ”How”, i.e., CMMI doesn’t endorse/mandate any particular method for compliance, so in practice some approaches can be developed to be compatible with CMMI.

OnDemand Process Asset Library (OPAL)

The *OnDemand Process Asset Library* (OPAL) represents the GBS implementation of the *Worldwide Project Management Method* (WWPMM), which defines IBM's common project management method for IBM projects worldwide. WWPMM describes the way projects in IBM are managed. WWPMM is sponsored by the Project Management Center of Excellence, to support a corporate action directing to design and implement a single, common project management method for IBM projects worldwide. The methods that comprise WWPMM are based on decades of experience derived from many different types of projects in a variety of constituencies.

It includes GBS tailored procedures and templates that satisfy the corporate practice for project management (WWPMM). By implementing OPAL, users have implemented WWPMM and there are no additional requirements to deploy WWPMM .

OPAL provides the assets and guidance required to plan and manage projects to meet client needs and support CMMI capability Level - from Level 2 to Level 5. OPAL is designed to be used as a reference management system that is used in conjunction with the IBM *Unified Method Framework (UMF).*

Survey of requirements on Virtual Testcenter

The survey of the customers' requirements on Virtual Testcenter and its results (the number stands for the count of customers, who have selected this choice):

This survey has been performed using the following 10 questions:

1. **Project size**
   - small → 0
   - medium →16
   - large →20

2. **Process models of customers or projects (multiple choice)**
   - ITIL → 4
   - V-Model XT → 4
   - CMMI → 20
   - SPICE → 1
   - SOX → 2
   - UMF → 13

3. **Project language**
   - German → 25
   - English → 15
   - Others

4. **Geography**
   - central → 0
   - non-central → 39

5. **Hosting strategy for test tools**
   - on customers → 20
   - on IBM → 11

6. **Number of non-IBM suppliers**
   - zero
   - up to 5 → 35
   - more than 5 → 5

7. **"Mind-set" of customers regarding test processes / structured test**
   - basis → 10
   - advanced → 26
   - expert/professional → 0

8. **Sensitivity of customer data**
   - not relevant → 1
   - possible storage on IBM → 13
   - must stay on customers → 22

9. Further use of test processes / test tools after finished projects
- o not planned → 1
- o planned → others

10. Desired covering of lifecycle (multiple choice)
- o requirements → 22
- o configuration → 20
- o defect → 35
- o change → 26
- o test → 35

Here the interpretation of each question will be made as follows:

1. **Project size**: is determined by the complexity of the project, as can be defined e.g. by the complexity of software source code, by the complexity of business processes, by number of team members (users and developers), by the duration of the project, by the contract price, or by the geography etc

2. **Process models of customers or projects**: which process model is adopted to support development activities? Or what model is used to assess and improve the test process quality, so that the software quality is ensured? (Relatively more projects have selected CMMI.)

3. **Project language**: what language will be used in the project? For the international projects and teams English is always selected.

4. **Geography**: do team members work together in the same location (central) or distributed in different locations (non-central) e.g. different cities or nations?

5. **Hosting strategy for test tools**: how are the used testing tools managed and accessed? On the side of IBM or customers?

6. **Number of non-IBM suppliers**: asks if non-IBM suppliers are required, if yes, how many non-IBM suppliers need to attend the project?

7. **"Mind-set" of customers regarding test processes / structured test**: what mind-set do customers have regarding test processes? Have they only basis, or advanced or even professional knowledge or requirements about test processes?

8. **Sensitivity of customer data**: how and where should customer data (e.g. test plan, test cases, and test scripts, defect etc. or other data produced during test process cycle) be stored?

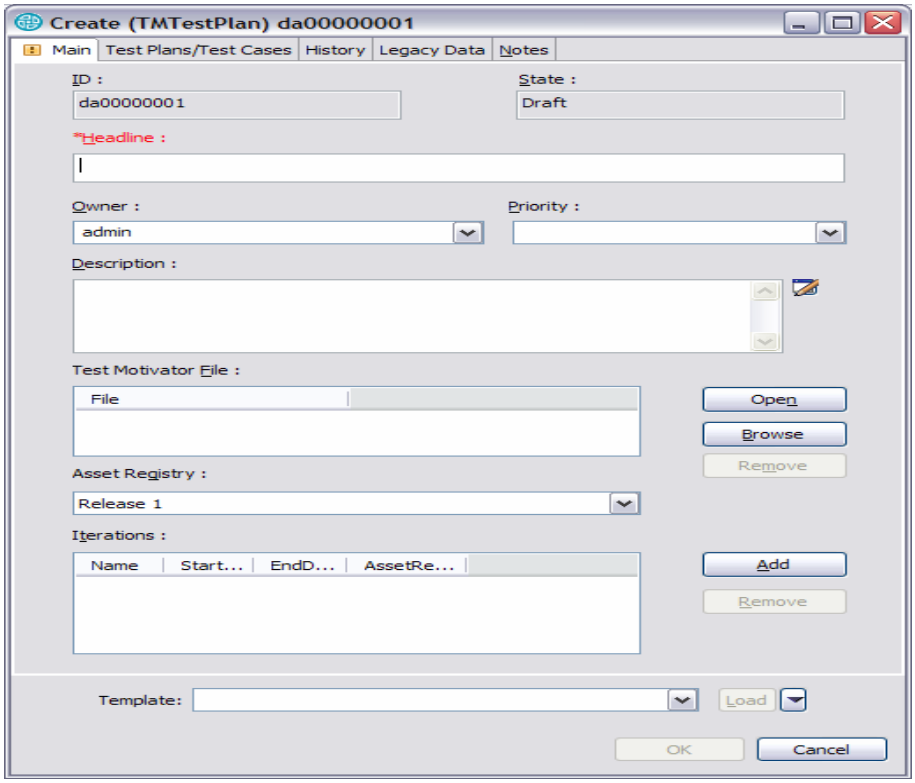
9. **Further use of test processes / test tools after finished projects**: determines if the available environment of test processes and test tools should be further used after projects are finished.

10. **Desired covering of lifecycle**: which of these workflows or processes of the project lifecycle should be required for performing software test projects: requirements management, configuration management, defect management, change management or test management?

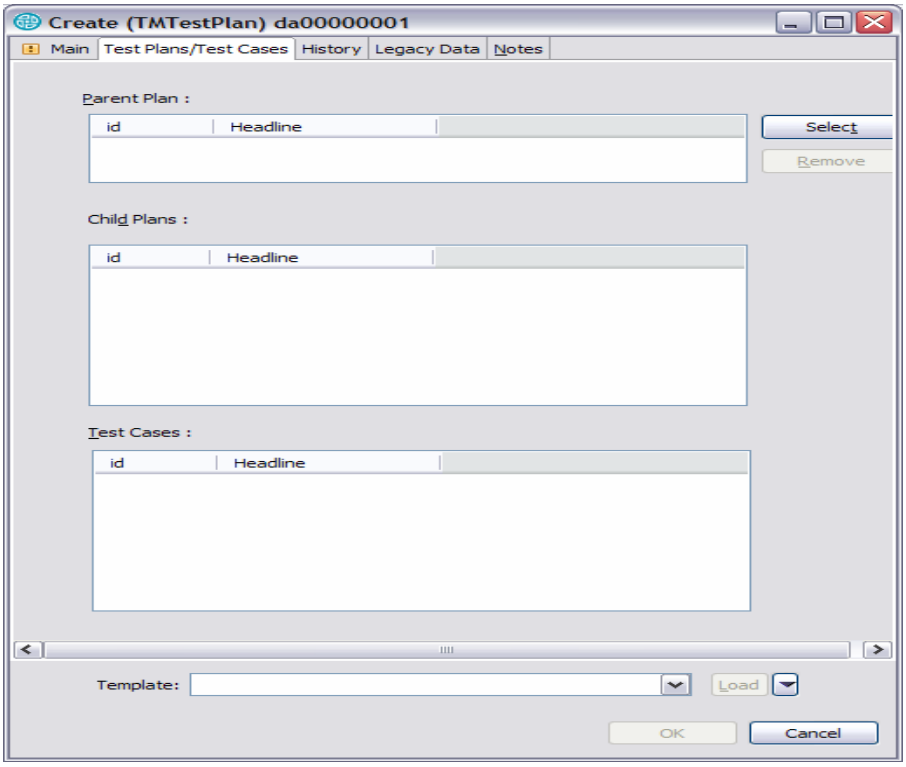
The main forms of CQ record types used in the processes for small projects

Test management

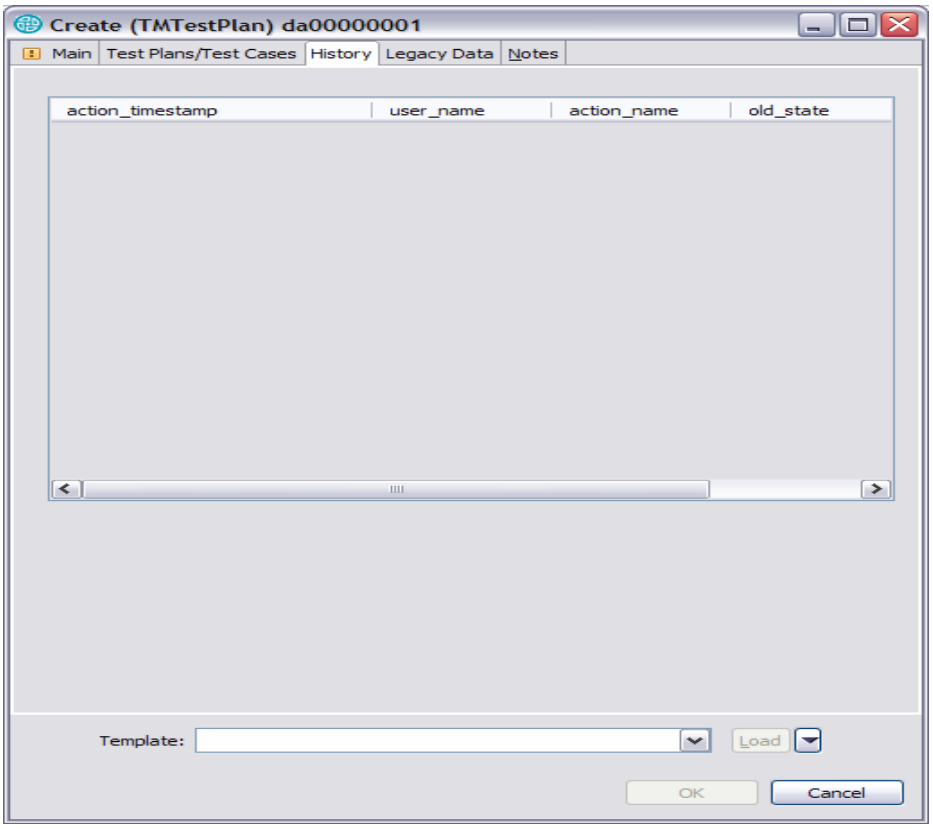
Test Plan



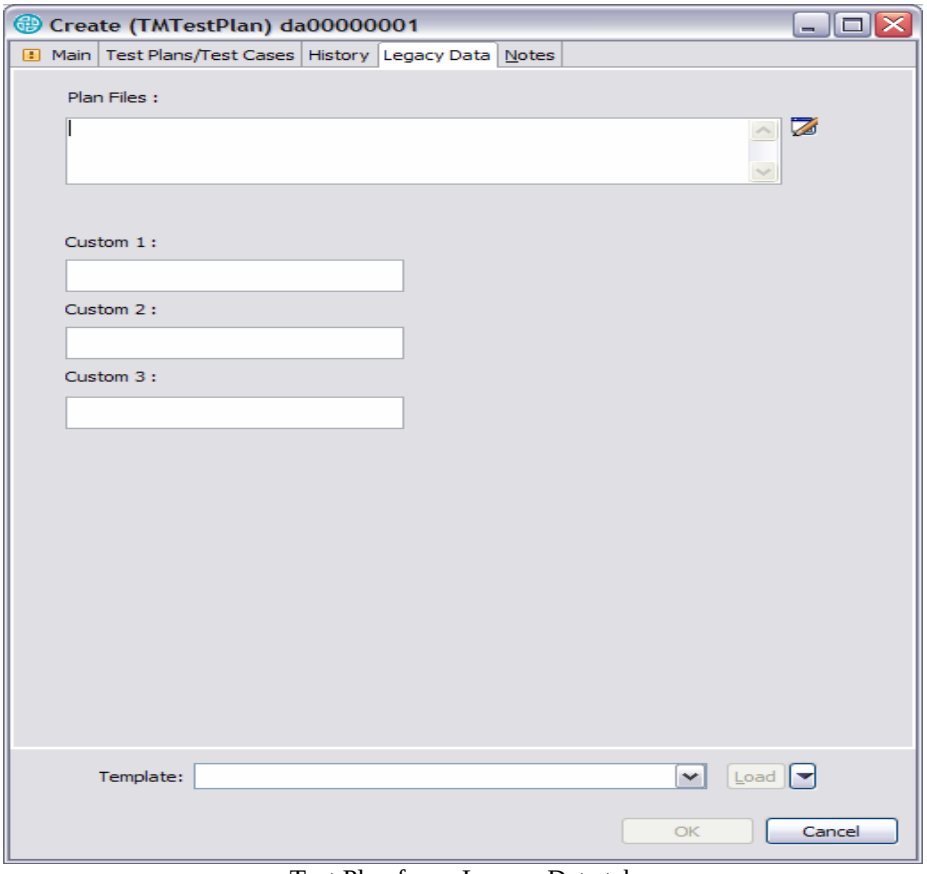
Test Plan form, Main tab



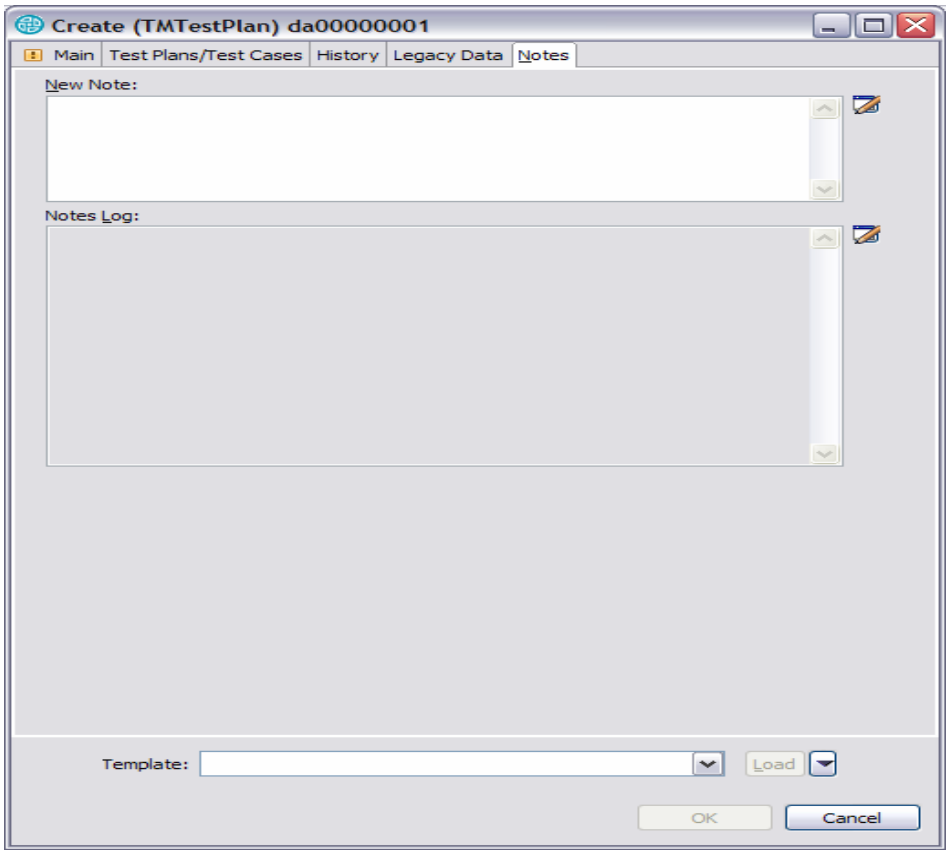
Test Plan form, Test Plans / Test Cases tab



Test Plan form, History tab

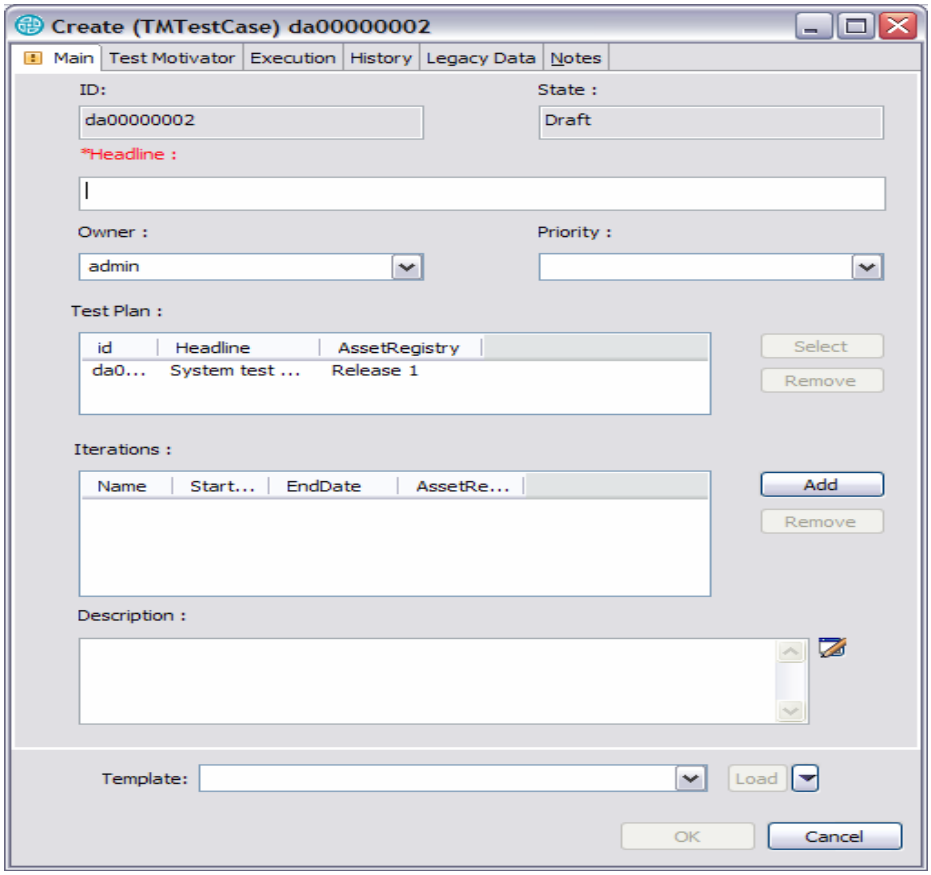


Test Plan form, Legacy Data tab

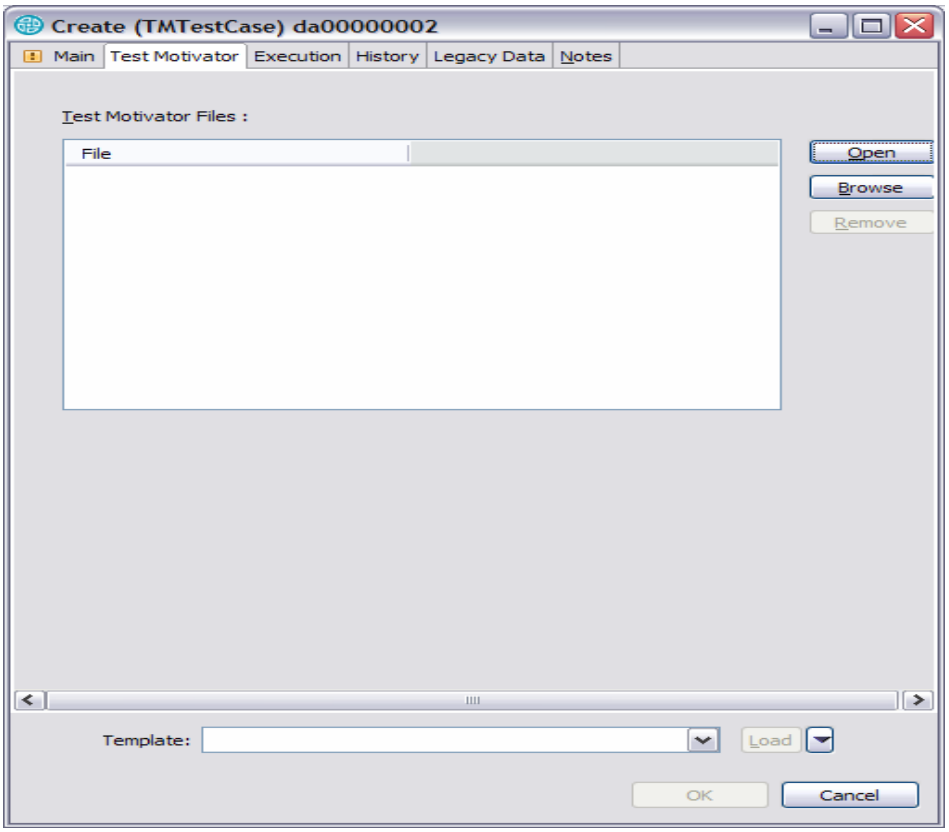


Test Plan form, Notes tab

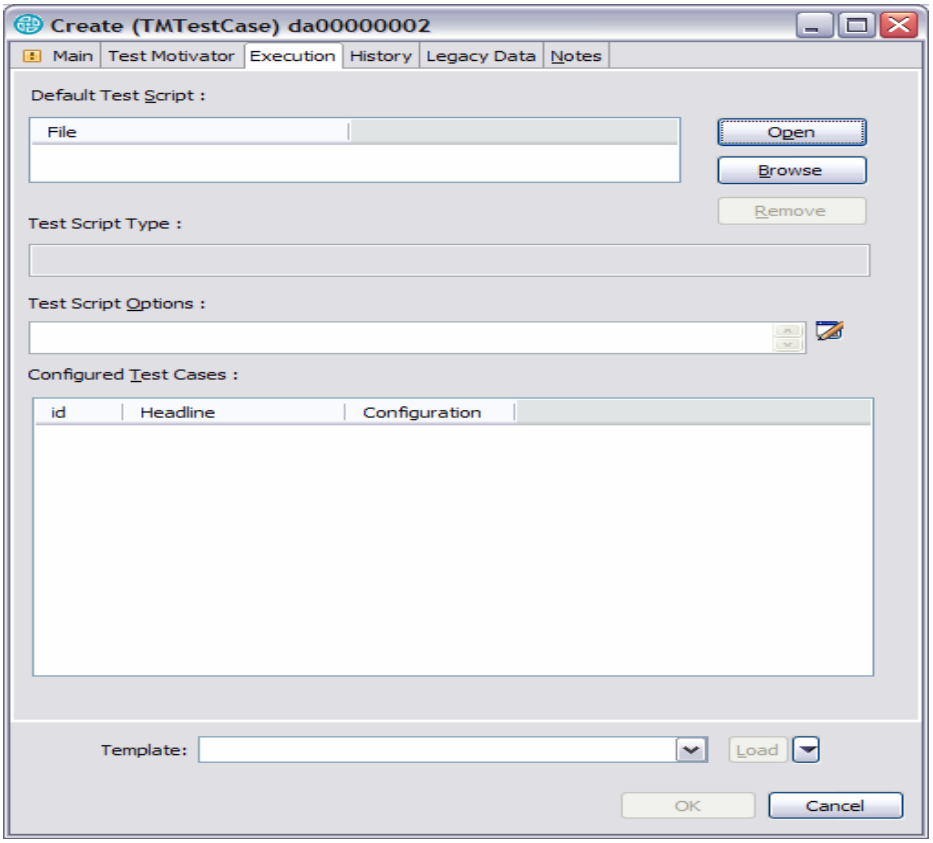
Test Case



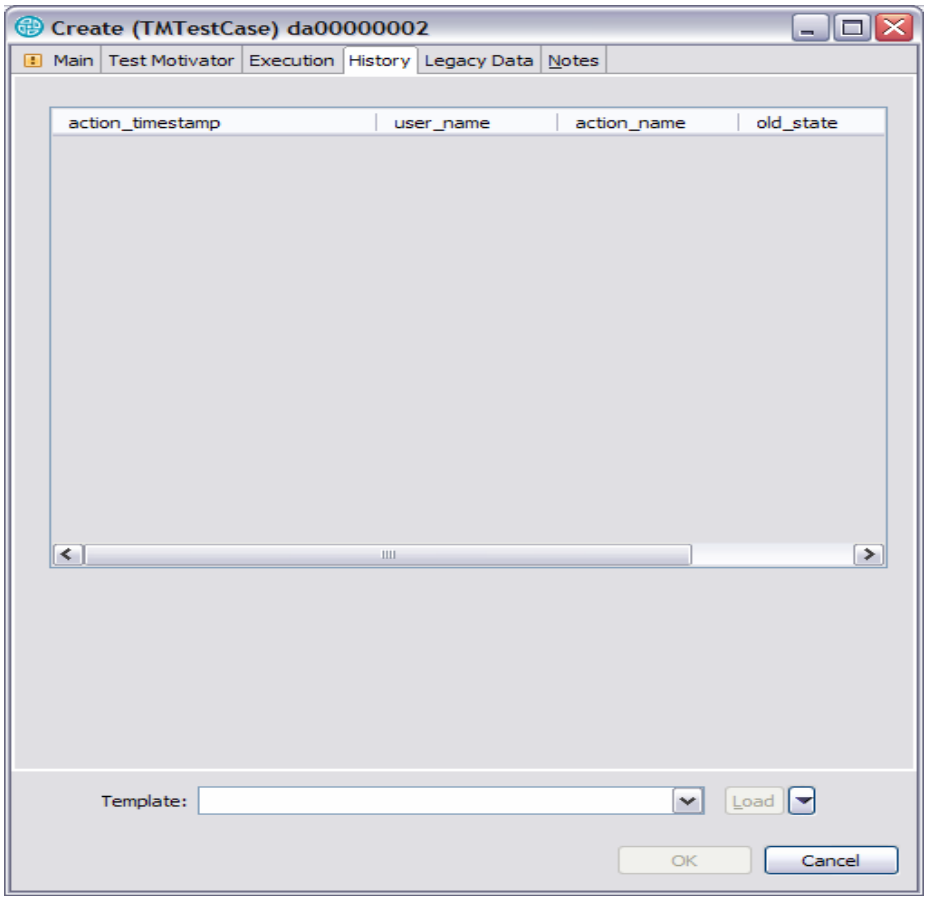
Test Case form, Main tab



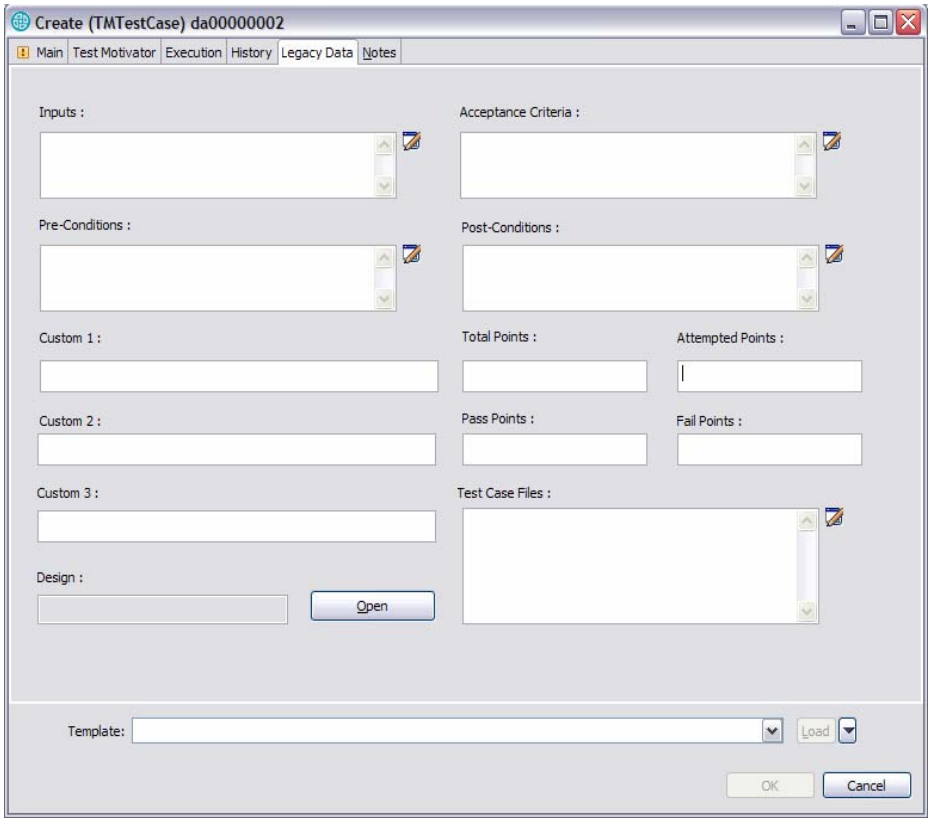
Test Case form, Test Motivator tab



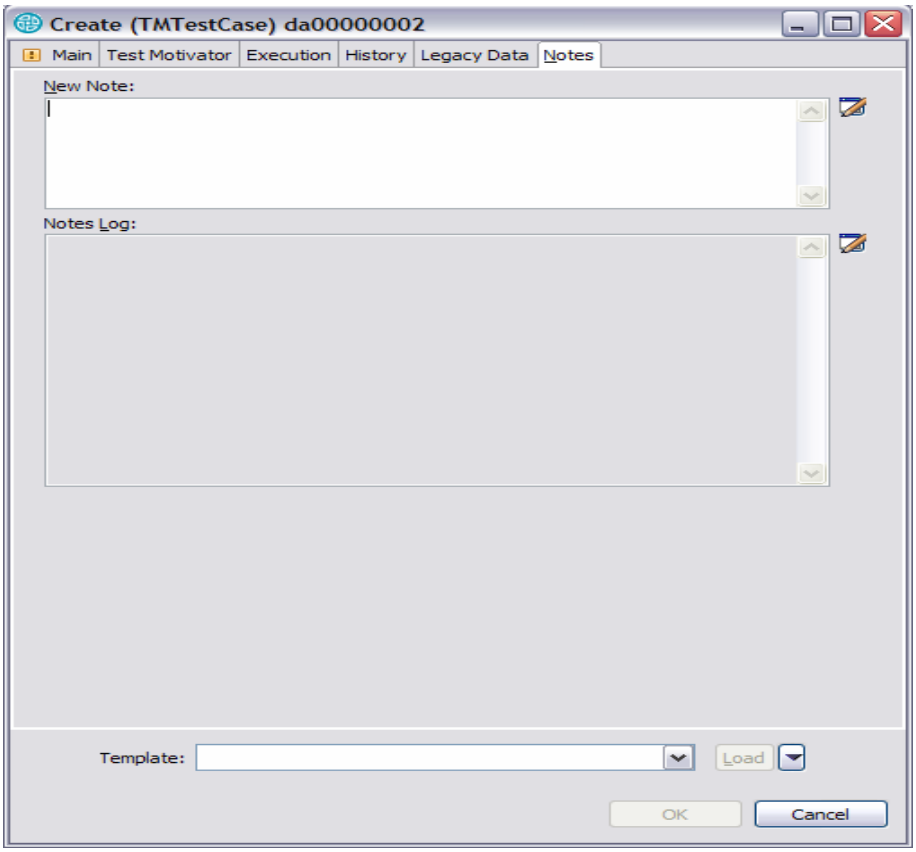
Test Case form, Execution tab



Test Case form, History tab

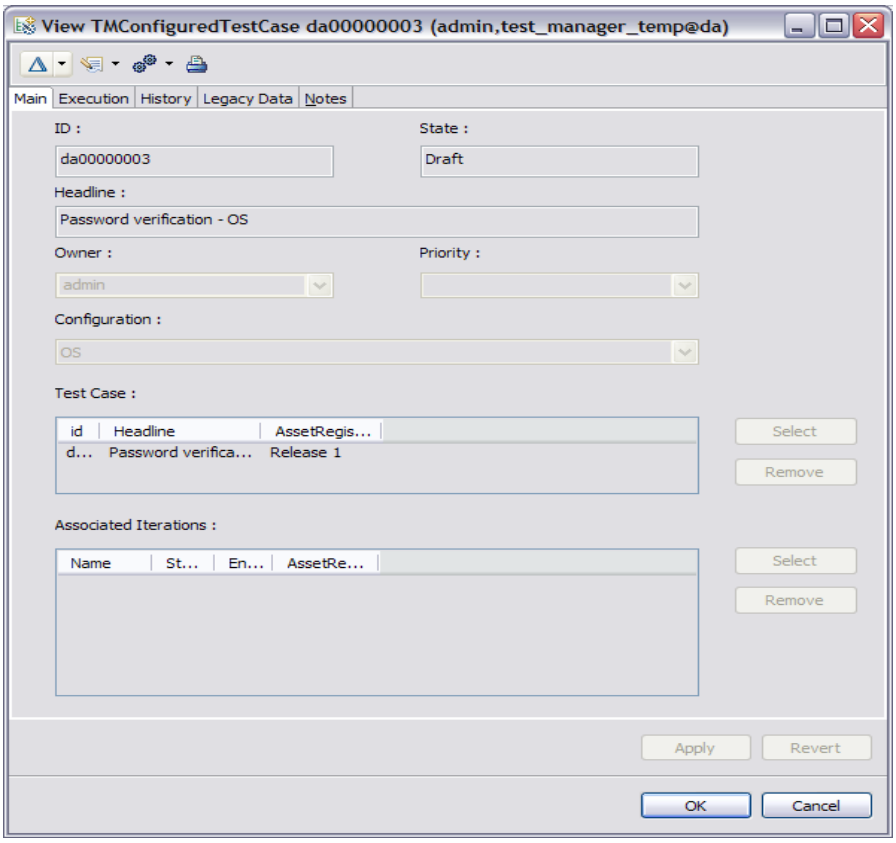


Test Case form, Legacy Data tab

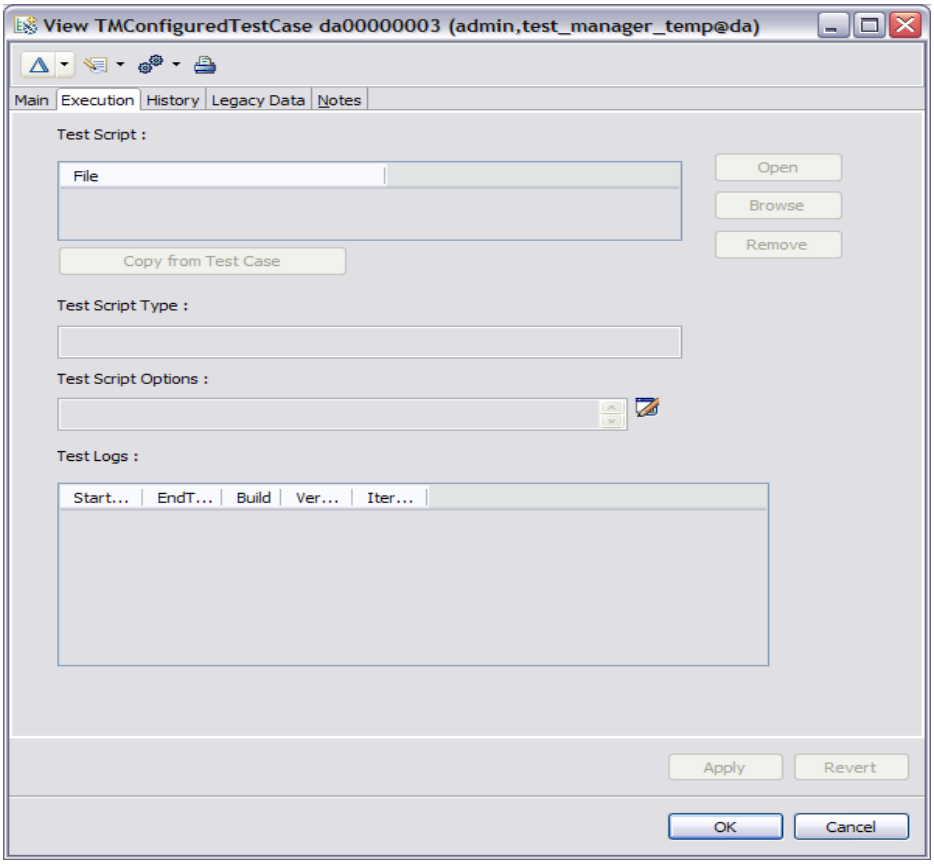


Test Case form, Notes tab

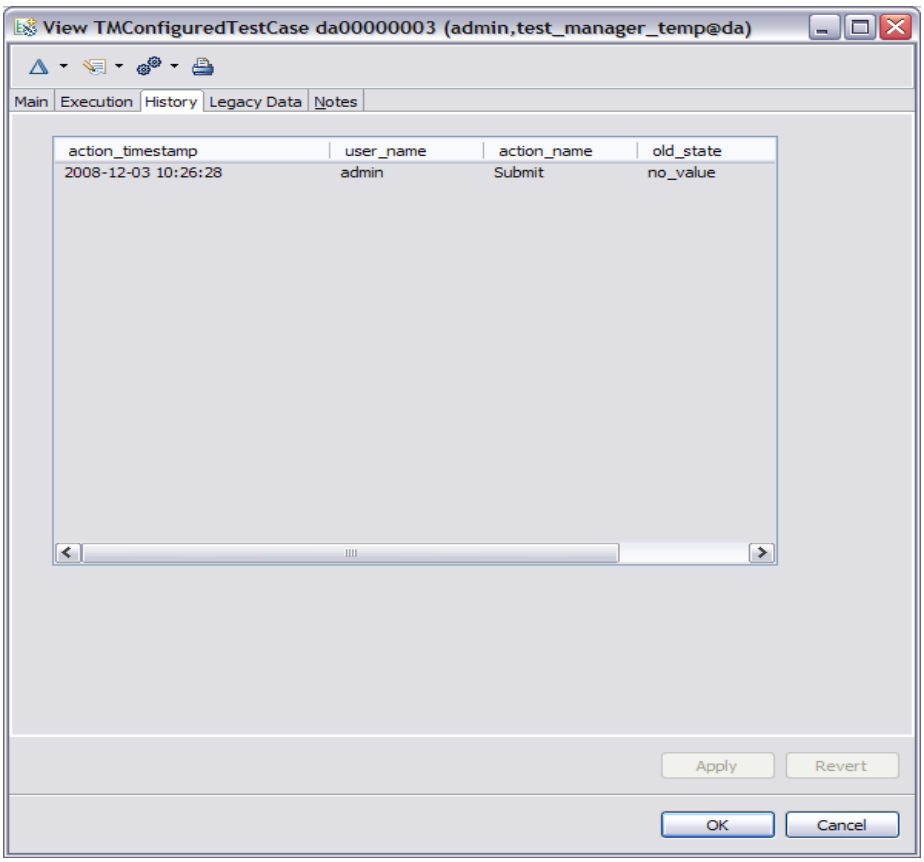
Configured Test Case



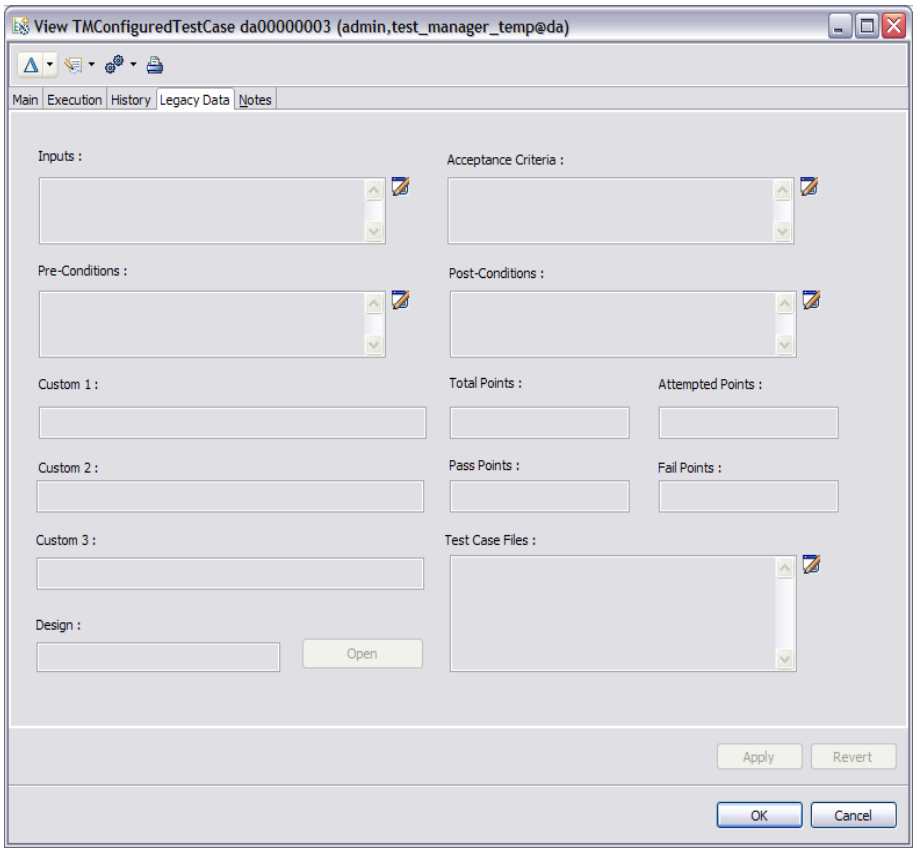
Conf.Test Case form, Main tab



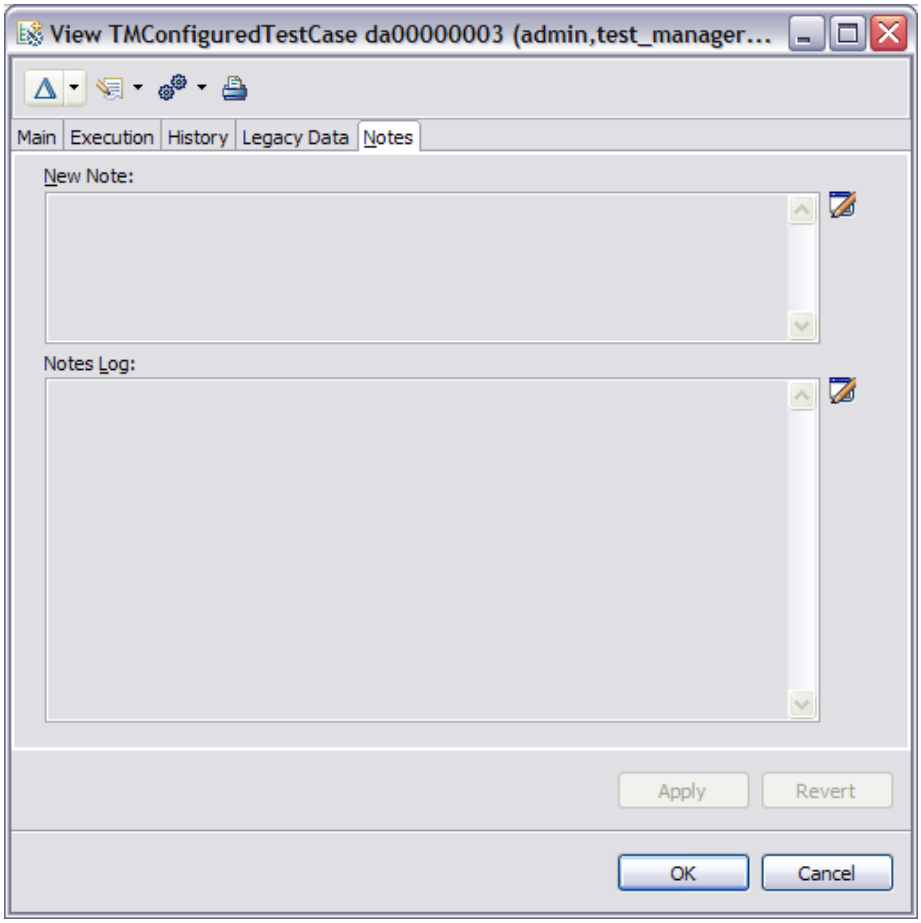
Conf.Test Case form, Execution tab



Conf.Test Case form, History tab

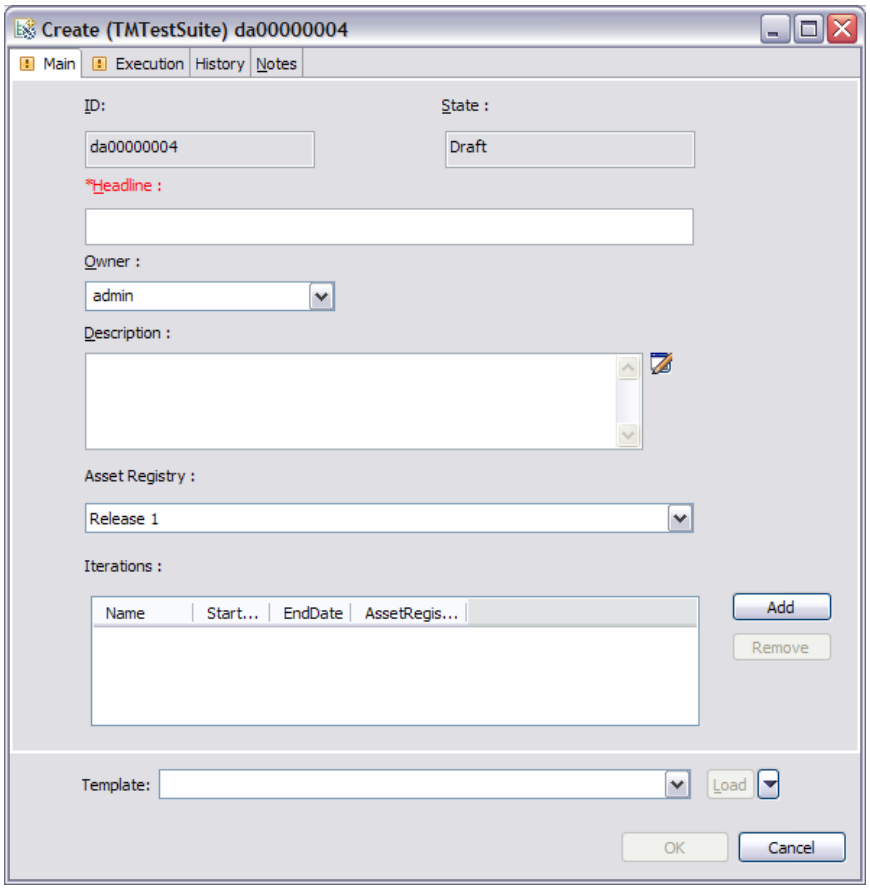


Conf.Test Case form, Legacy Data tab

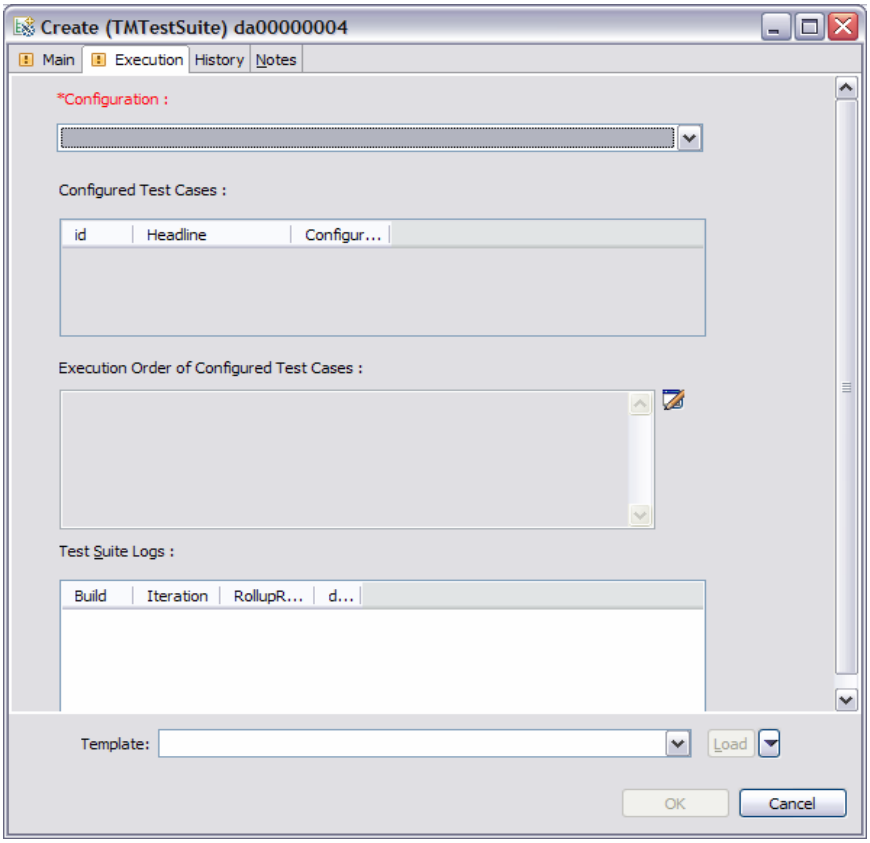


Conf.Test Case form, Notes tab

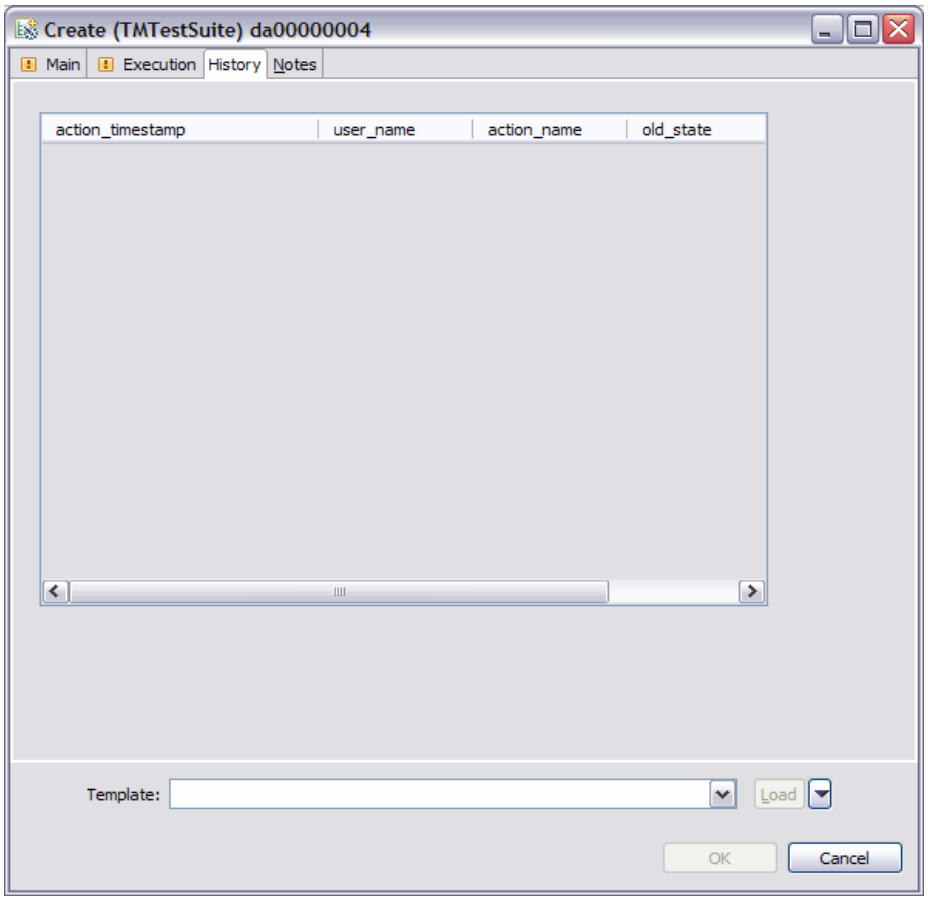
Test Suite



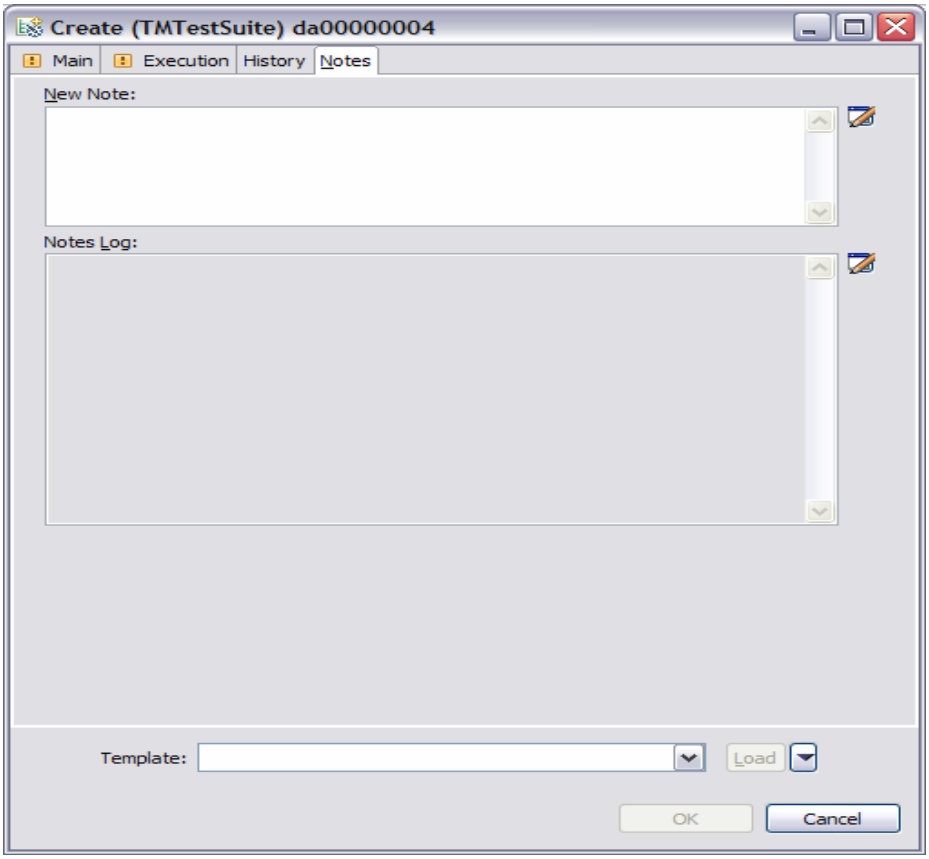
Test Suite form, Main tab



Test Suite form, Execution tab

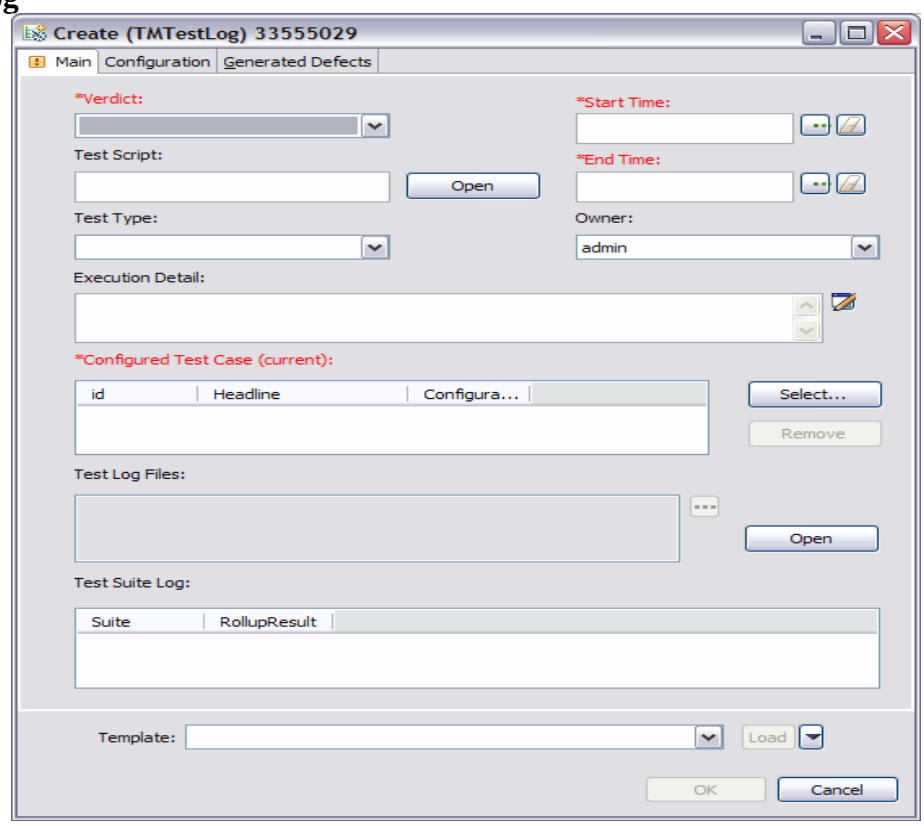


Test Suite form, History tab

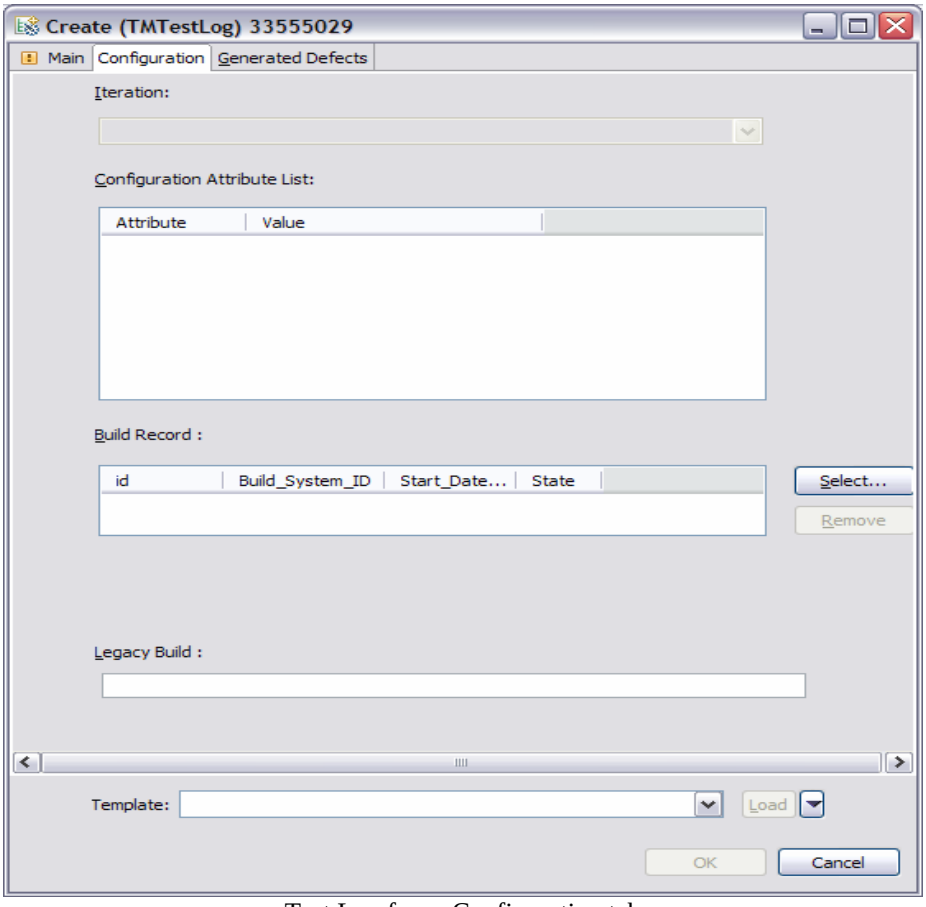


Test Suite form, Notes tab

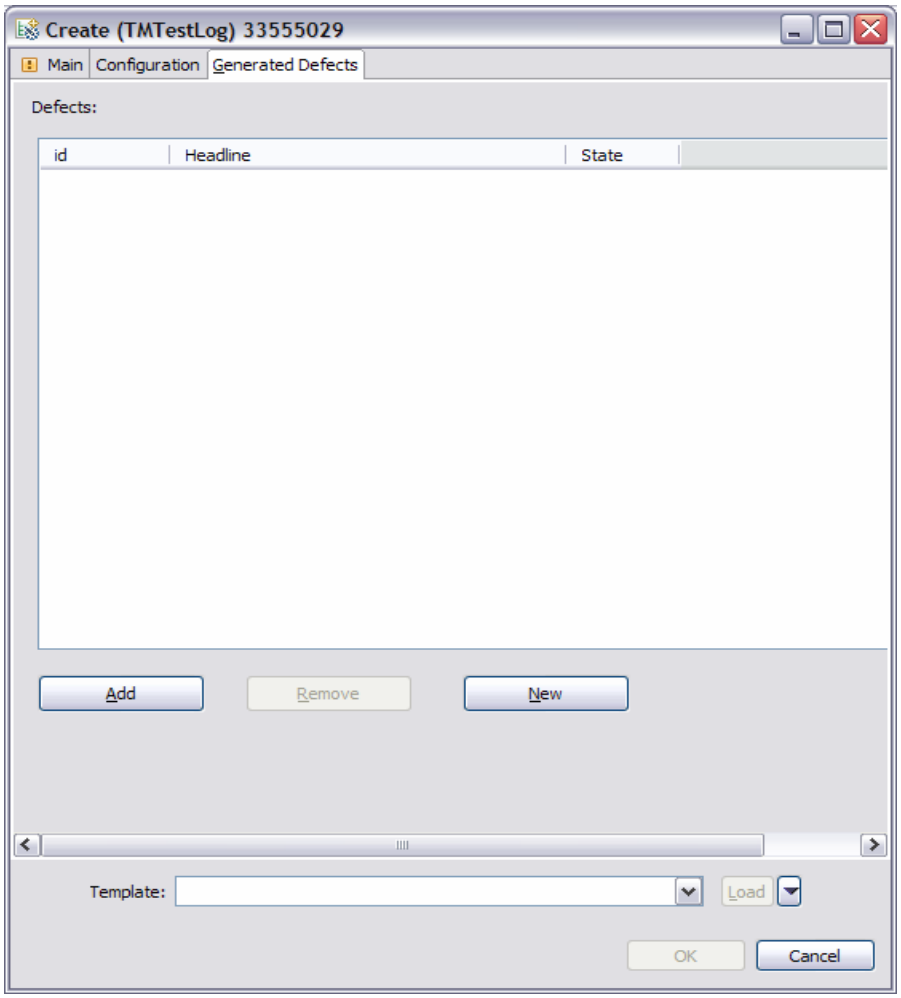
Test Log



Test Log form, Main tab

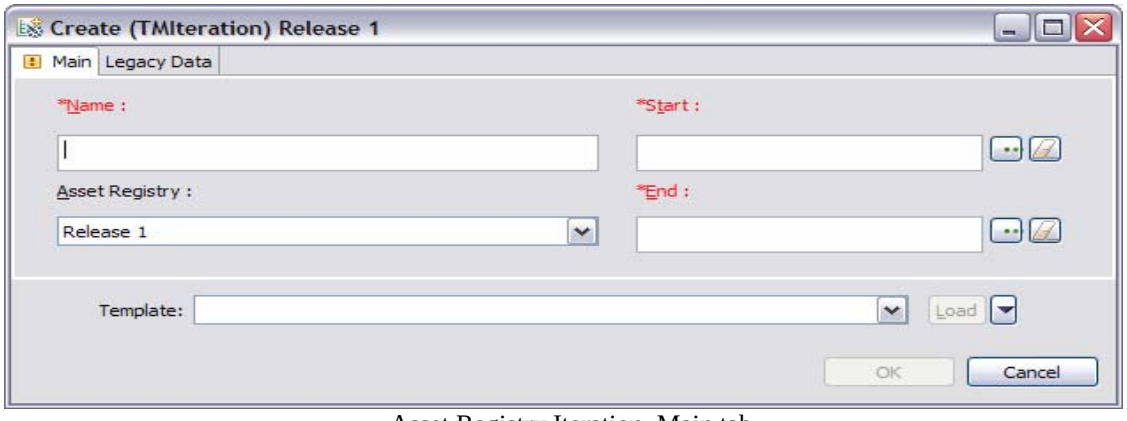


Test Log form, Configuration tab

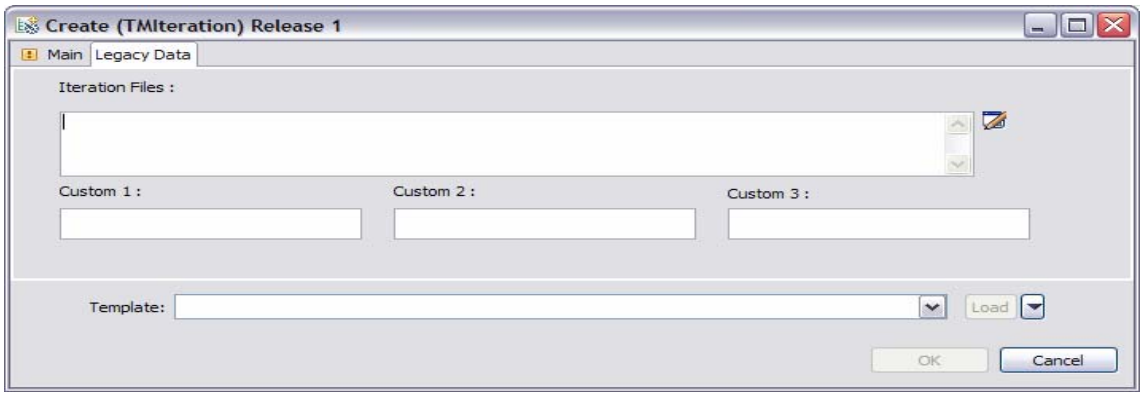


Test Log form, Generated Defects tab

Iteration

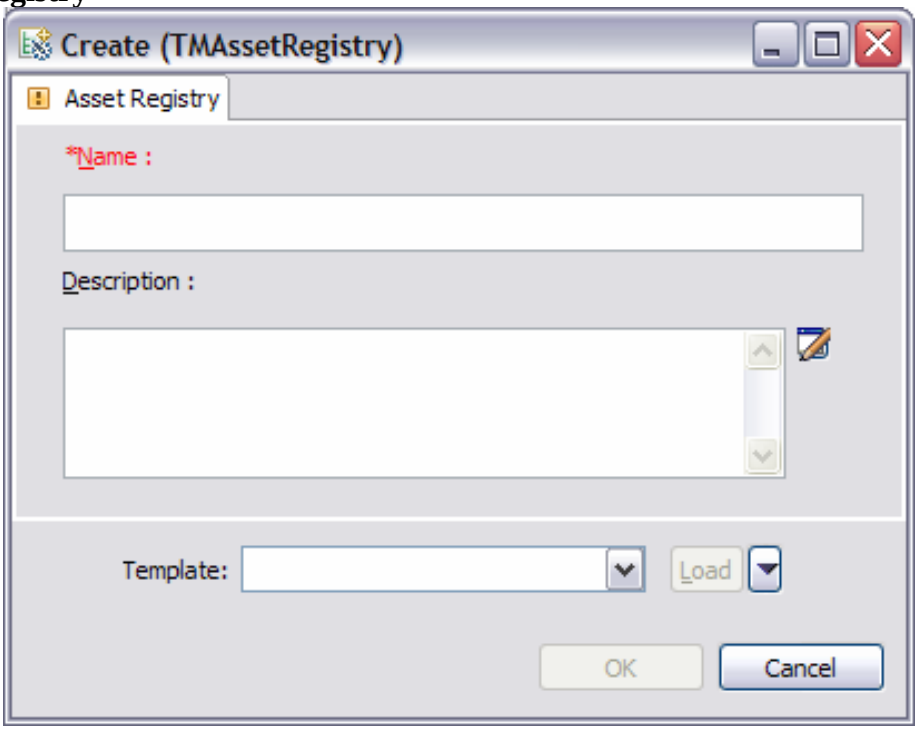


Asset Registry Iteration, Main tab



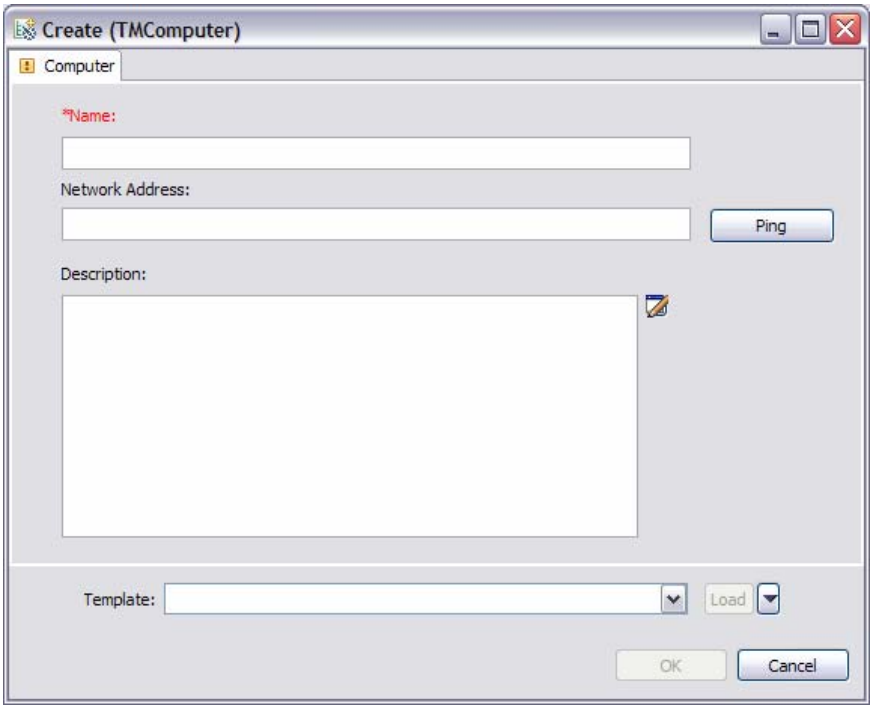
Iteration, Legacy Data tab

Asset Registry

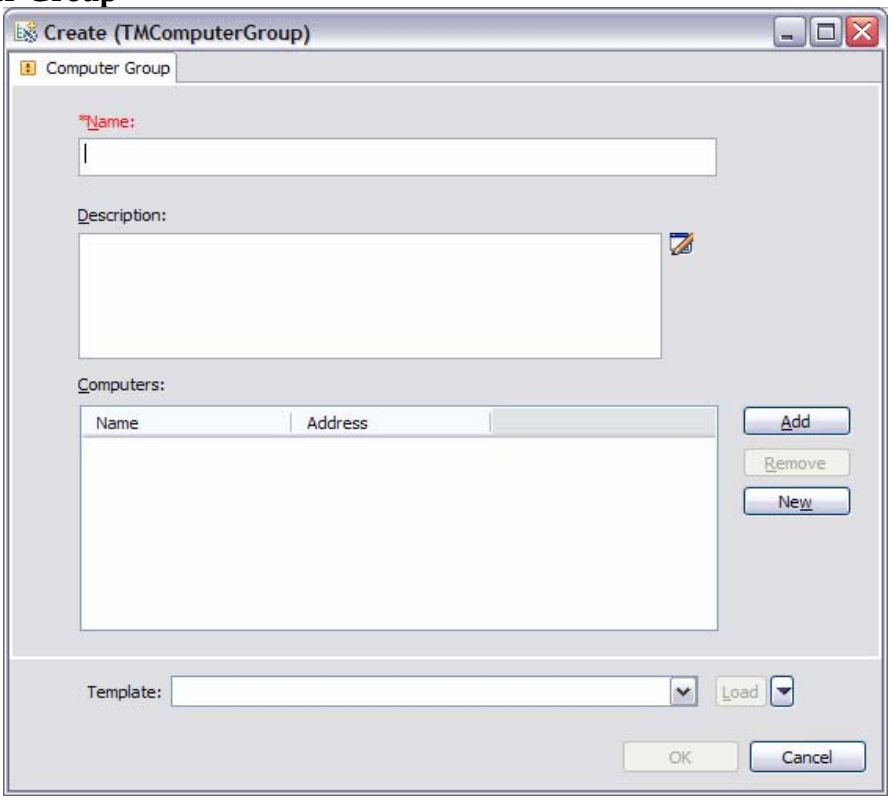


Asset Registry tab

Computer



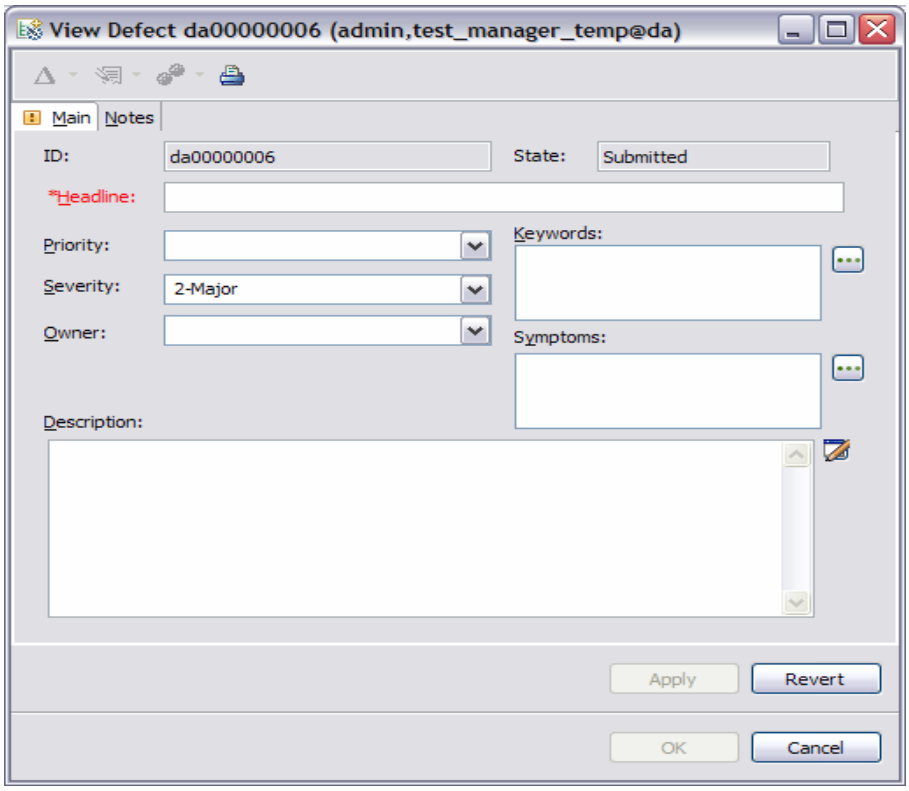
Computer Group



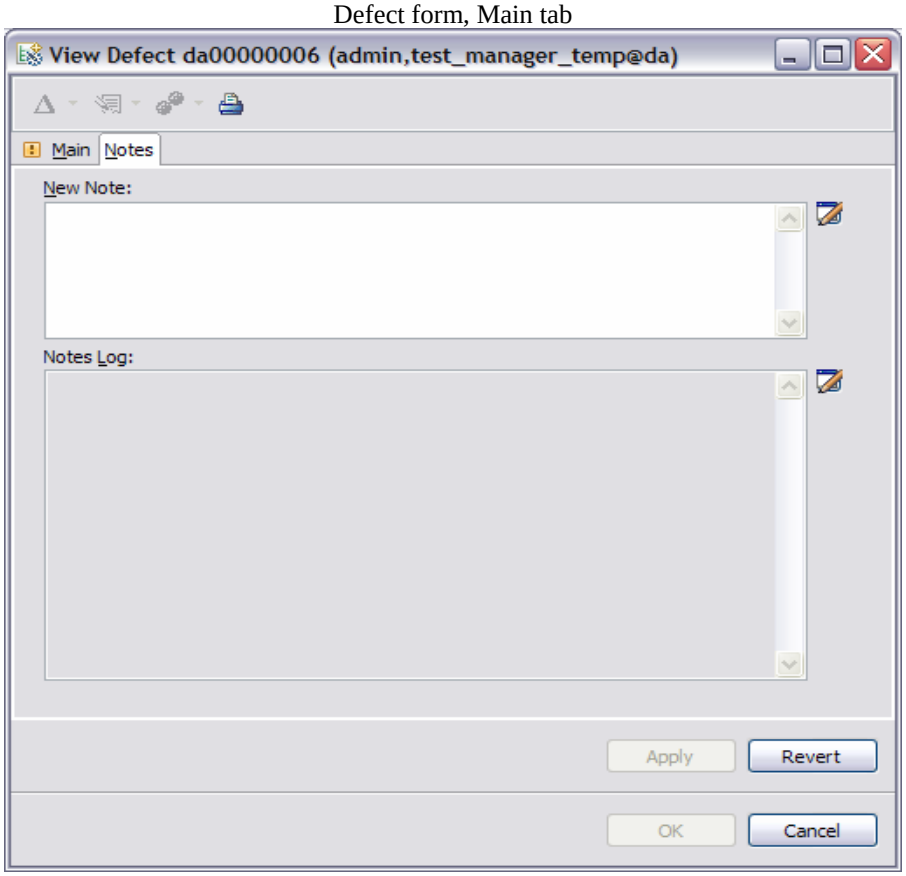
Computer Group tab

Defect management

Defect



Defect form, Main tab

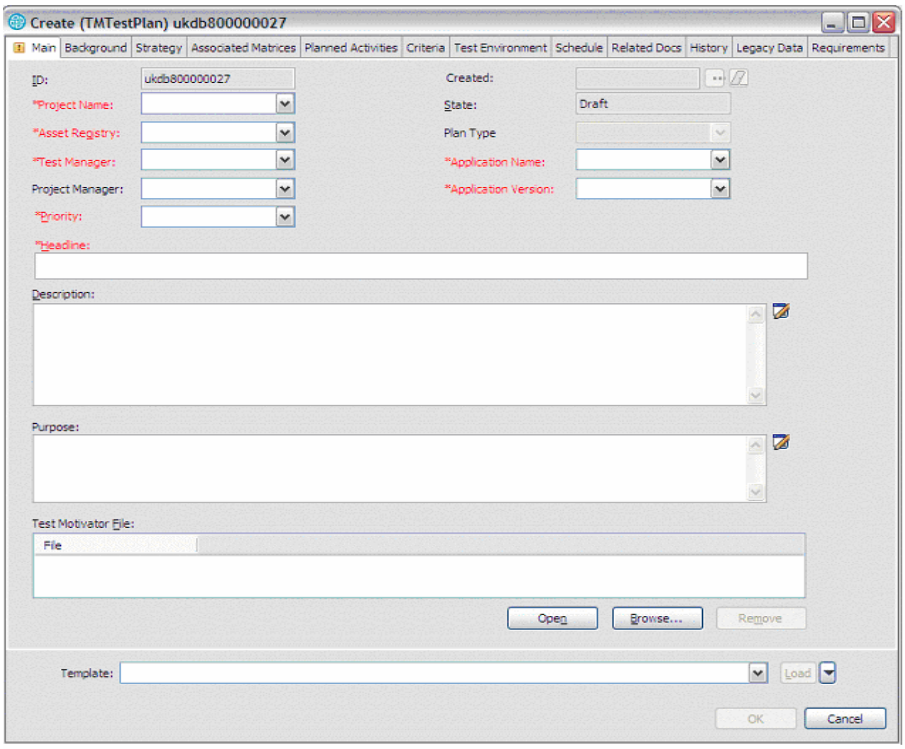


Defect form, Notes tab

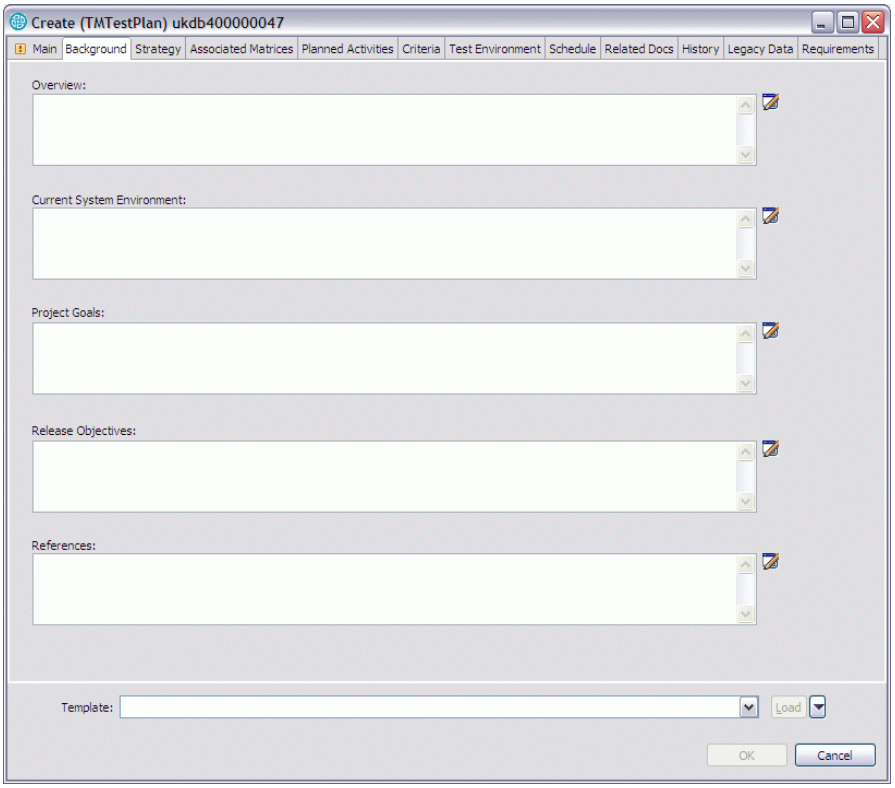
The main forms of CQ record types used in the processes for large projects

Test management

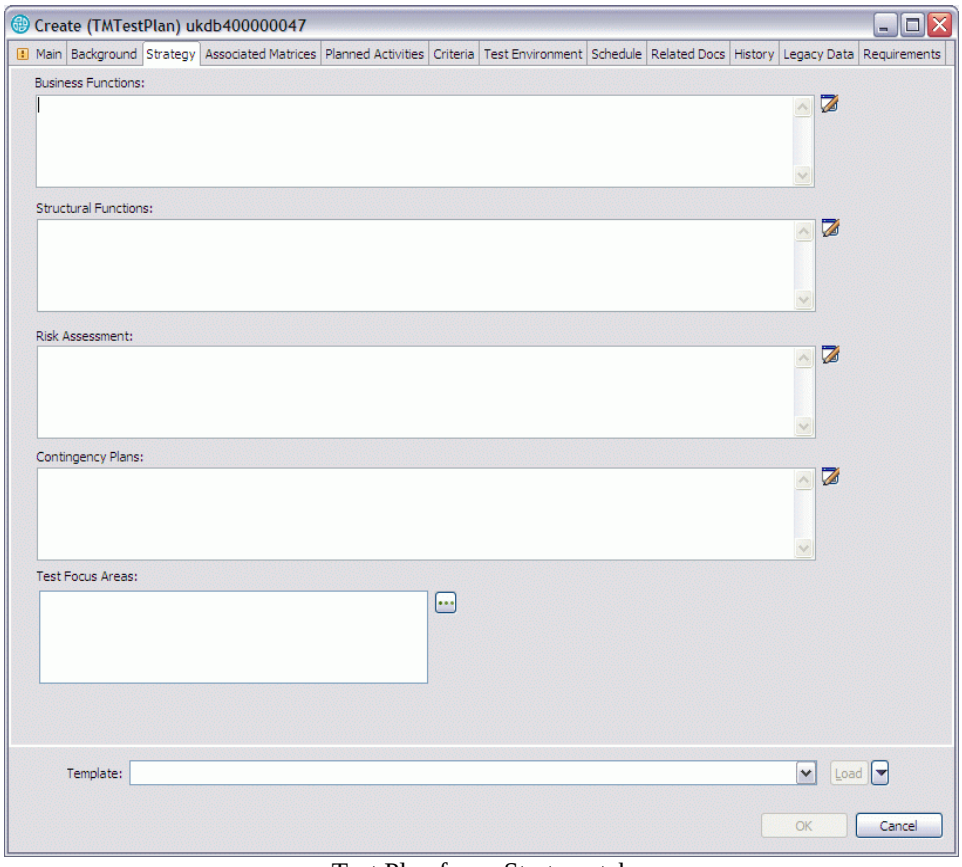
Test Plan



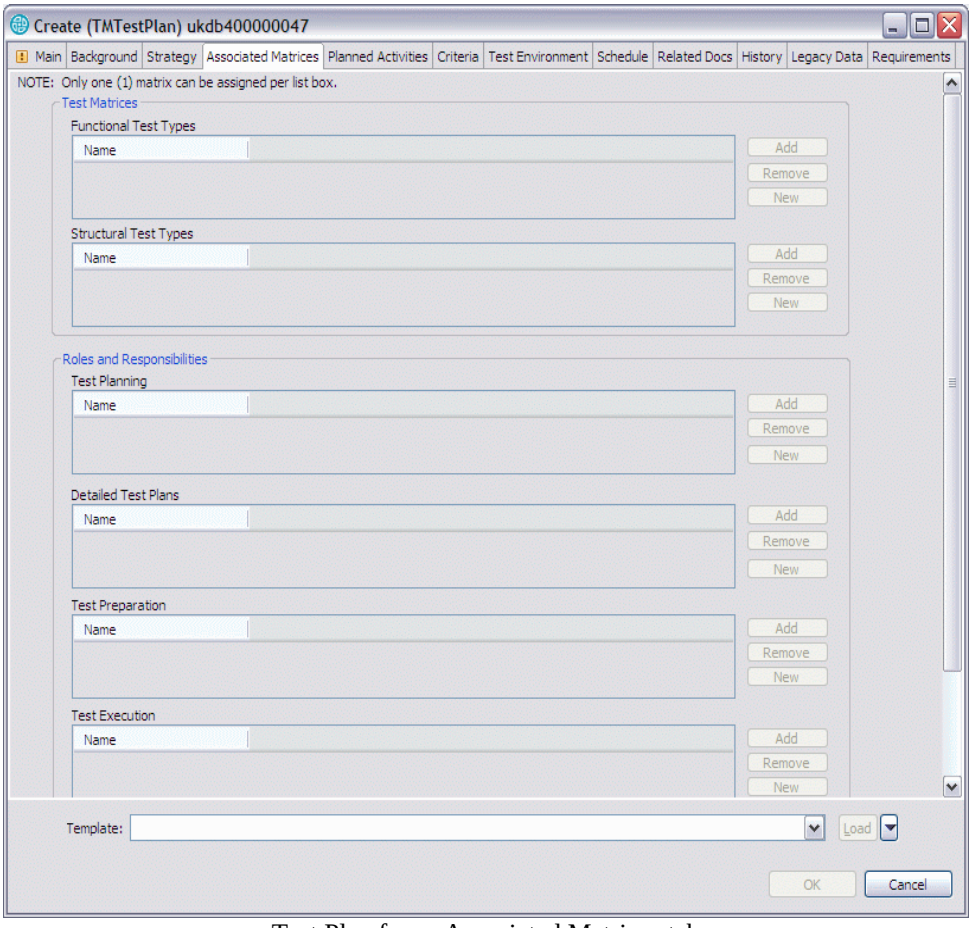
Test Plan form, Main tab



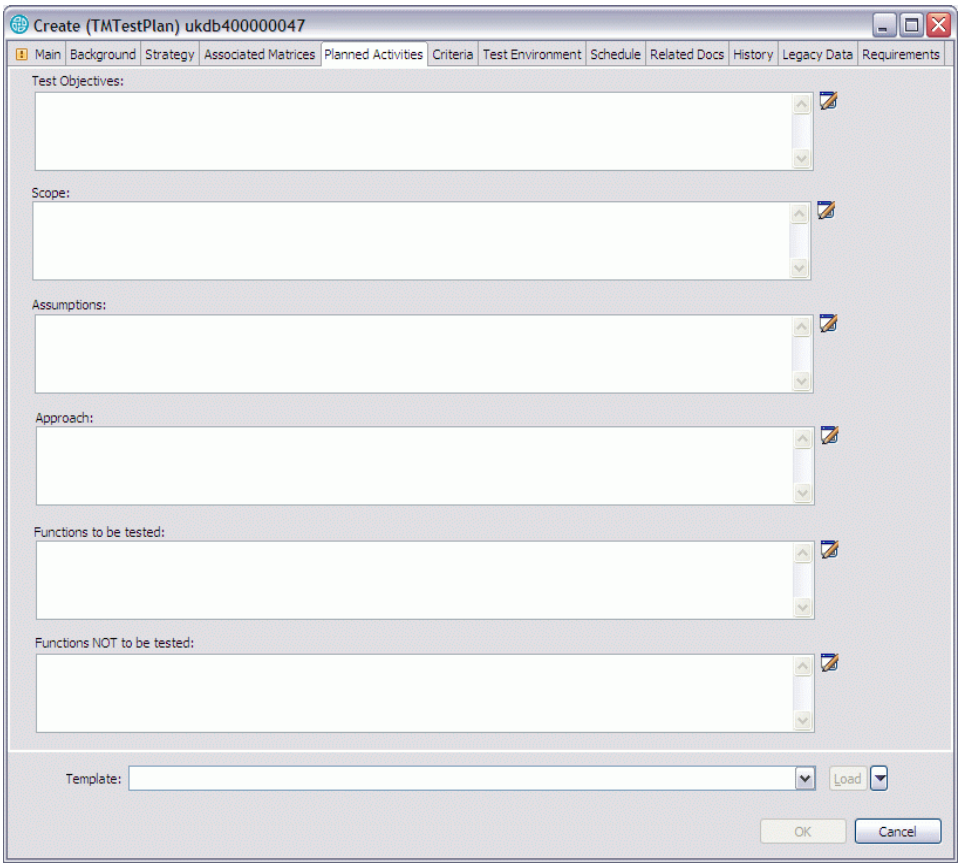
Test Plan form, Background tab



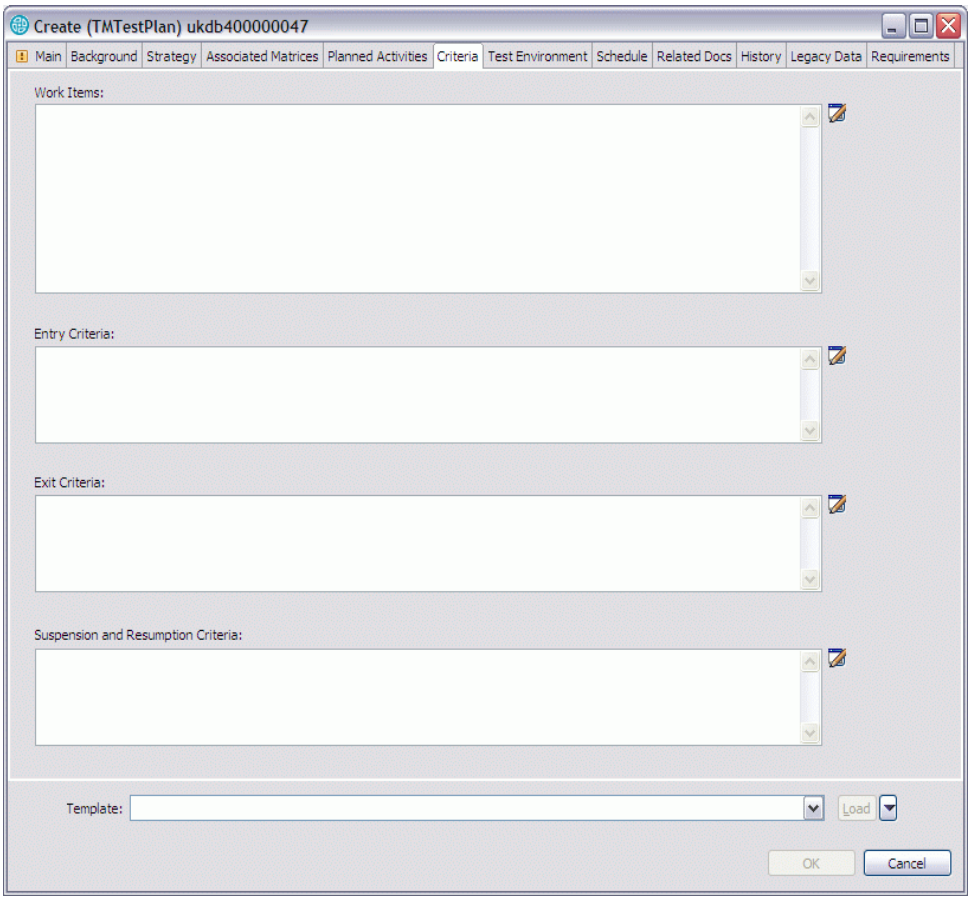
Test Plan form, Strategy tab



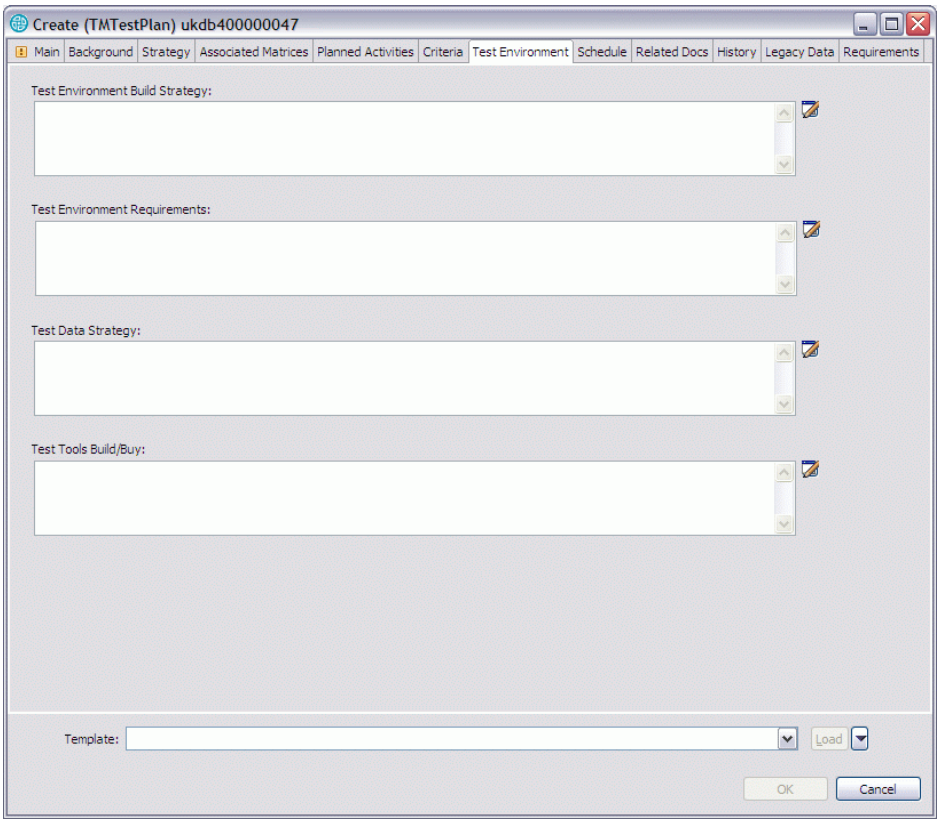
Test Plan form, Associated Matrices tab



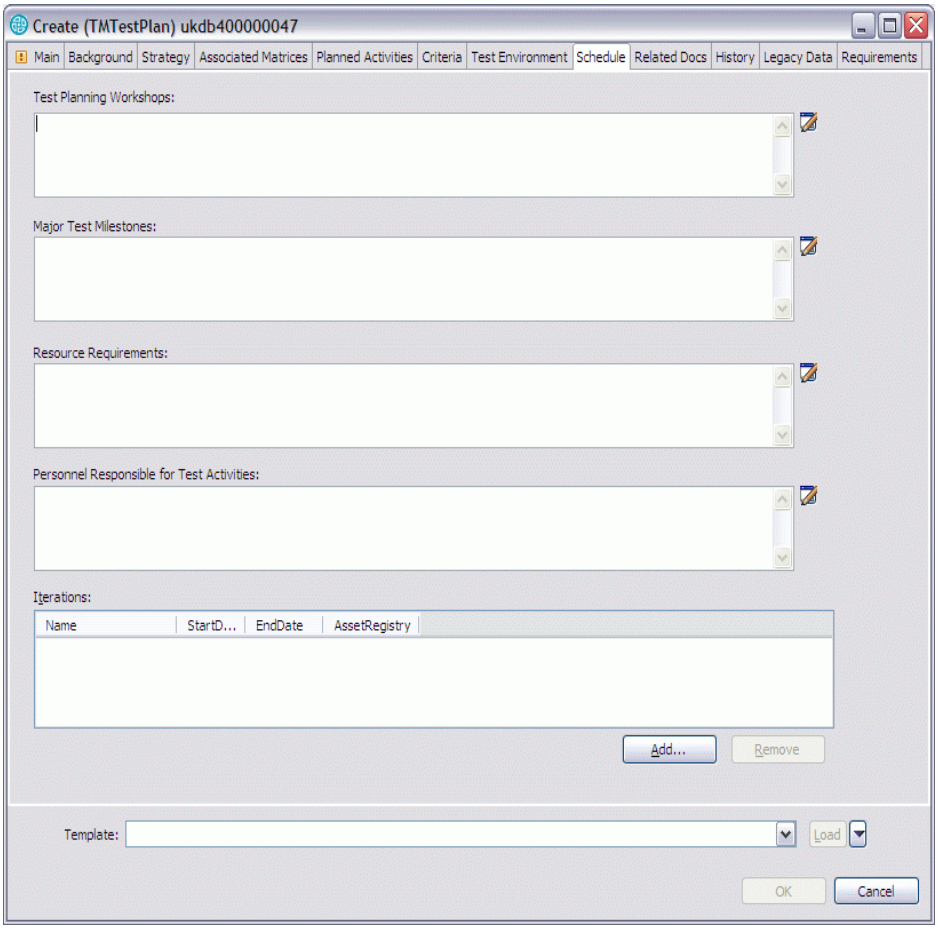
Test Plan form, Planned Activities tab



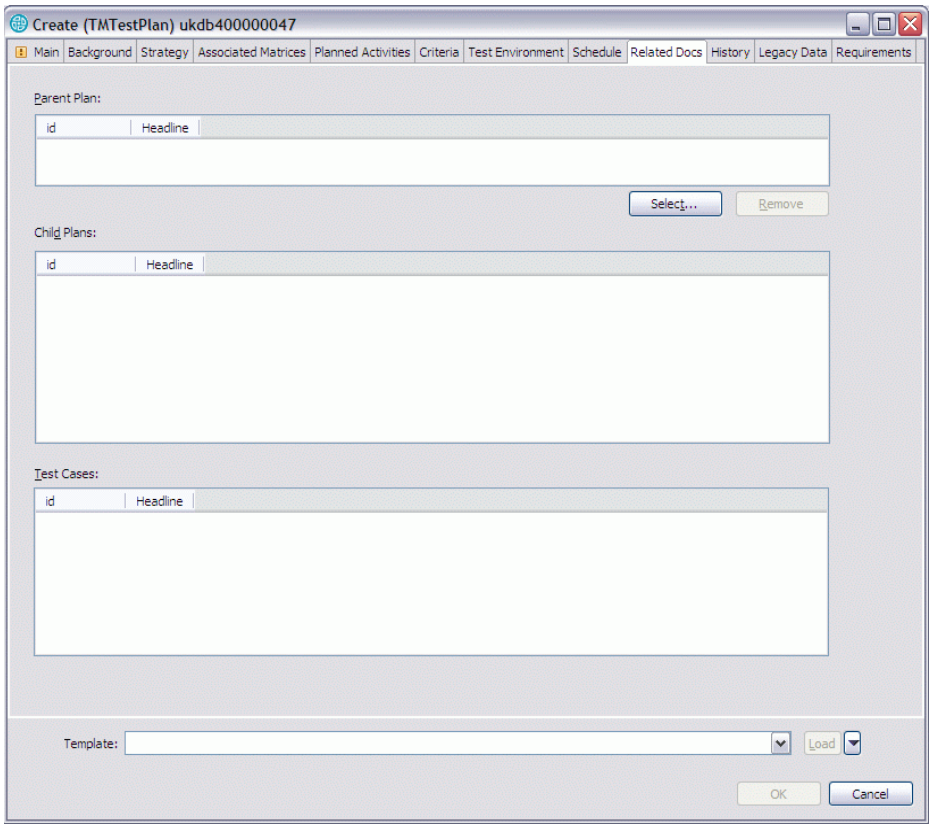
Test Plan form, Criteria tab



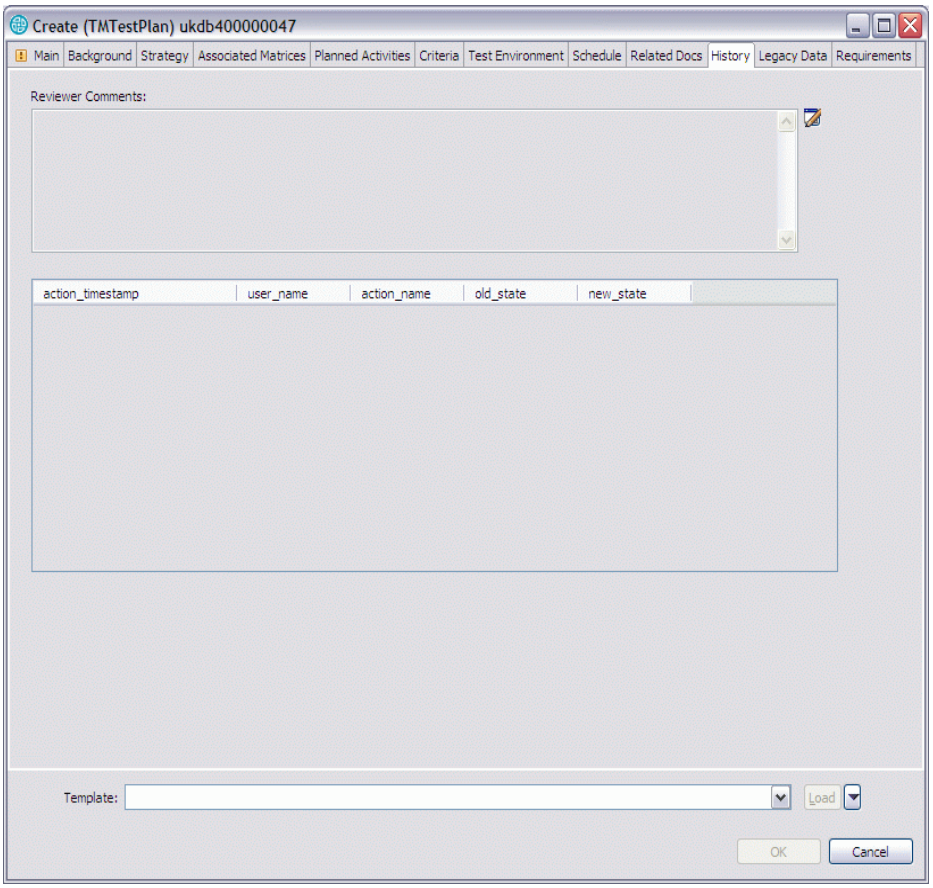
Test Plan form, Test Environment tab



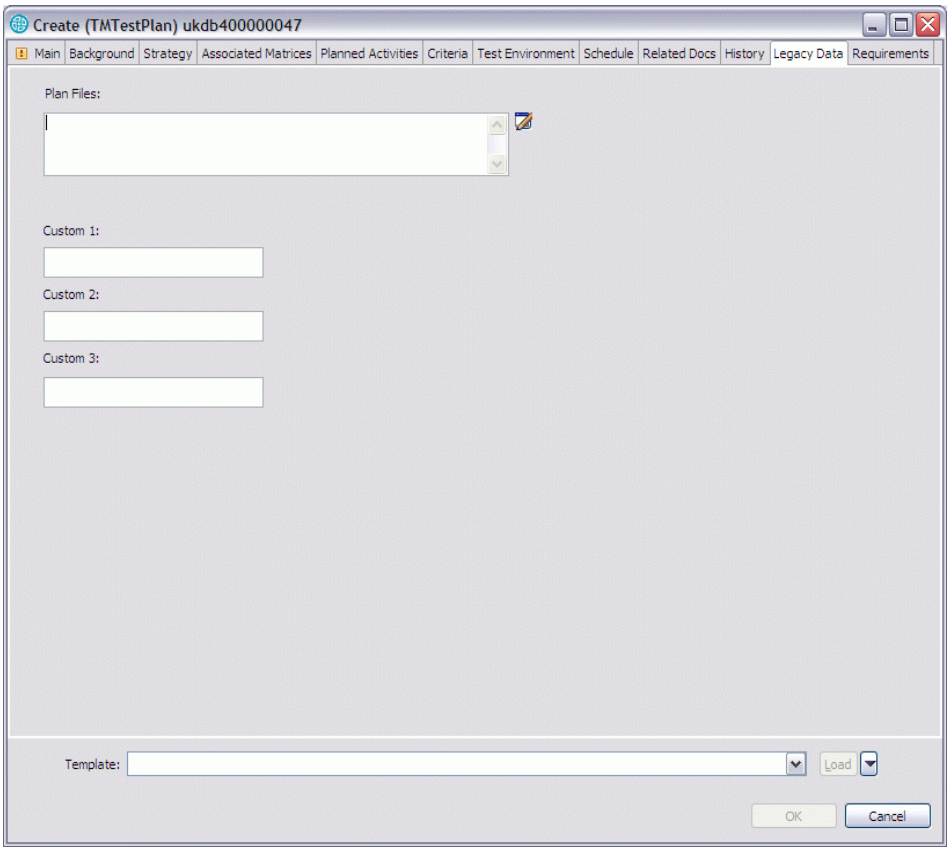
Test Plan form, Schedule tab



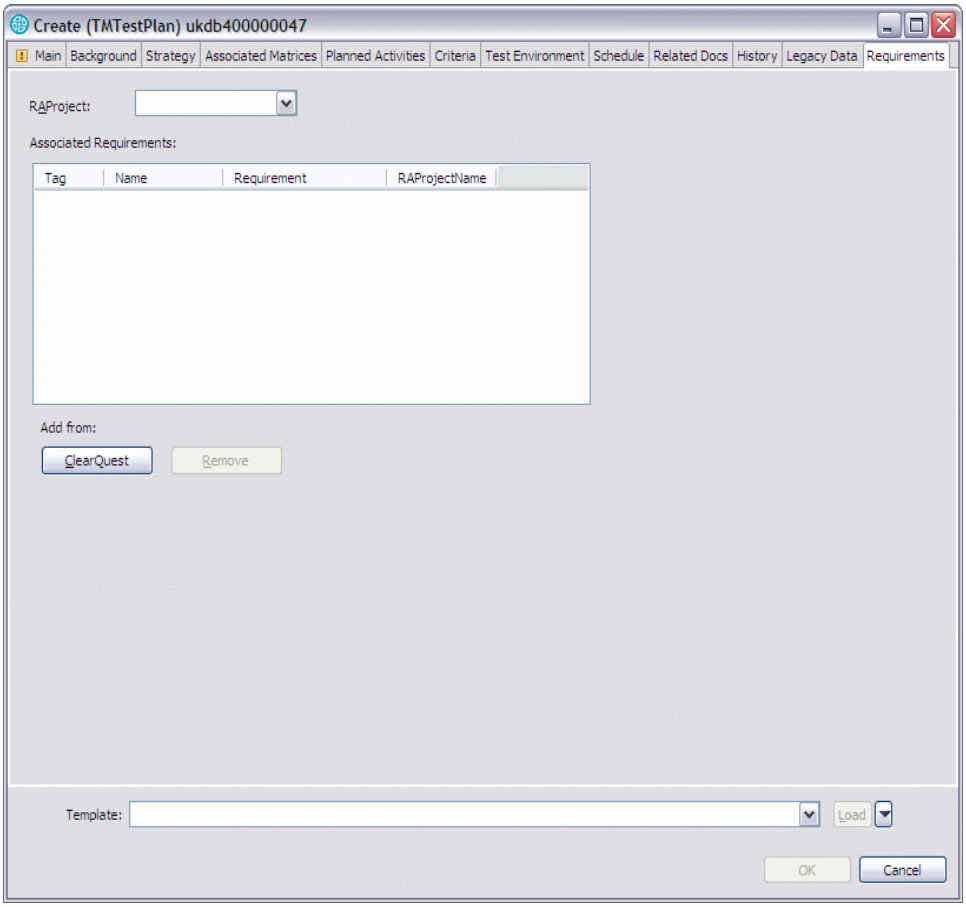
Test Plan form, Related Docs tab



Test Plan form, History tab

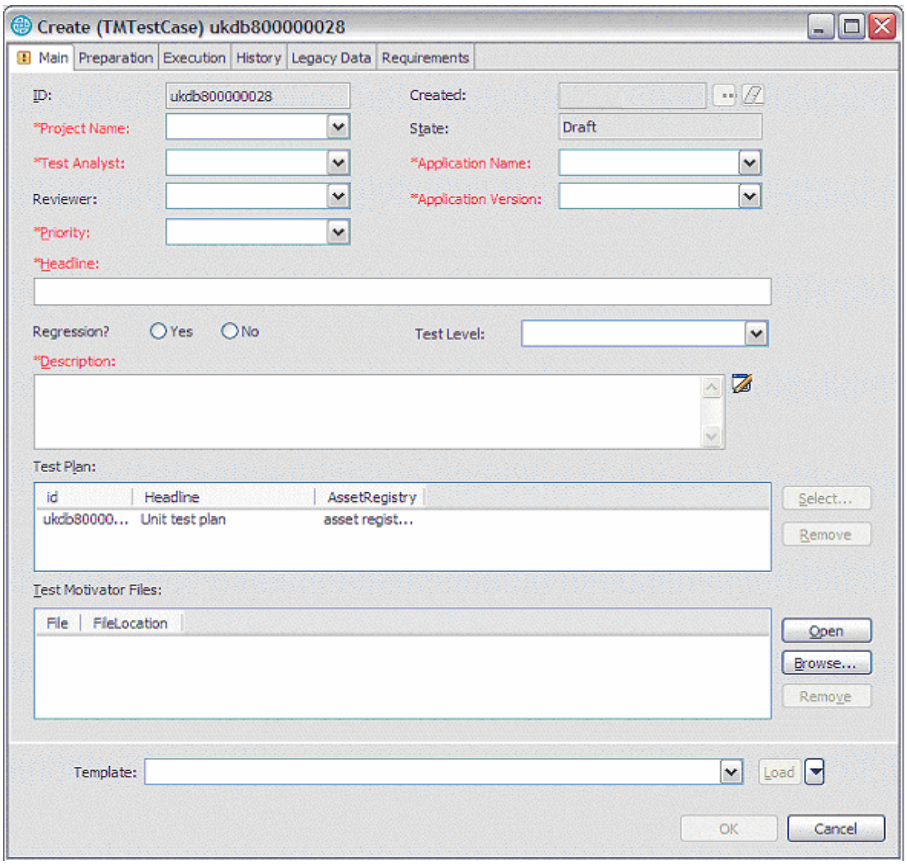


Test Plan form, Legacy Data tab

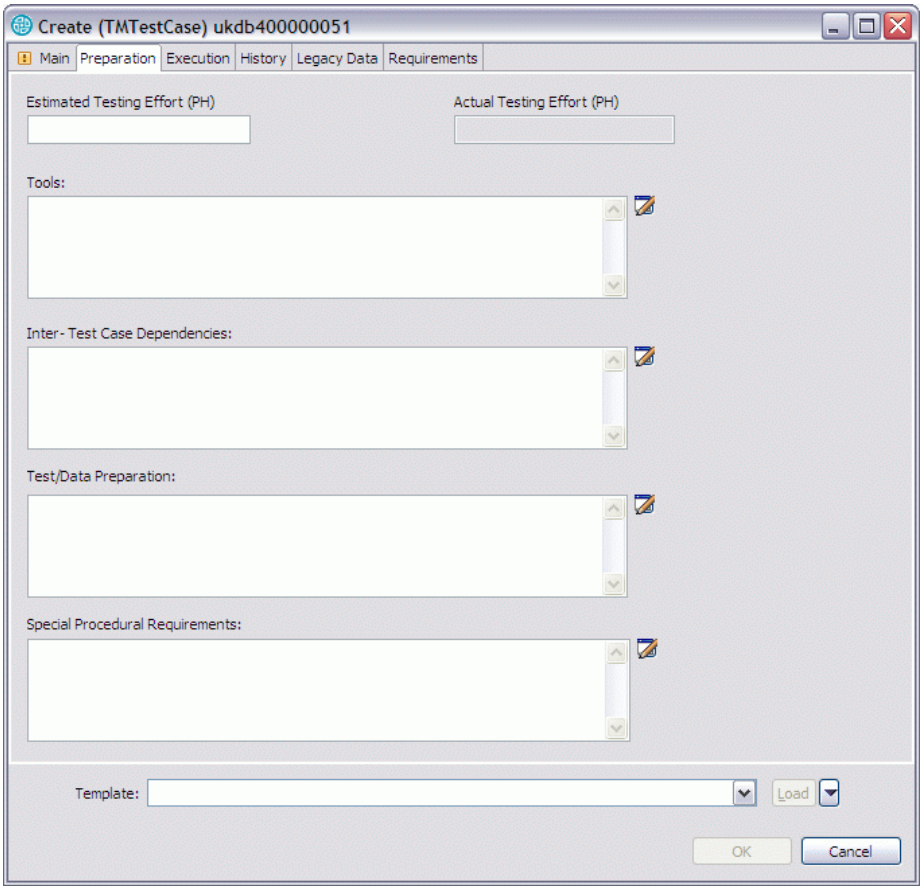


Test Plan form, Requirements tab

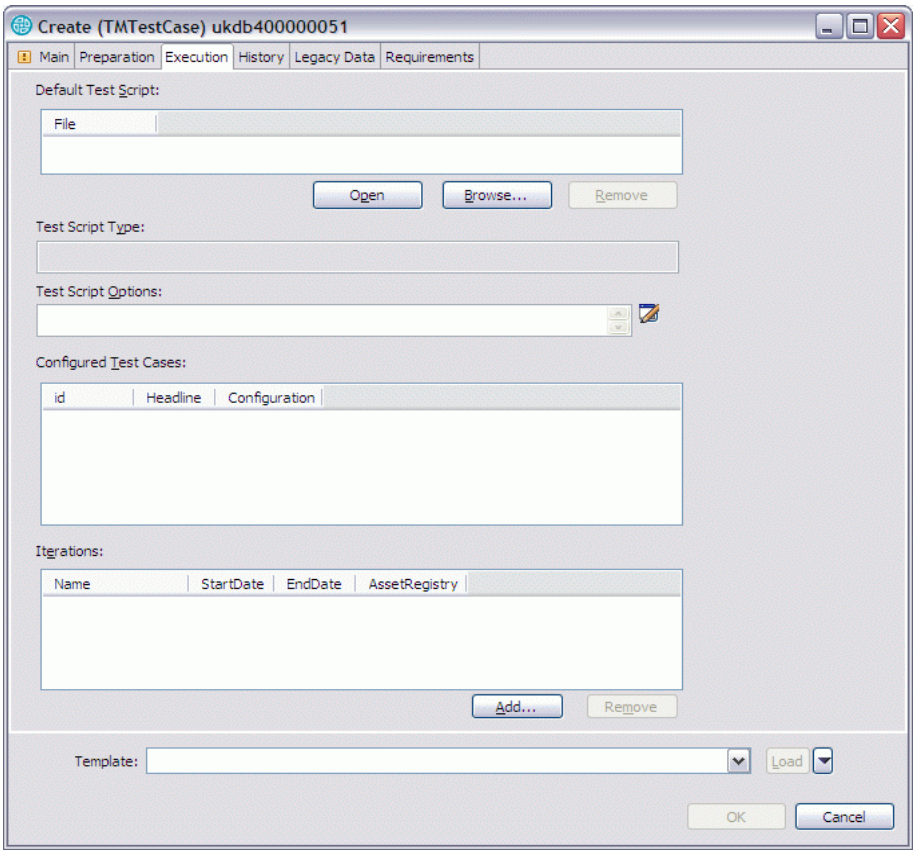
Test Case



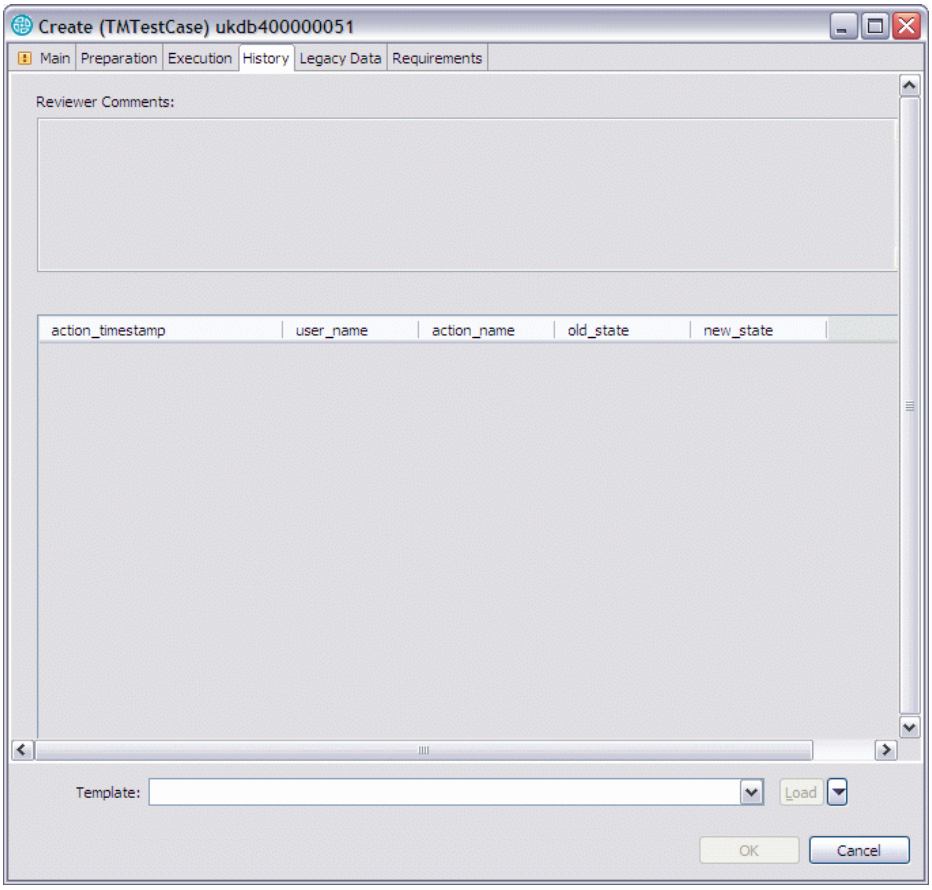
Test Case form, Main tab



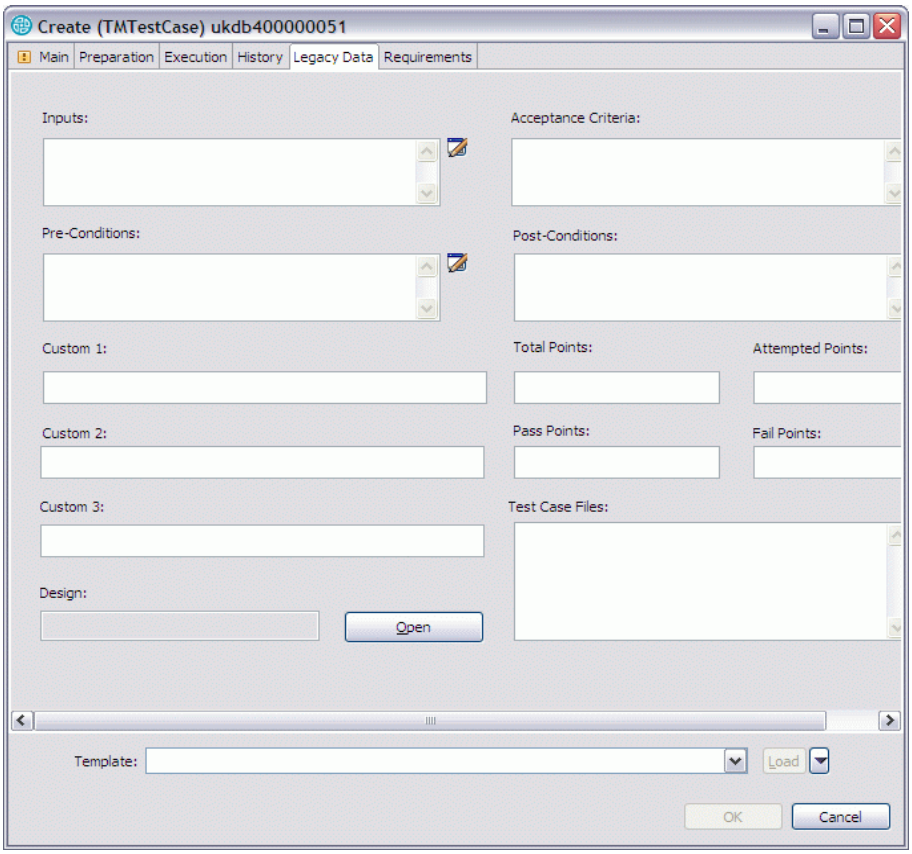
Test Case form, Preparation tab



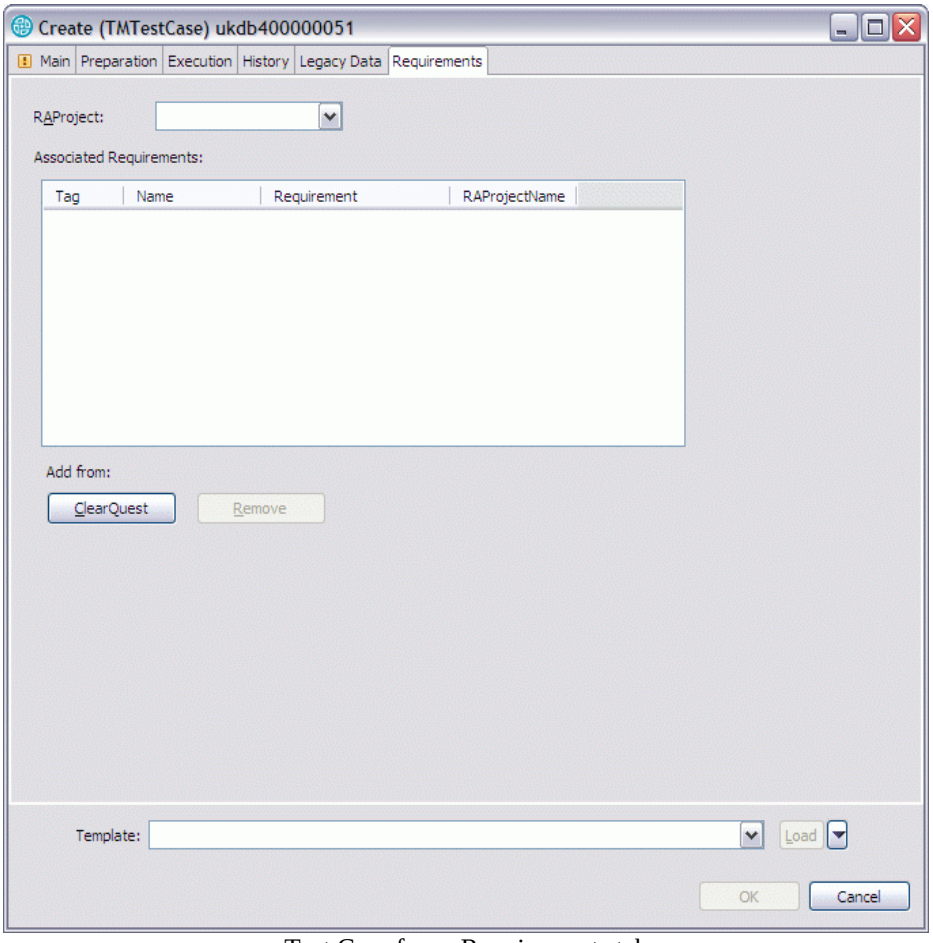
Test Case form, Execution tab



Test Case form, History tab

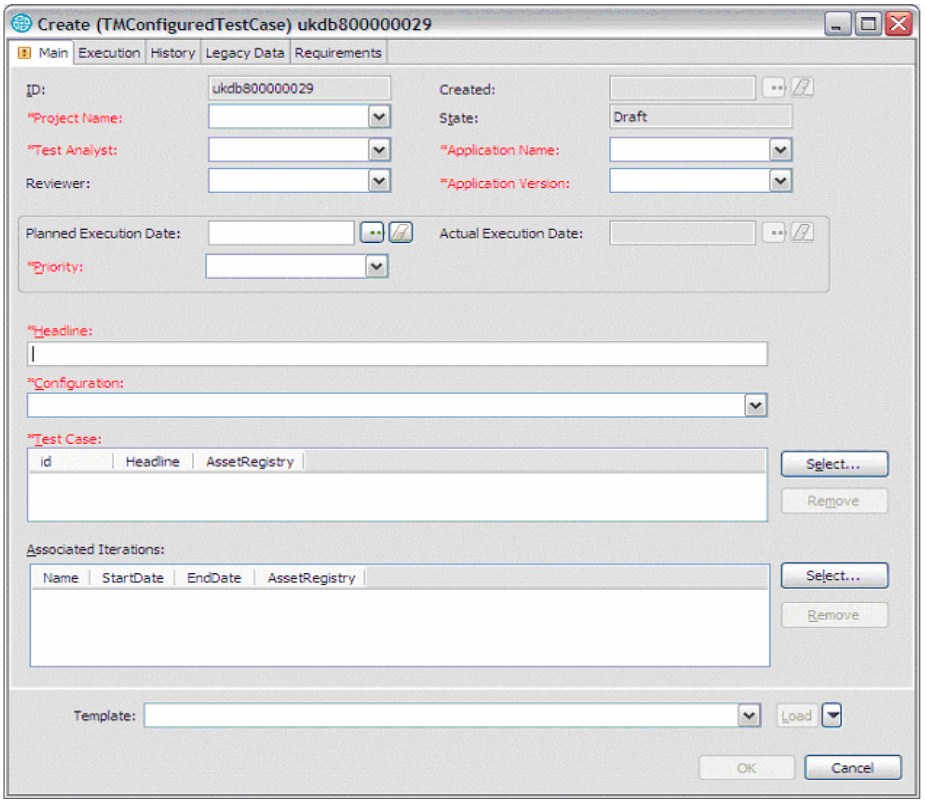


Test Case form, Legacy Data tab

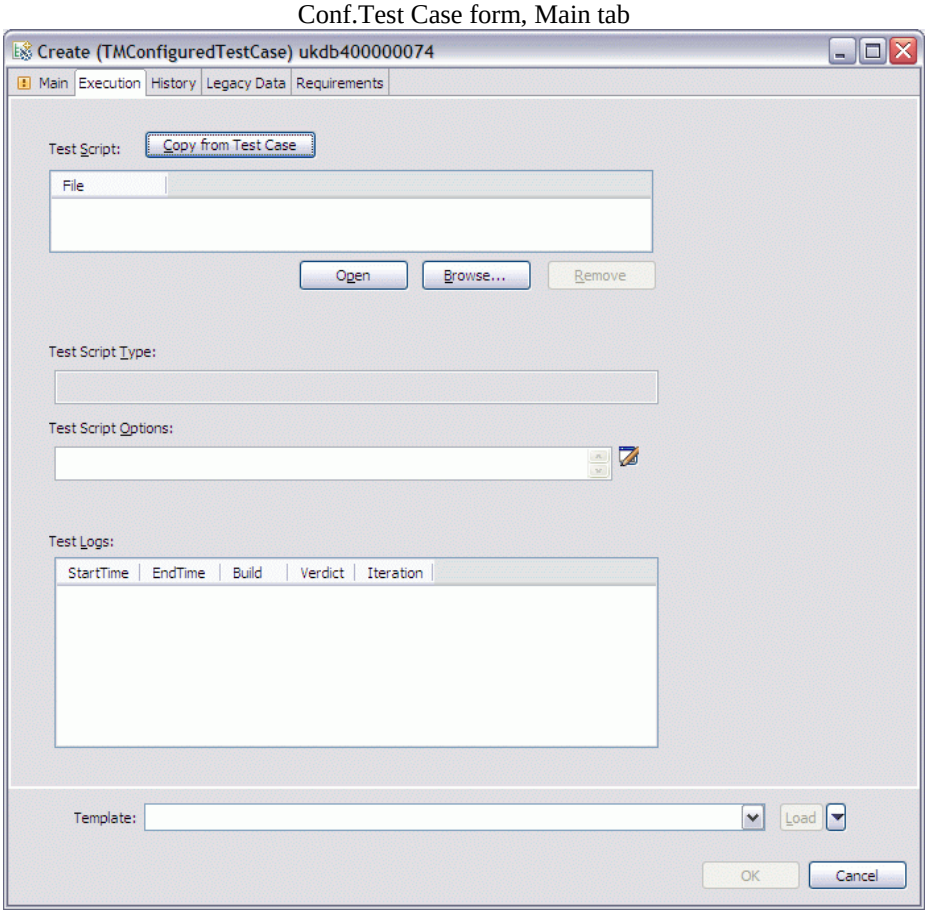


Test Case form, Requirements tab

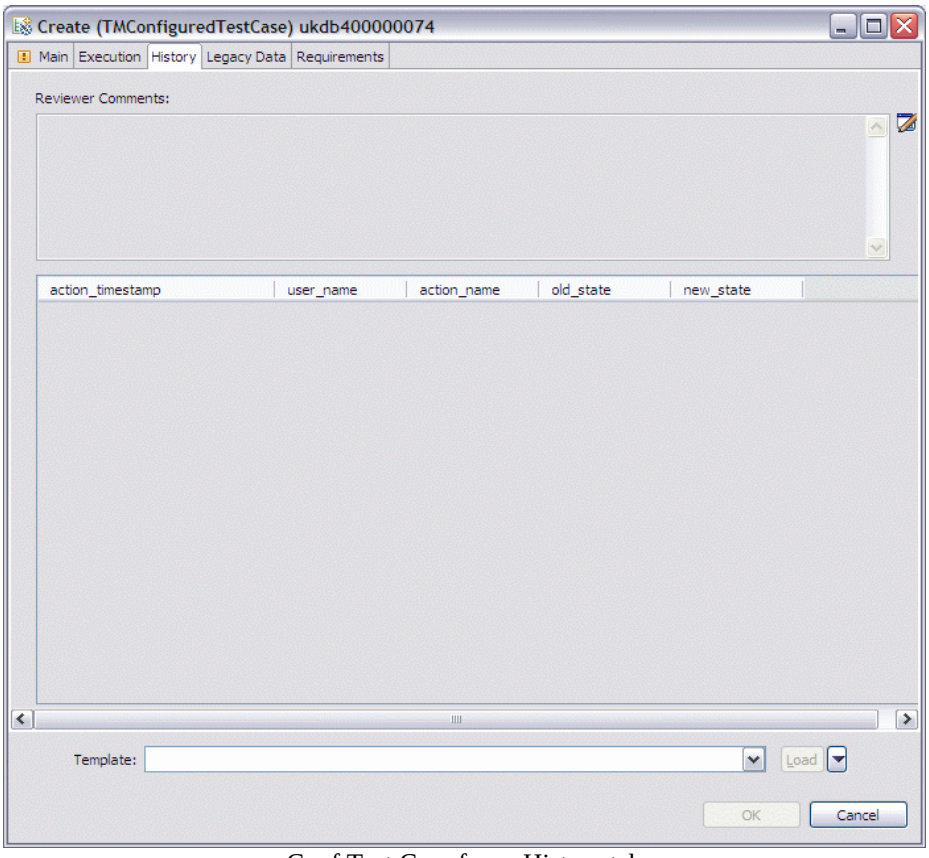
Configured Test Case



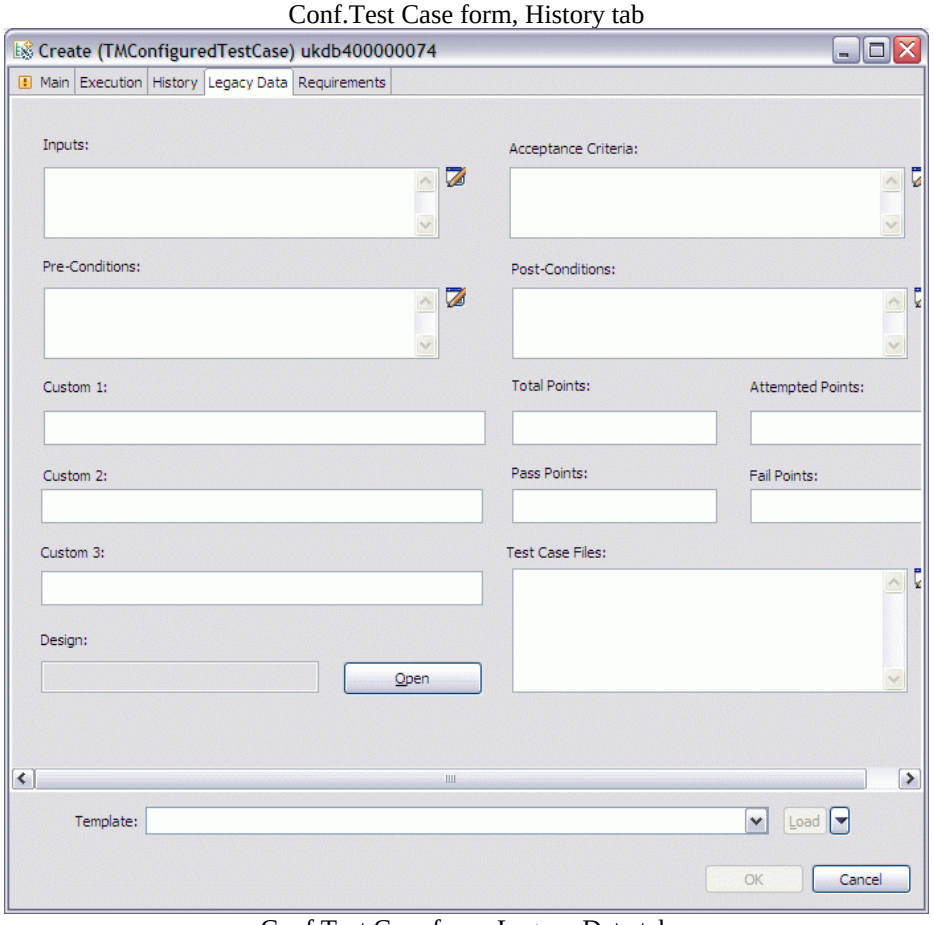
Conf.Test Case form, Main tab



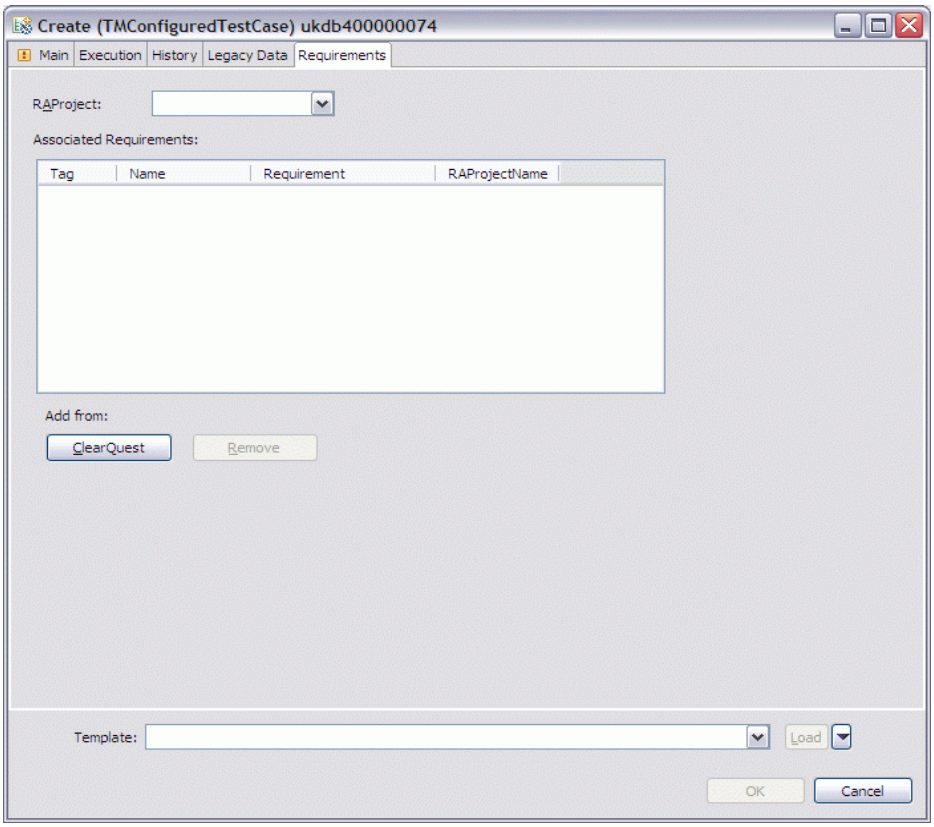
Conf.Test Case form, Execution tab



Conf.Test Case form, History tab

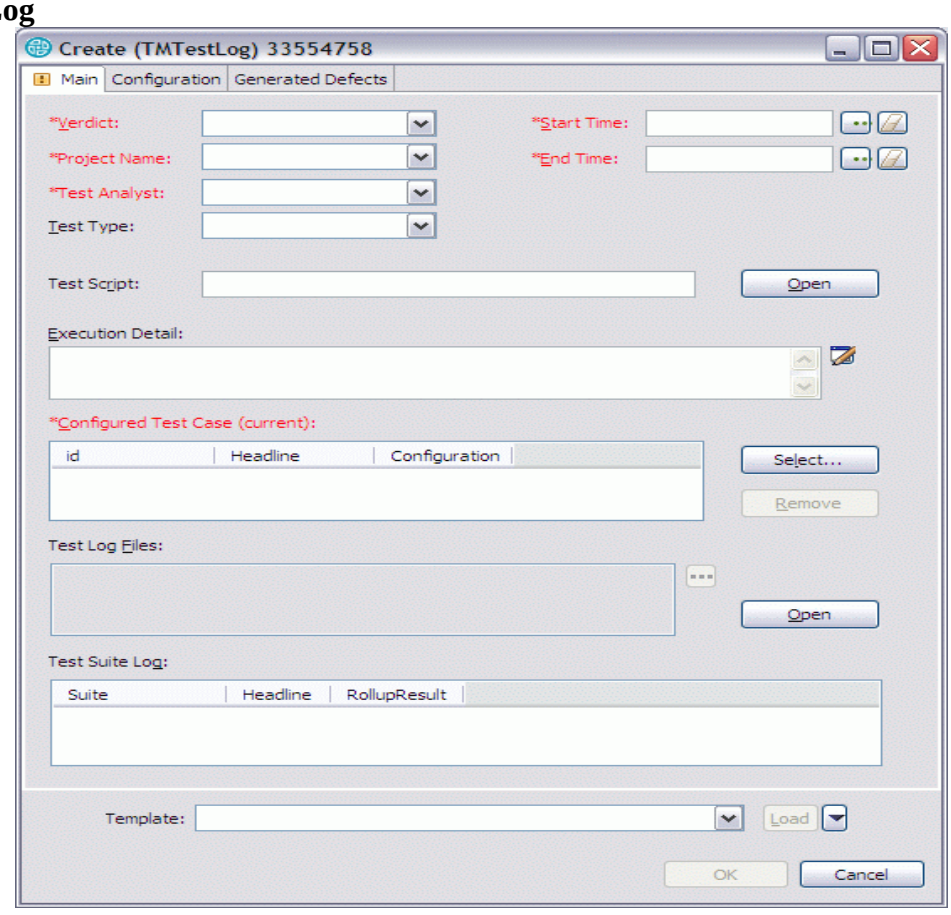


Conf.Test Case form, Legacy Data tab

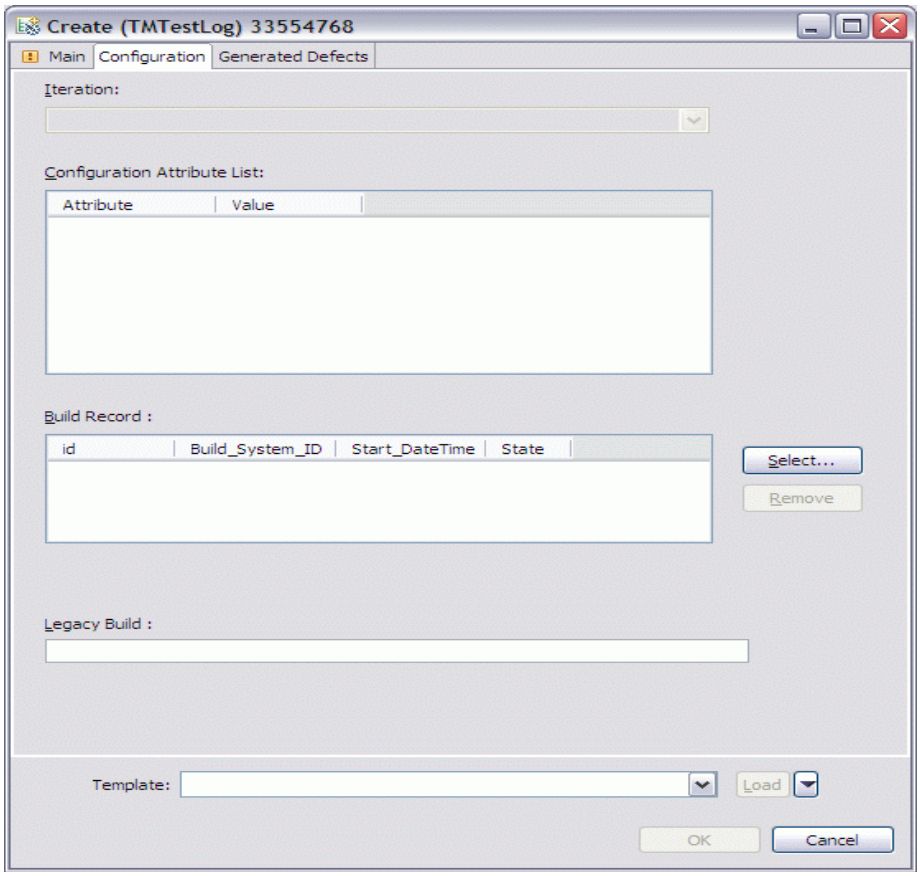


Conf.Test Case form, Requirements tab

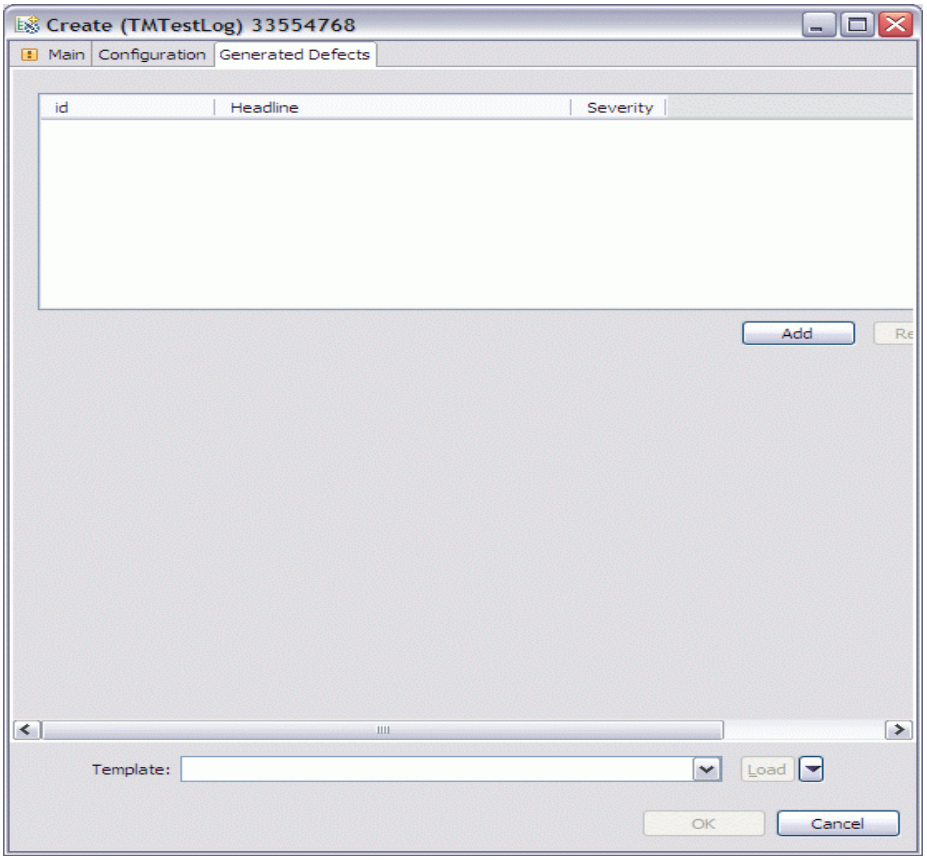
Test Log



Test Log form, Main tab

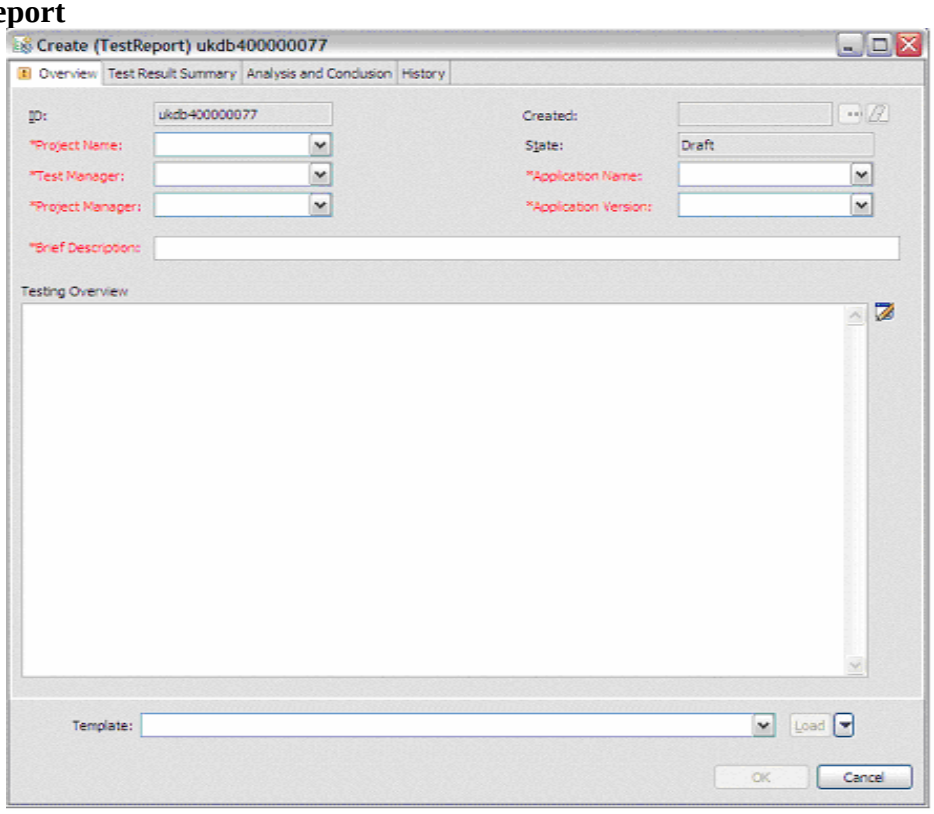


Test Log form, Configuration tab

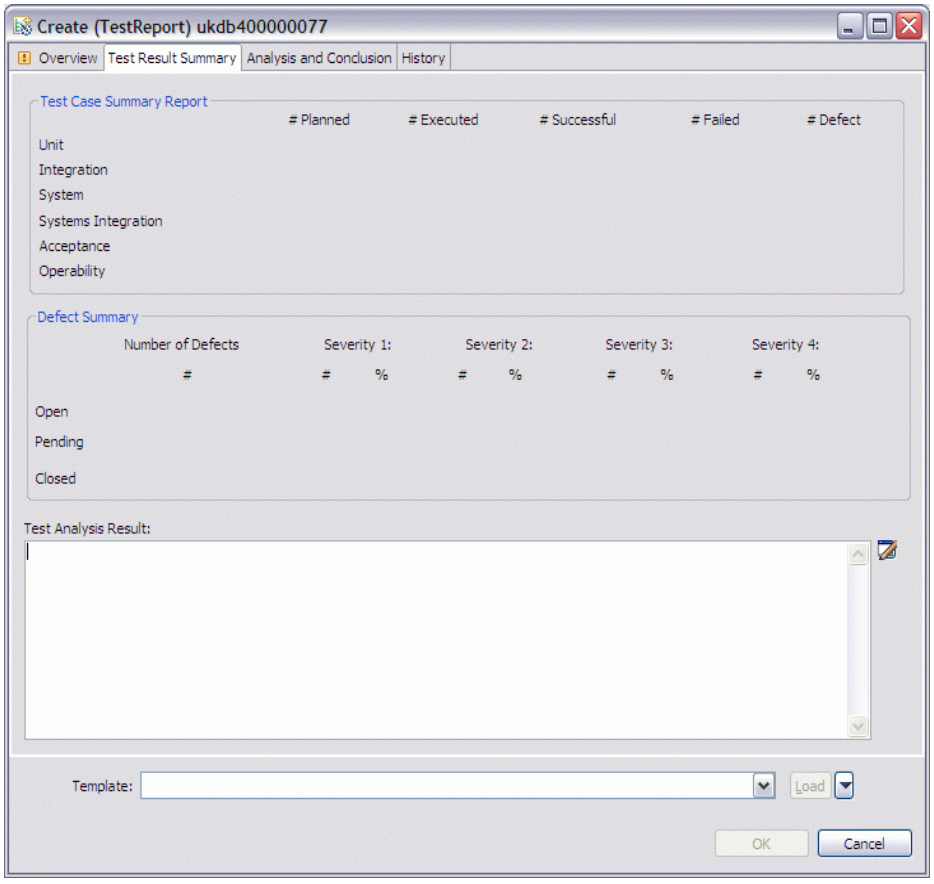


Test Log form, Generated Defects tab

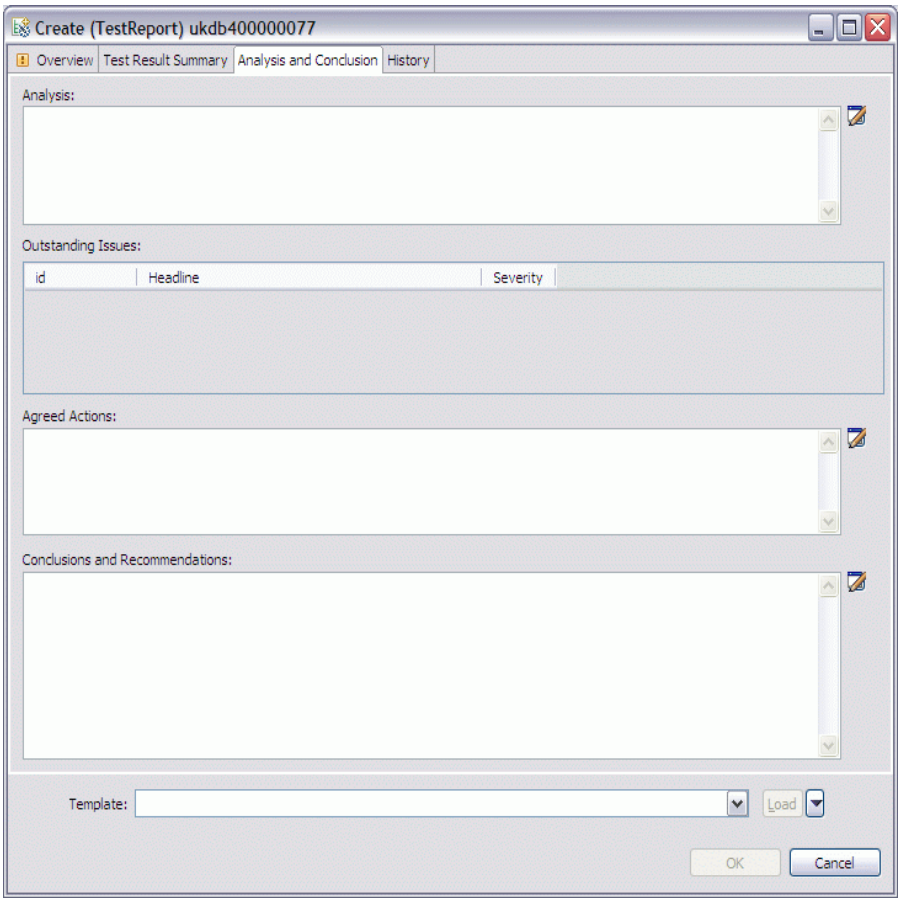
Test Report



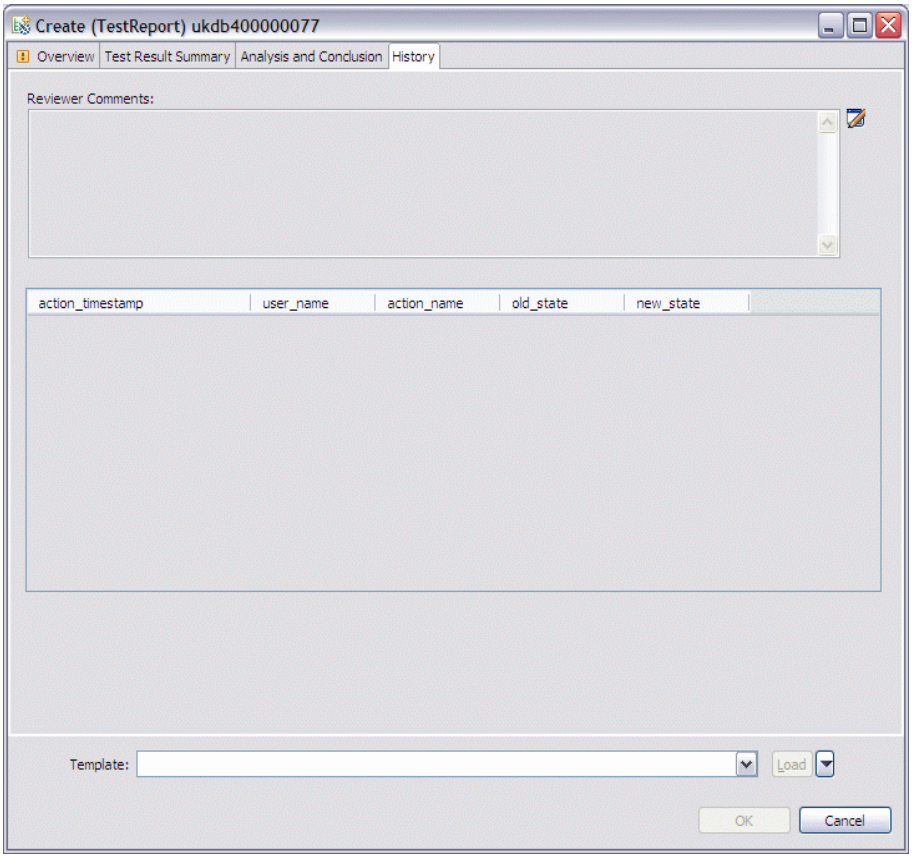
Test Report form, Overview tab



Test Report form, Test Result Summary tab

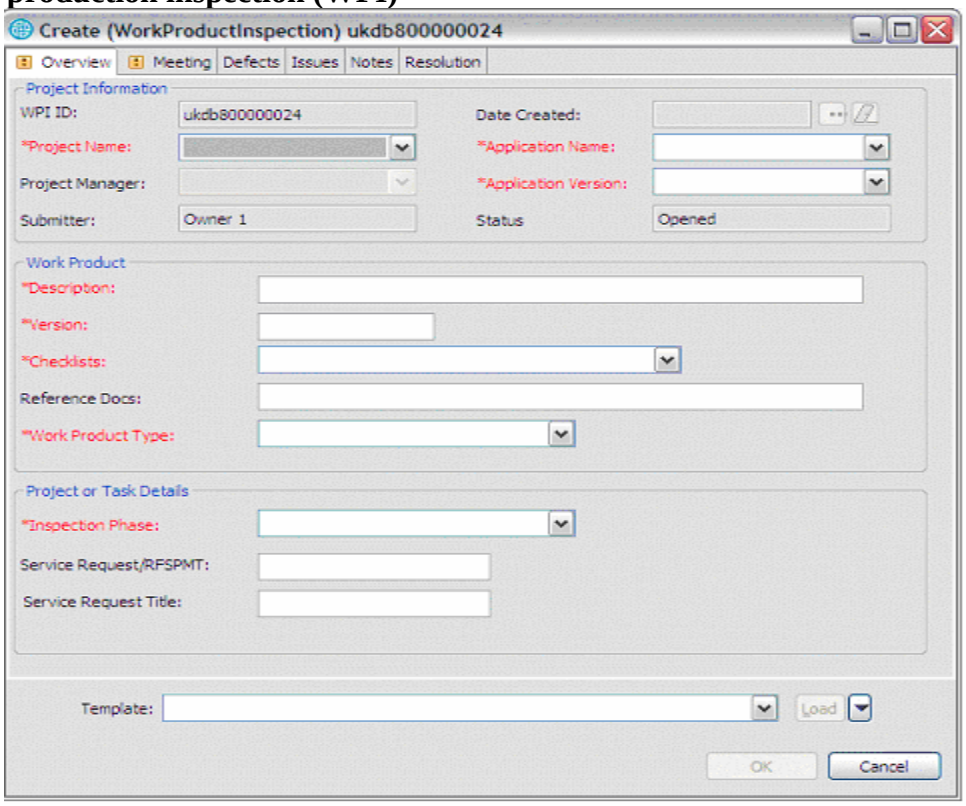


Test Report form, Analysis and Conclusion tab

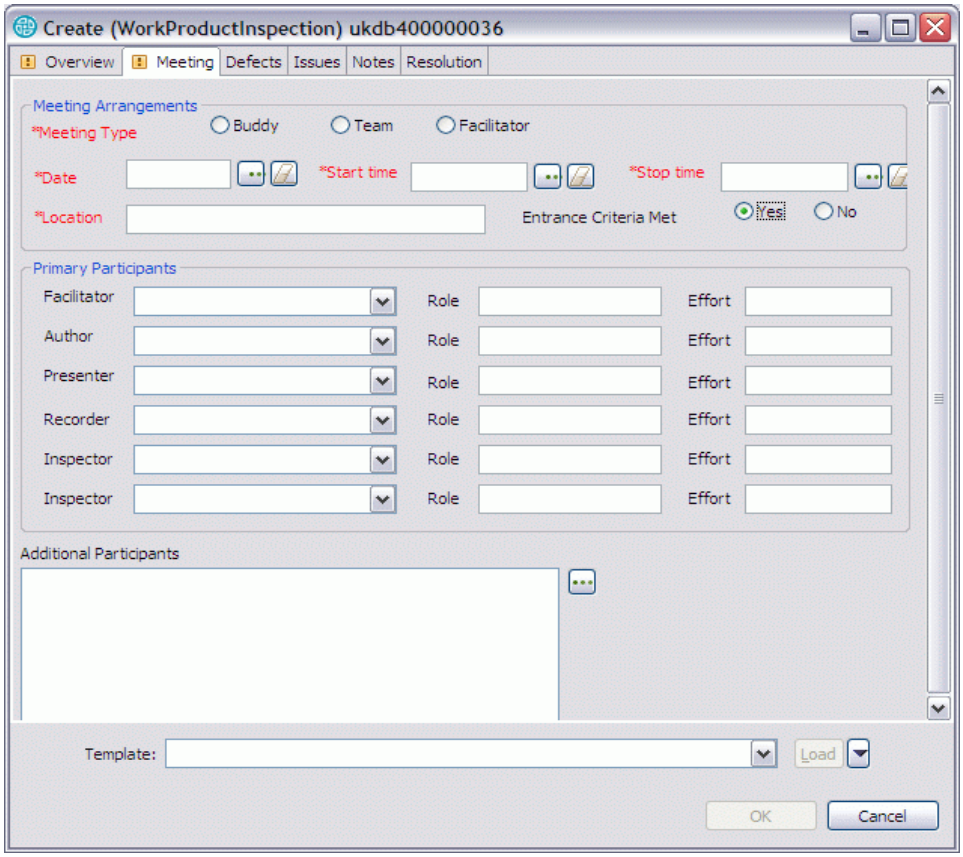


Test Report form, History tab

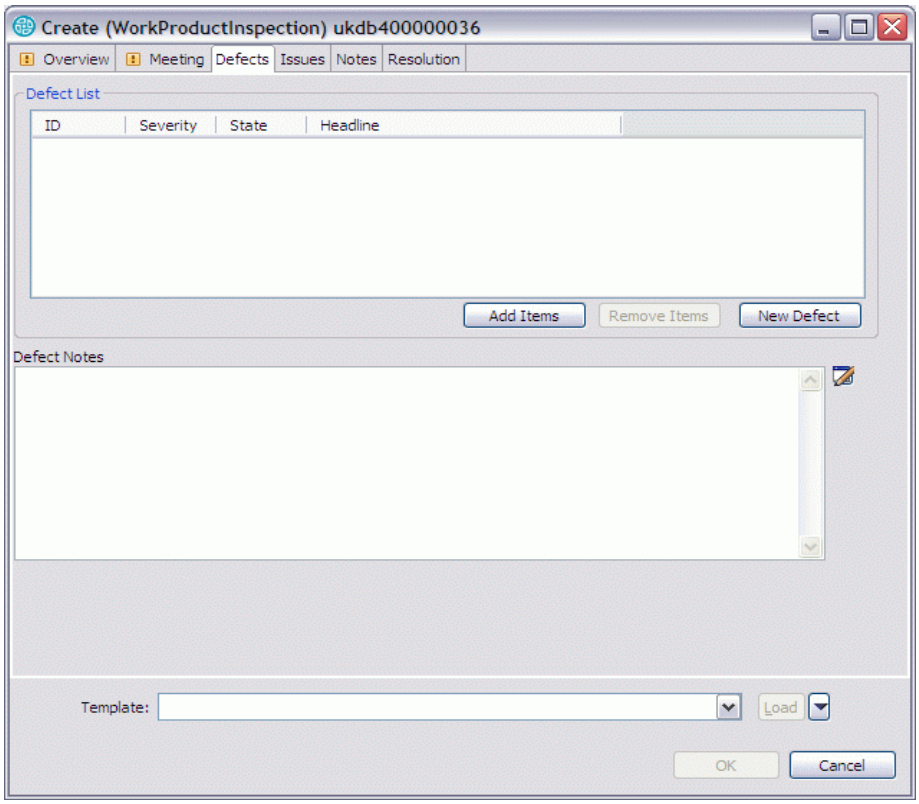
Work production inspection (WPI)



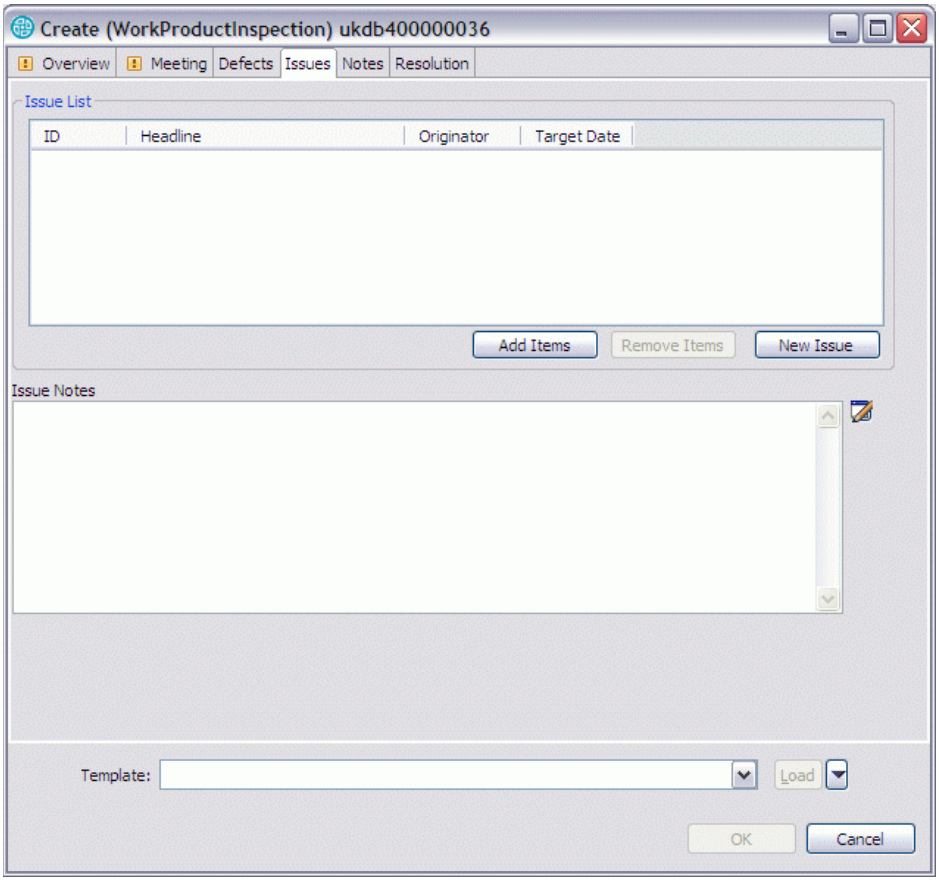
WPI form, Overview tab



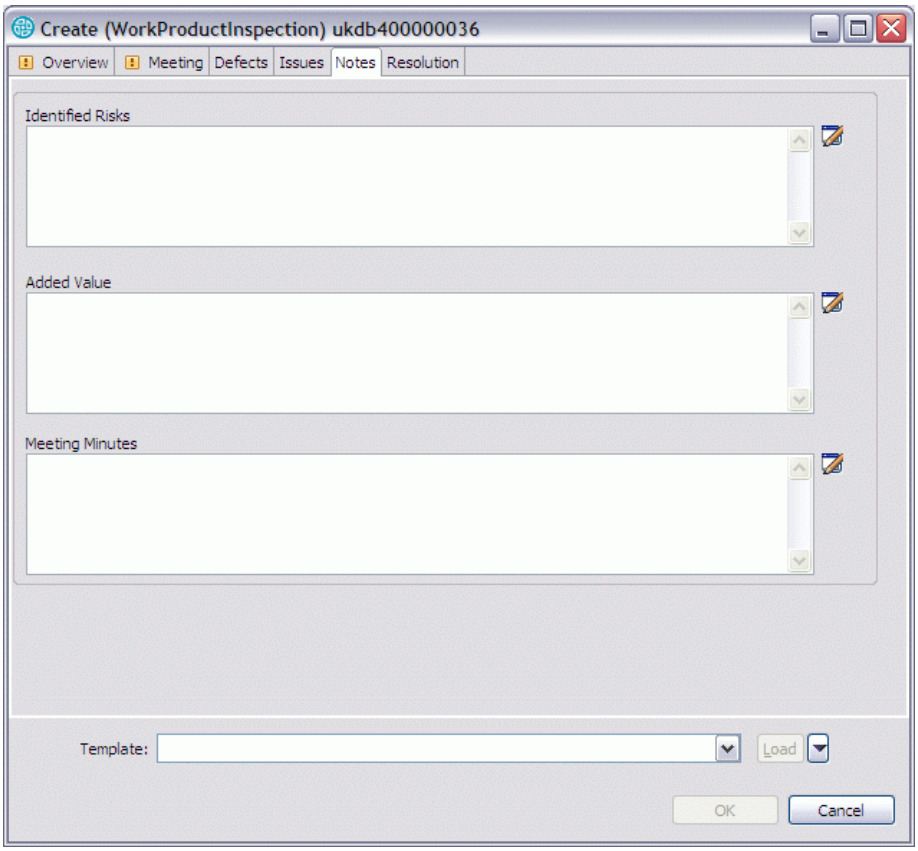
WPI form, Meeting tab



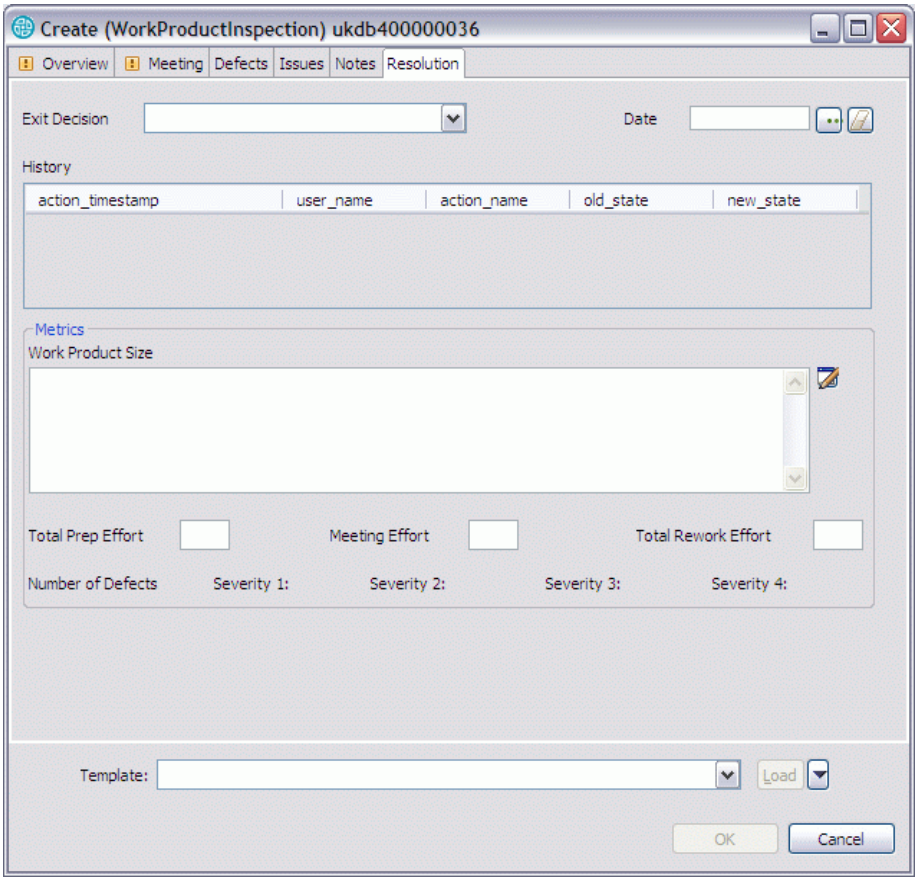
WPI form, Defects tab



WPI form, Issues tab

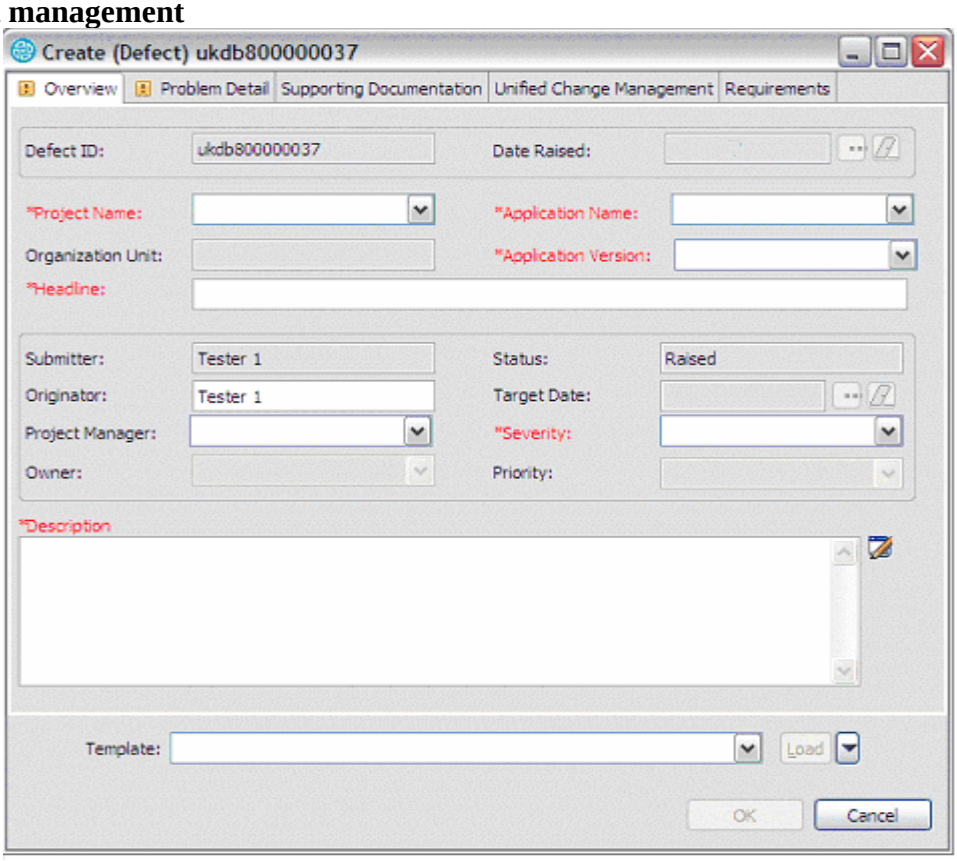


WPI form, Notes tab

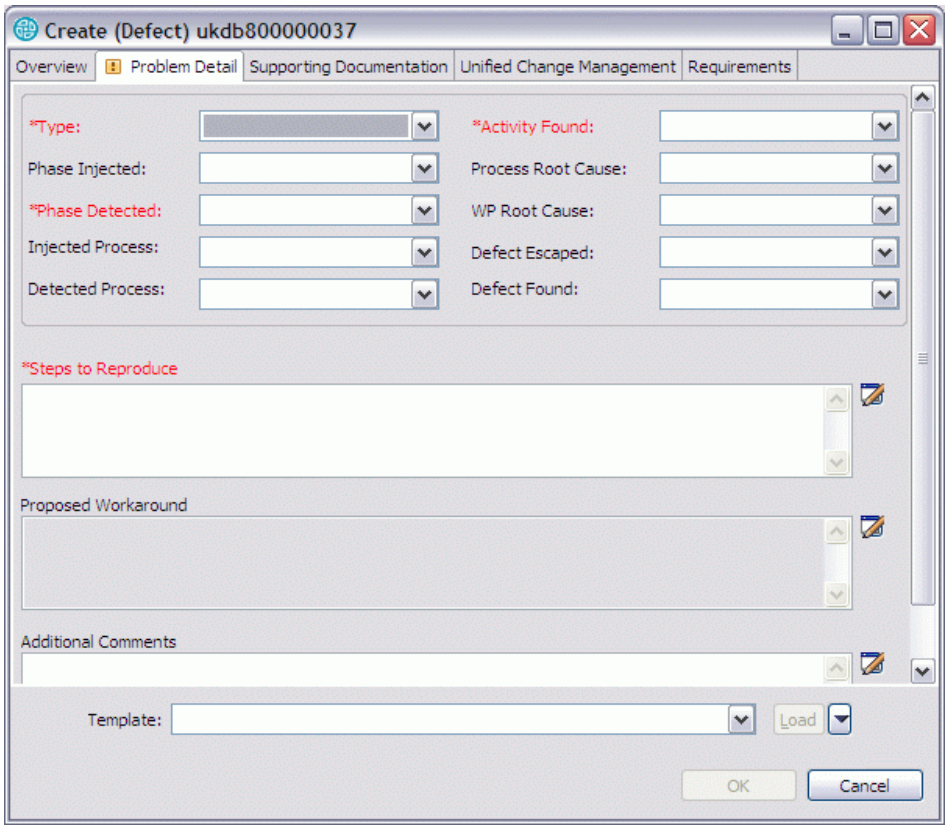


WPI form, Resolution tab

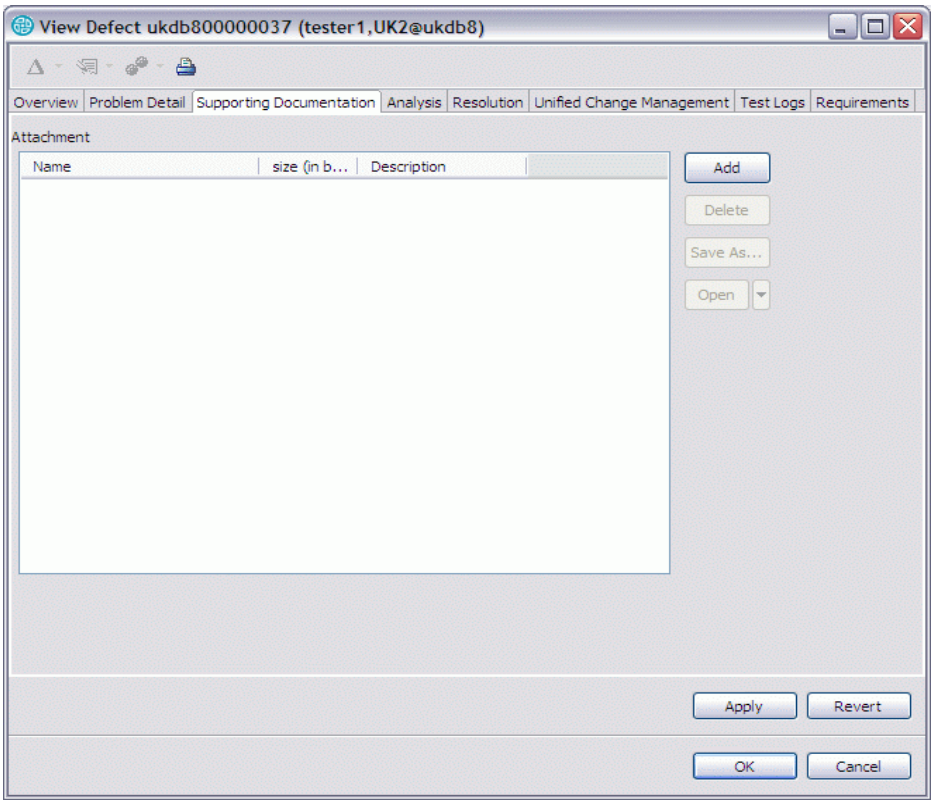
Defect management



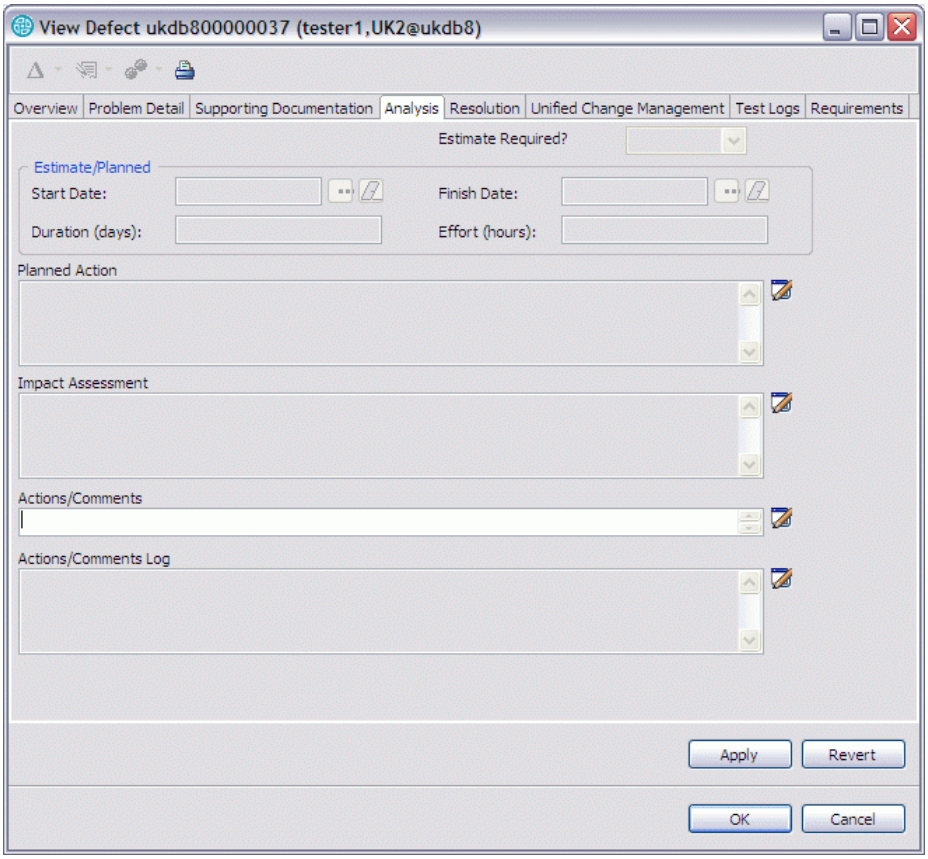
Defect form, Overview tab



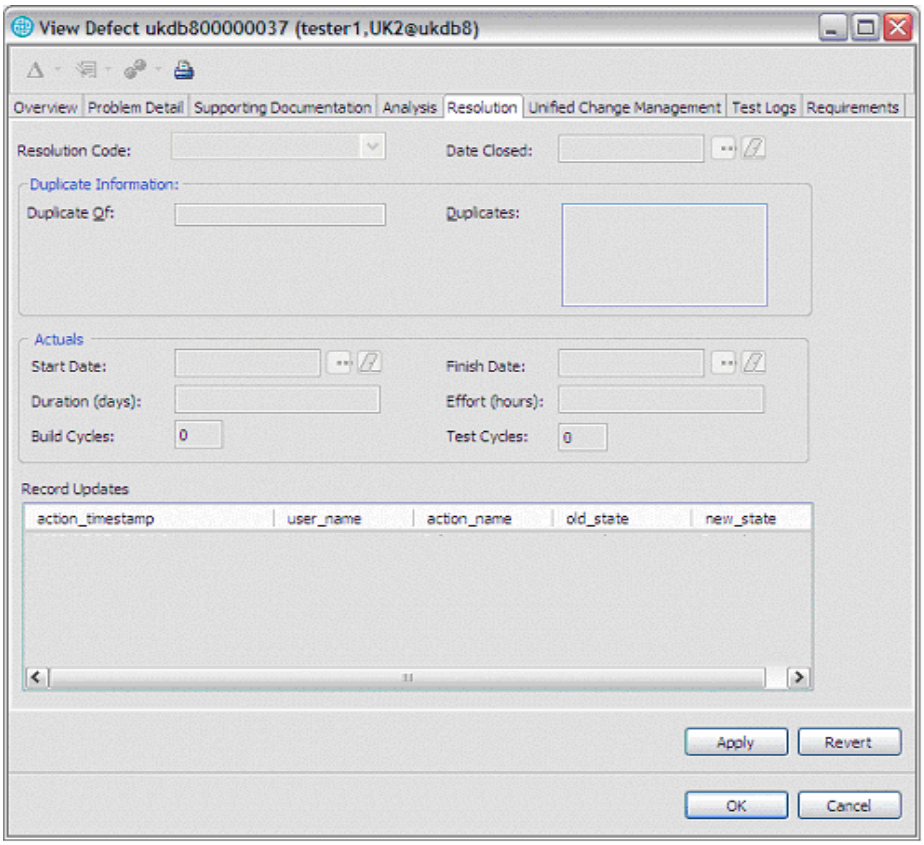
Defect form, Problem Detail tab



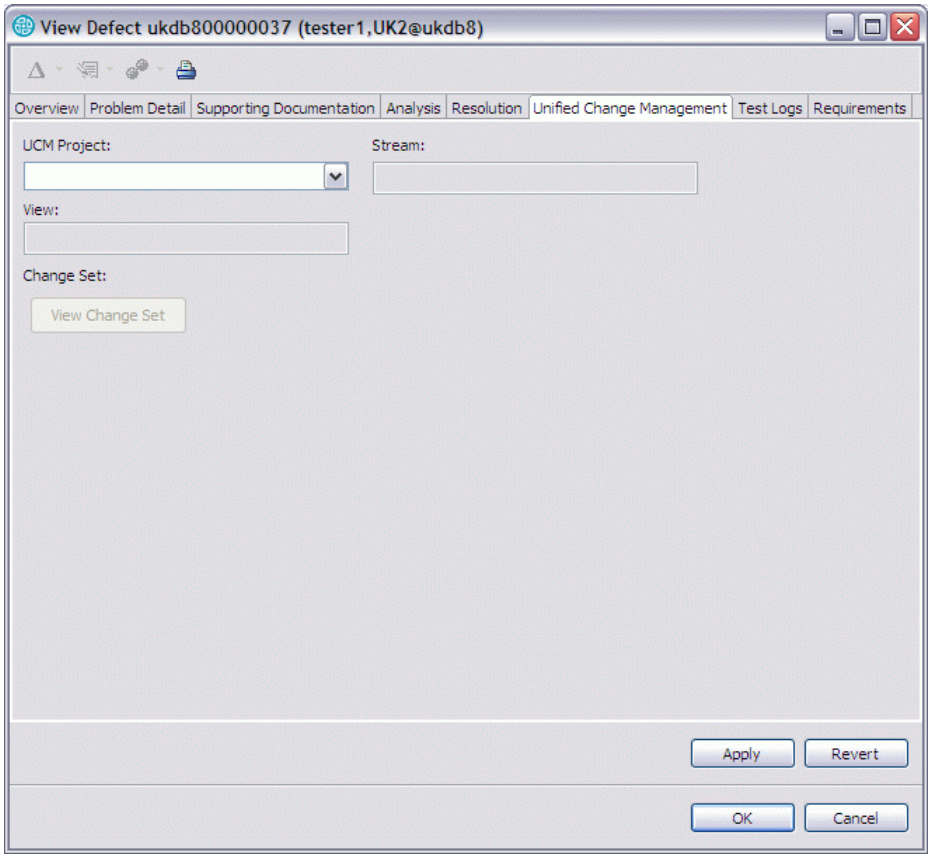
Defect form, Supporting Documentation tab



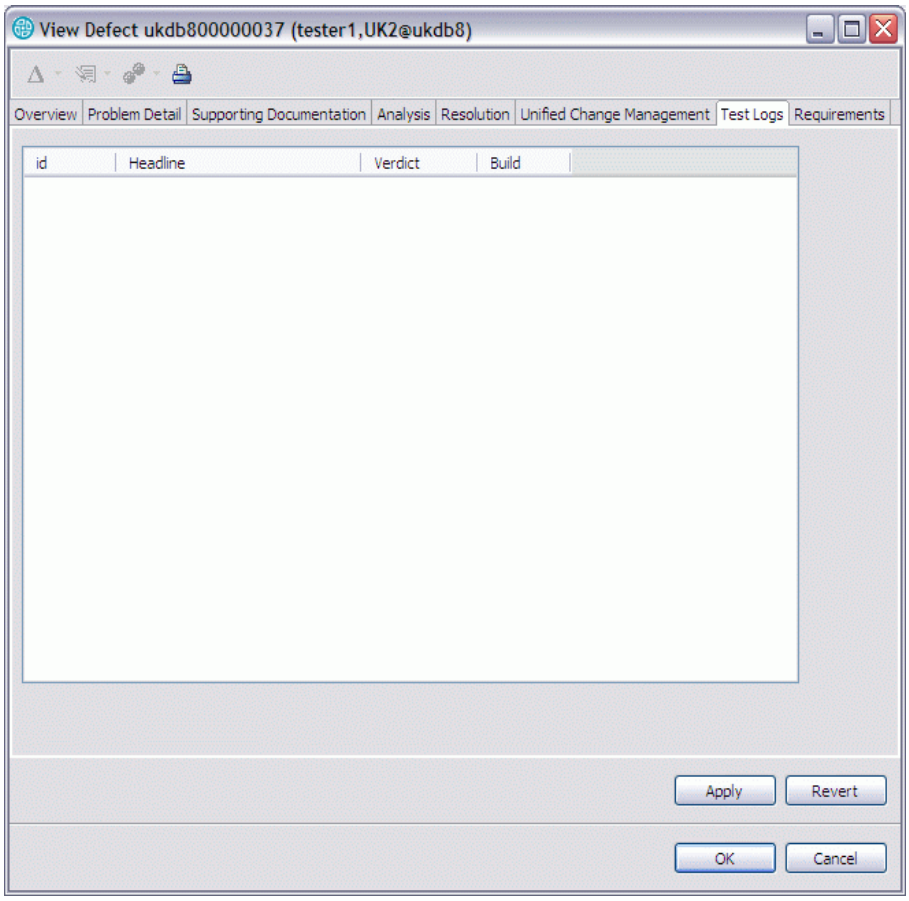
Defect form, Analysis tab



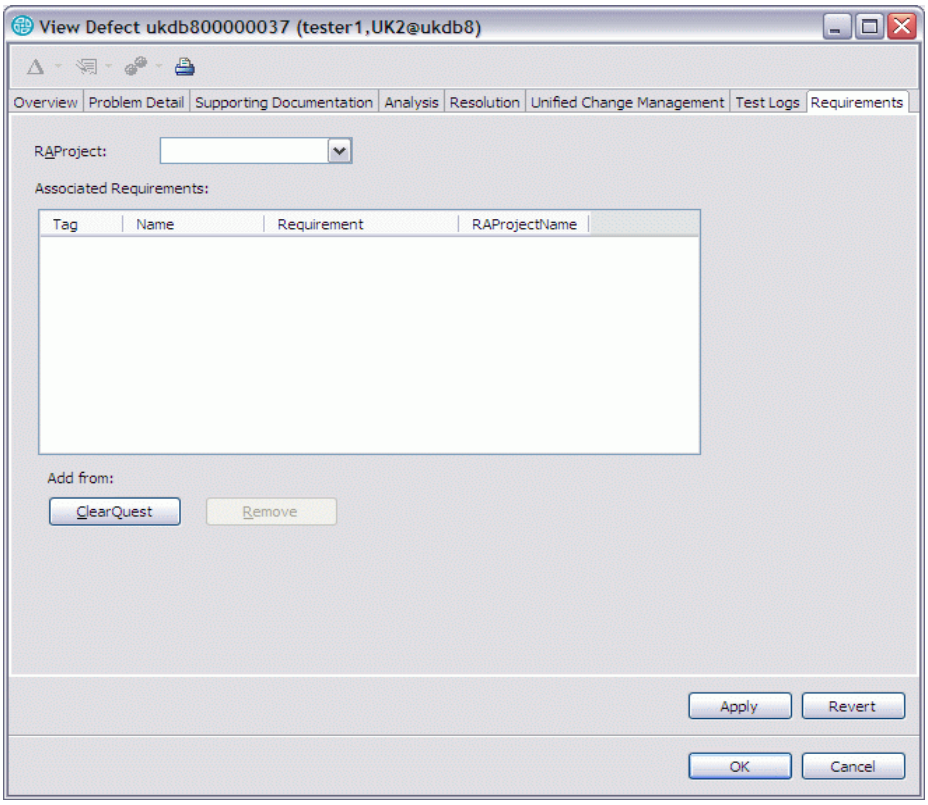
Defect form, Resolution tab



Defect form, UCM tab

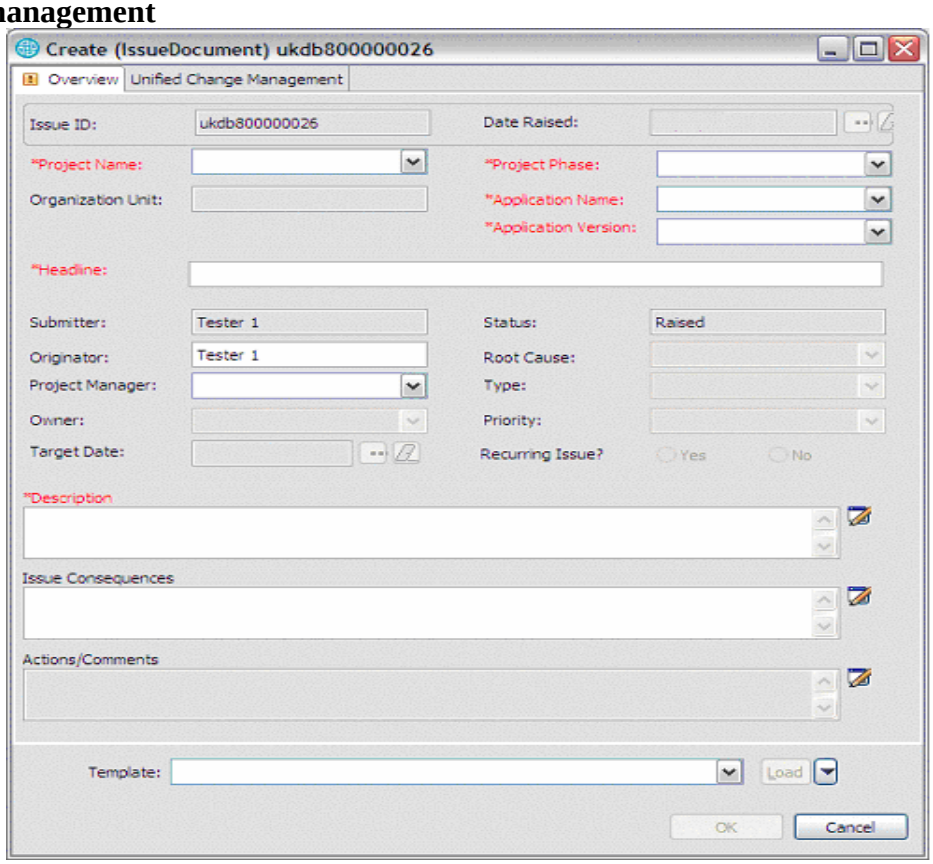


Defect form, Test Logs tab

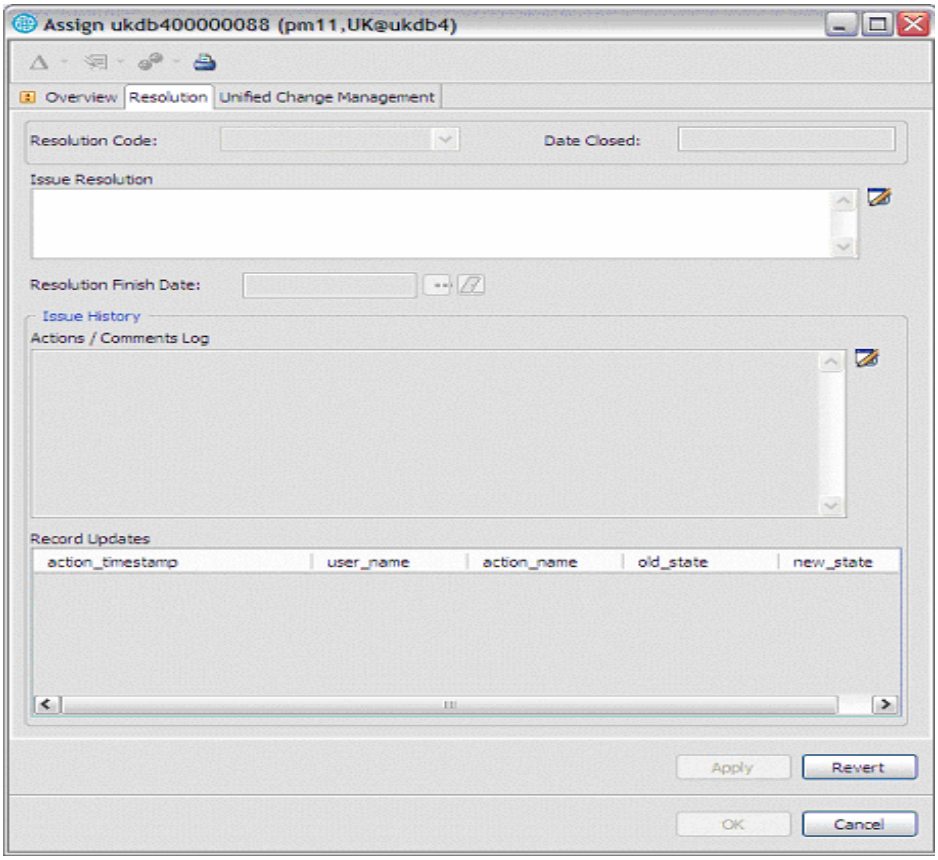


Defect form, Requirements tab

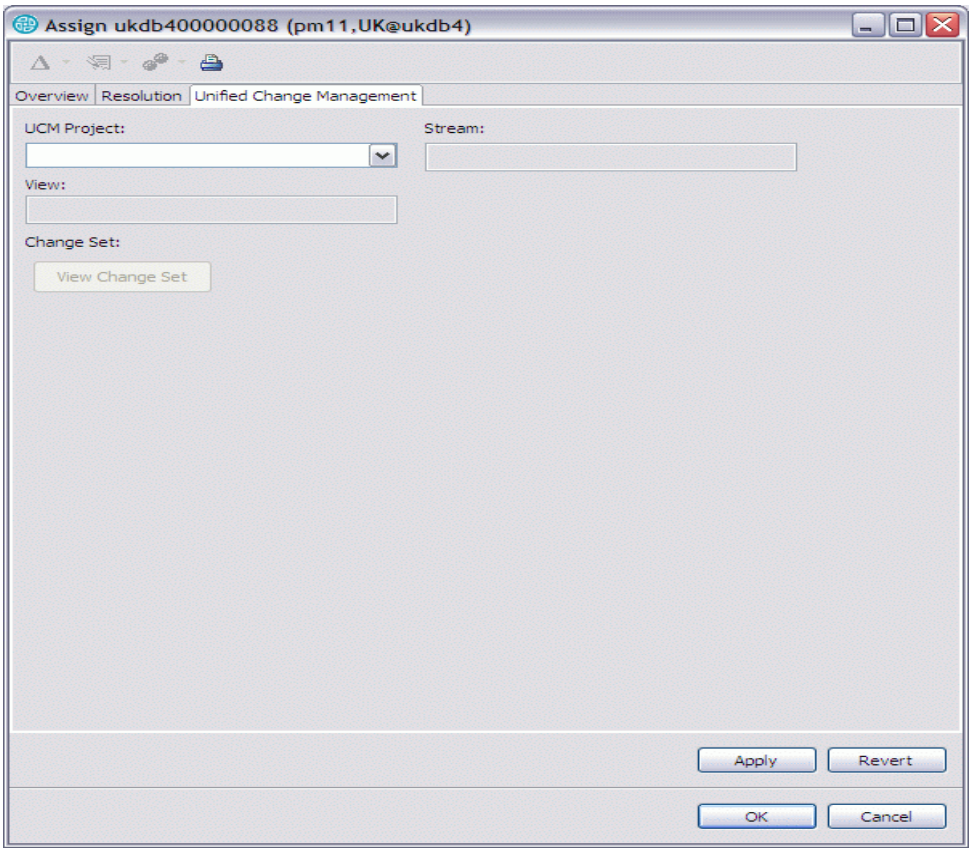
Issue management



Issue form, Overview tab

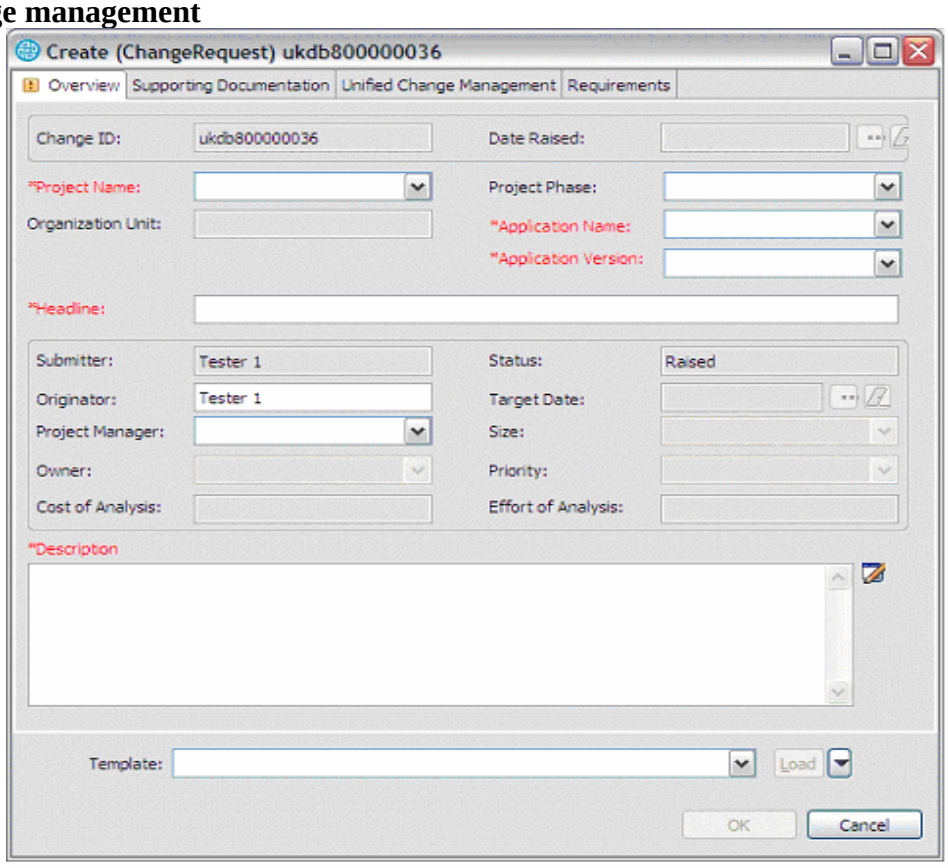


Issue form, Resolution tab

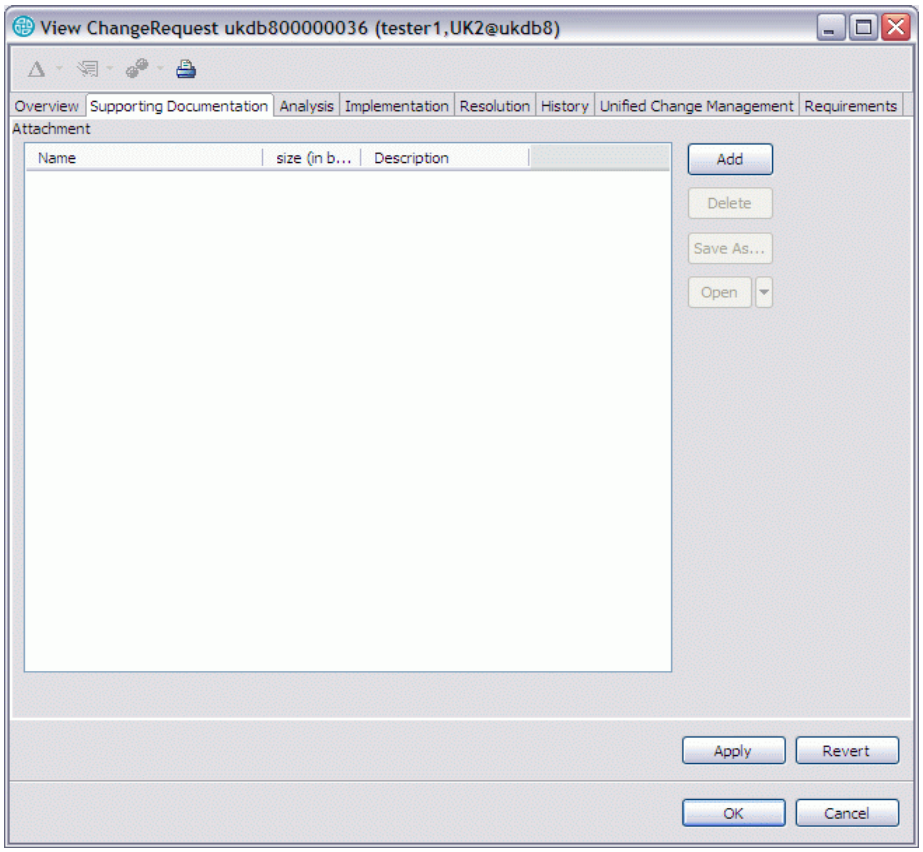


Issue form, UCM tab

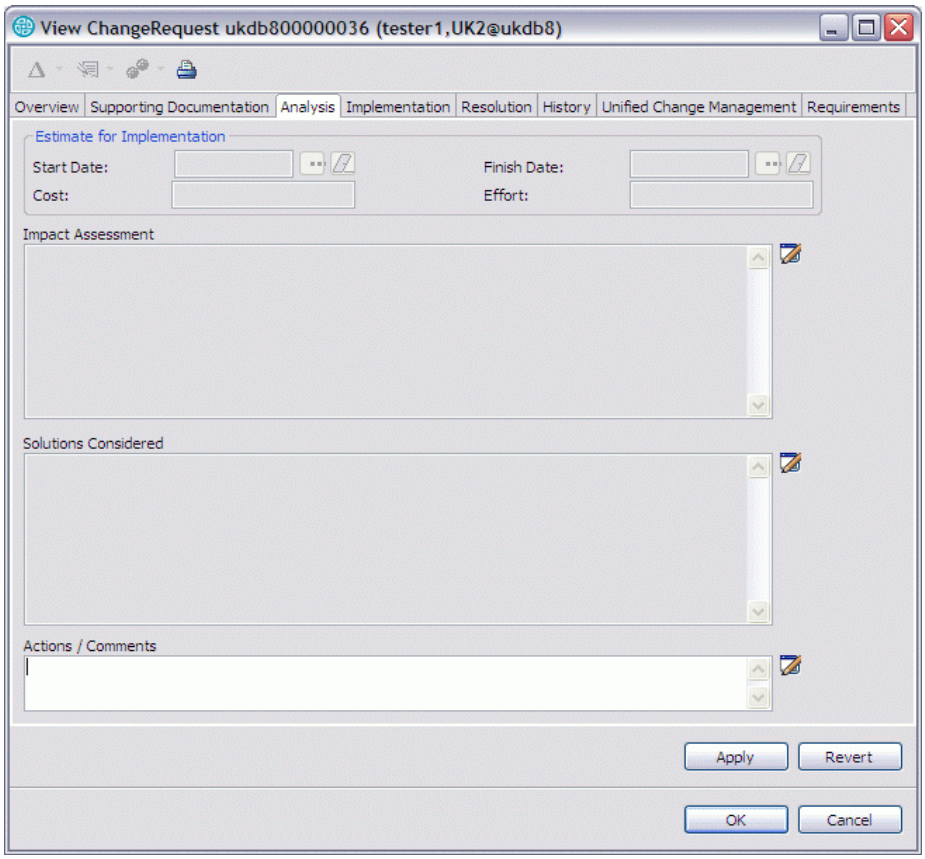
Change management



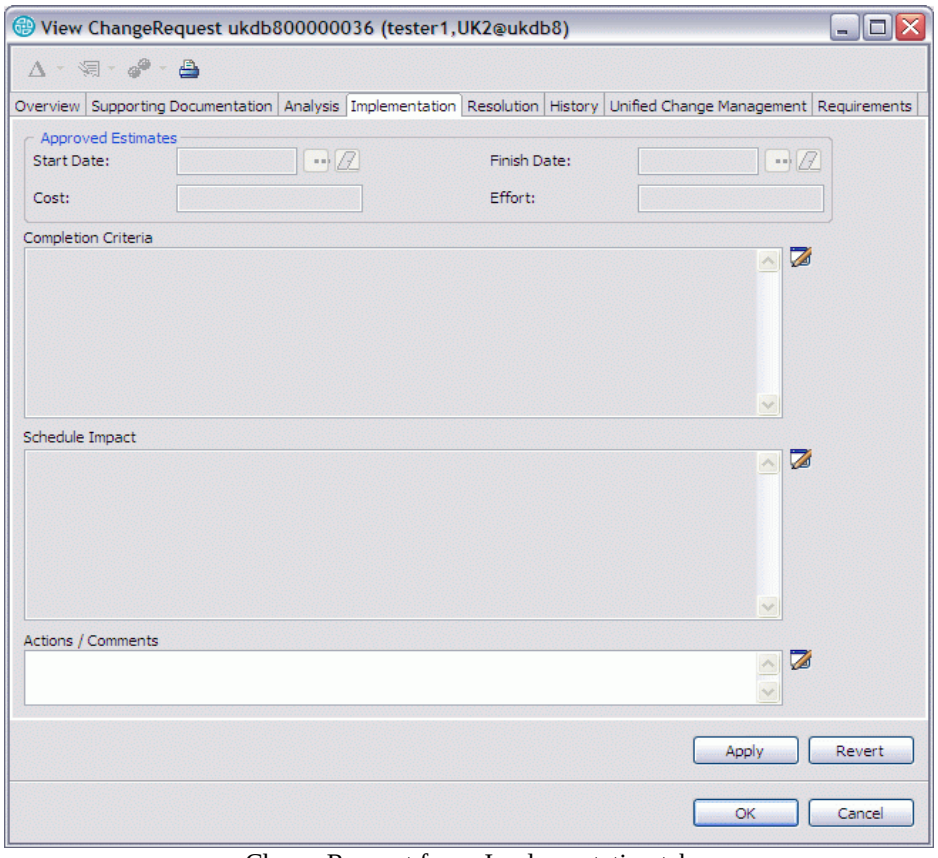
ChangeRequest form, Overview tab



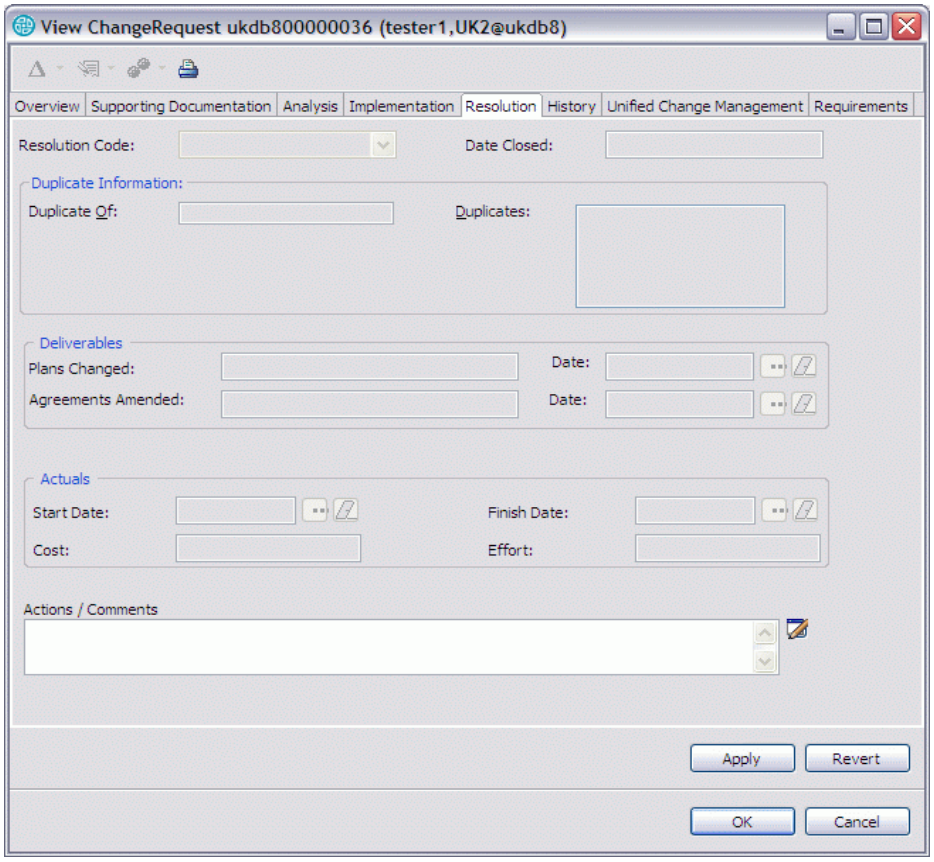
ChangeRequest form, Supporting Documentation tab



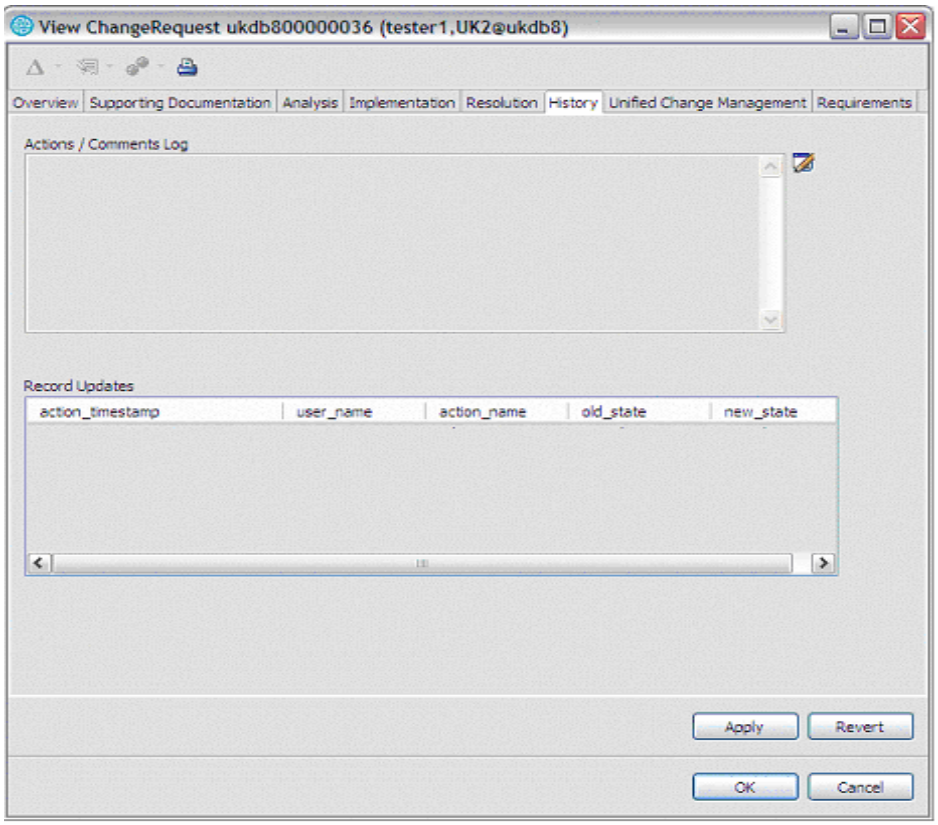
ChangeRequest form, Analysis tab



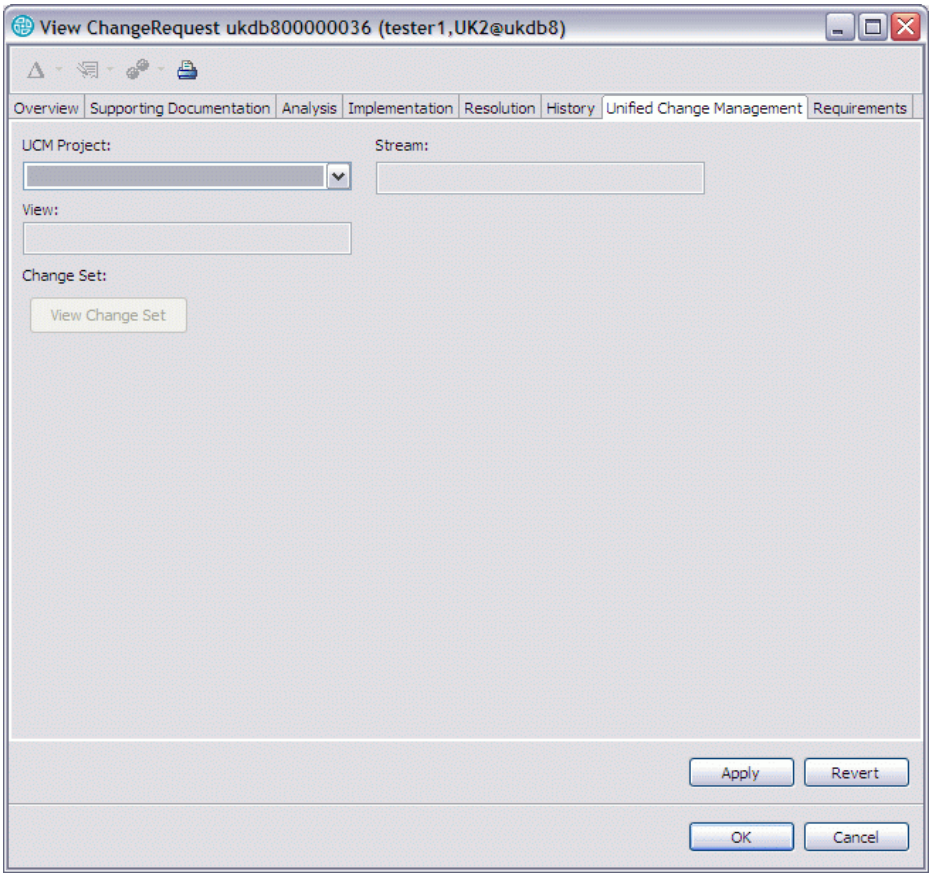
ChangeRequest form, Implementation tab



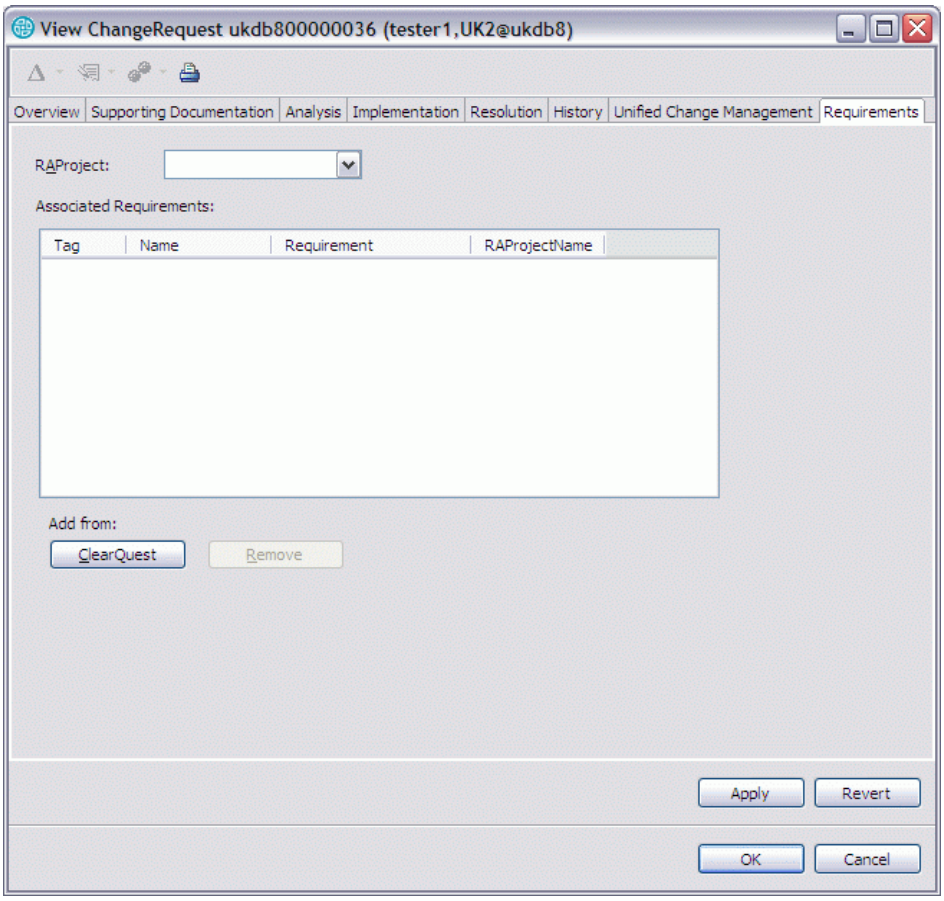
ChangeRequest form, Resolution tab



ChangeRequest form, History tab



ChangeRequest form, UCM tab



ChangeRequest form, Requirements tab