

ELIMINATE
THE
TESTING
BOTTLENECK

Maximize software quality with Requirements Based Testing

Contents

Executive summary	3
The quality crisis	3
Errors in requirements specifications	4
Problematic test coverage	5
Introducing Requirements Based Testing	5
Test early and frequently	5
Test with your head, not with your gut	5
Test with measurement and improvement in mind	6
The Requirements Based Testing process	6
Ensuring quality of requirements	7
Designing logical test cases	7
Ensuring quality of test cases	8
Ensuring quality of design and code	8
Completing and executing tests	8
Traceability	8
Summary of Requirements Based Testing process	9
Implementing Requirements Based Testing with Borland	9
Borland RBT solution architecture	9
Reviewing requirements against business objectives	10
Detecting ambiguities in requirements	10
Applying formality and structure to requirements	11
Reviewing requirements and test cases with various stakeholders	11
Mapping requirements against use cases	11
Generation and optimization of test cases	12
Tracing requirements to test cases	12
Completion and execution of test cases against code	12
Summary	13
Borland RBT process consulting support	13
Conclusion	14

Executive summary

Most business and IT executives agree that any company that is able to rapidly deliver software of high and predictable quality with minimum budgets enjoys a significant advantage over its competitors. However, practical experience shows that the challenges associated with software quality remain largely unsolved: Despite massive investments in testing automation, only a small fraction of the IT application portfolio enjoys sufficient test coverage. It is also apparent that many applications eventually do not meet their users' requirements. These problems lead to poor user experience, lack of adoption, massive application rework—and ultimately loss of competitive advantage.

Chief among the reasons behind this problem is the fact that, in many software organizations, quality is handled as an afterthought. In such organizations, quality verification efforts typically begin only once code has been completed. At this point, clearly the testing team is under the gun to certify the application as quickly as possible, and such certification is often perceived as a bottleneck that prevents deployment into production. In this environment, it is difficult to ensure that the requirements are correct; to properly plan tests; to ensure correctness, completeness and solid coverage of requirements; and to gain visibility into the various quality aspects of the tested application. It is no wonder that frequently this proves to be a costly and frustrating exercise to all involved stakeholders.

This white paper presents the fundamental principles of Requirements Based Testing (RBT), an approach designed to establish a radically different testing process that:

- Is tightly coupled to application requirements
- Is integrated through various activities throughout the software lifecycle
- Delivers measurably high, systematic and cost-effective requirements and test coverage

By focusing on the above principles, RBT eliminates the quality bottleneck and ensures that quality is no longer treated as an afterthought.

The paper also explains how Borland® solutions can help your organization overcome the challenges associated with RBT.

The quality crisis

The Standish Group's Chaos Report and other studies describe what has been the industry reality for decades: The majority of software projects fail to achieve schedule and budget goals. The poor quality of software is one of the biggest reasons behind these many failures, which often result in massive rework of application requirements, design and code. Such rework extends release cycles and consumes significant additional budgets. To combat the high rate of project failures, it is therefore necessary to better understand the reasons behind the poor quality of the products being developed.

Accumulated industry experience and numerous studies show that the two primary reasons behind poor application quality are defects in requirements specifications and problematic system test coverage. The following sections provide more information about these issues.

Errors in requirements specifications

For some, it may be surprising that a system can be thoroughly and successfully tested, yet make its users miserable. The reason is simple: The development team got the requirements wrong.

Requirements of complex software applications are often negotiated through two concurrent dialogues, which continuously evolve throughout the project lifecycle. These dialogues are focused around two questions: What do we need to build? What can we build? The quality of these dialogues often determines the ultimate quality of the constructed application.

Unfortunately, along the way something often goes very wrong. A study by James Martin¹ shows that the root causes of 56 percent of all defects identified in software projects are introduced in the requirements phase. Further, it states that about 50 percent of requirements defects are the result of poorly written, ambiguous, unclear and incorrect requirements. The other 50 percent of requirements defects can be attributed to incompleteness of specification—i.e., requirements that were simply omitted.

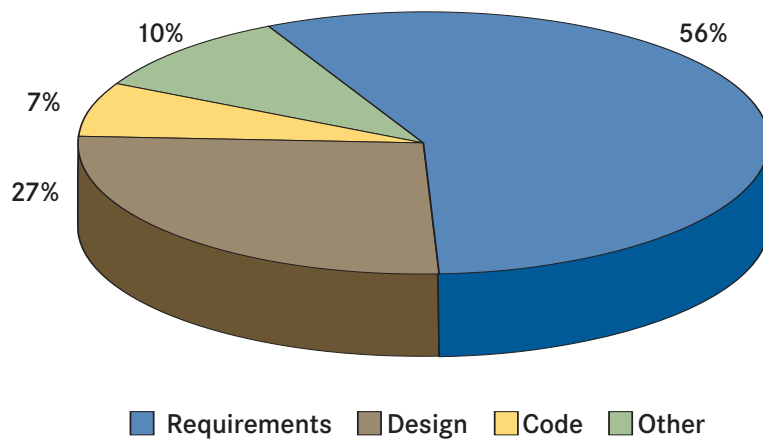


Figure 1: Distribution of defects in software projects

Other statistics demonstrate similar problems:

- 82 percent of application rework is related to errors in requirements²
- Problems in requirements represent 44 percent of the reasons behind project cancellations³
- Only 54 percent of initial project requirements are actually realized⁴

To summarize, the top two issues with quality of requirements are:

- Requirements and specifications are incomplete, due to lack of input from users and other key stakeholders.
- Requirements are specified poorly, by domain experts who lack skills to define consistent, accurate and testable requirements, and whose tools don't allow for efficient validation and/or review of the requirements with the users.

Problematic test coverage

Testing is a tedious and time-consuming activity, requiring careful planning and thorough execution. Achieving good test coverage is a key goal of testing: the better the test coverage gets, the better the chances to catch defects in the testing

1 James Martin, "An Information Systems Manifesto"
 2 James Martin, "An Information Systems Manifesto"
 3 The Standish Group: CHAOS Report
 4 The Standish Group: CHAOS Report

phase and cover all important and “corner” scenarios. Unfortunately, even if requirements are properly stated (i.e., they are complete, accurate and testable), it is still difficult to deliver optimal test coverage.

The following reasons make good test coverage exceptionally hard to accomplish:

- Tests are often conducted at the end of the development process, as schedules become tighter and tighter. As a result, test planning and execution are often performed with severe time constraints, which make it more difficult to effectively cover a large percentage of the system’s functionality.
- The complexity of modern applications (and in particular distributed applications) makes it very hard to cover all of the possible scenarios, due to the sheer number of alternative paths through the application. Any attempt to systematically (or automatically) consider all possible combinations results in a set of test cases that is simply too large to manage and takes a very long time to execute (which means further delay in certification efforts).
- Application requirements change frequently, but their changes are not properly managed. Since traceability between requirements and test cases is often not properly maintained, it is difficult to ensure continuous test coverage of changing requirements.

As a result of these issues, existing methods for creation and selection of test cases are mostly informal, and rely heavily on the experience and skill of testers. It is uncommon that test case design is performed in a systematic and rigorous manner. Many cases are designed based on gut feel and intuition, which is one of the major reasons why software quality and required testing effort tend to be unpredictable.

Introducing Requirements Based Testing

Requirements Based Testing (RBT) is an approach that is implemented through well-defined processes, which target the root causes of the quality problems mentioned previously. RBT is based on the following principles:

- Test early and frequently.
- Test with your head, not with your gut.
- Test with measurement and improvement in mind.

Test early and frequently

RBT promotes a “testing” process that is integrated throughout the entire software development lifecycle. As soon as requirements, design and code artifacts are ready, they are reviewed and “tested” against best practices, business objectives, requirements, use cases and test cases. Testing becomes a parallel activity to the development process, a constant pursuit that spans all roles and makes all stakeholders aware of quality objectives.

As a result, testing is no longer a bottleneck activity, performed after code is delivered under extreme time pressures. Many defects are detected at earlier development phases, when it is much cheaper to fix them. Additionally, there are significantly fewer surprises when the code is delivered.

Test with your head, not with your gut

RBT supports methodical and systematic design of test cases, to ensure predictable test coverage. Following a rigorous process, RBT attempts to facilitate test case design that does not rely on the skill or experience of specific testers. Rather, RBT strives to inject repeatability and method to the test planning process in order to make test coverage more

predictable. RBT also attempts to apply various optimization techniques to produce the minimum number of test cases required for sufficient test coverage. By doing this, RBT makes the testing cycle more rapid and manageable.

Test with measurement and improvement in mind

RBT promotes a quality process that can be managed and improved through measurement. Throughout this process, multiple measures help to quantify the status of deliverables and activities. This helps managers and process experts oversee quality initiatives across the IT application portfolio.

The Requirements Based Testing process

Requirements Based Testing was perceived and refined through practical industry experience as well as academic research. To various degrees, it is widely implemented by many organizations to ensure that they define software requirements and design test cases that provide good test coverage and ensure that quality activities are well integrated throughout the entire development process.

The following diagram shows the RBT activities and areas of focus of the RBT process.

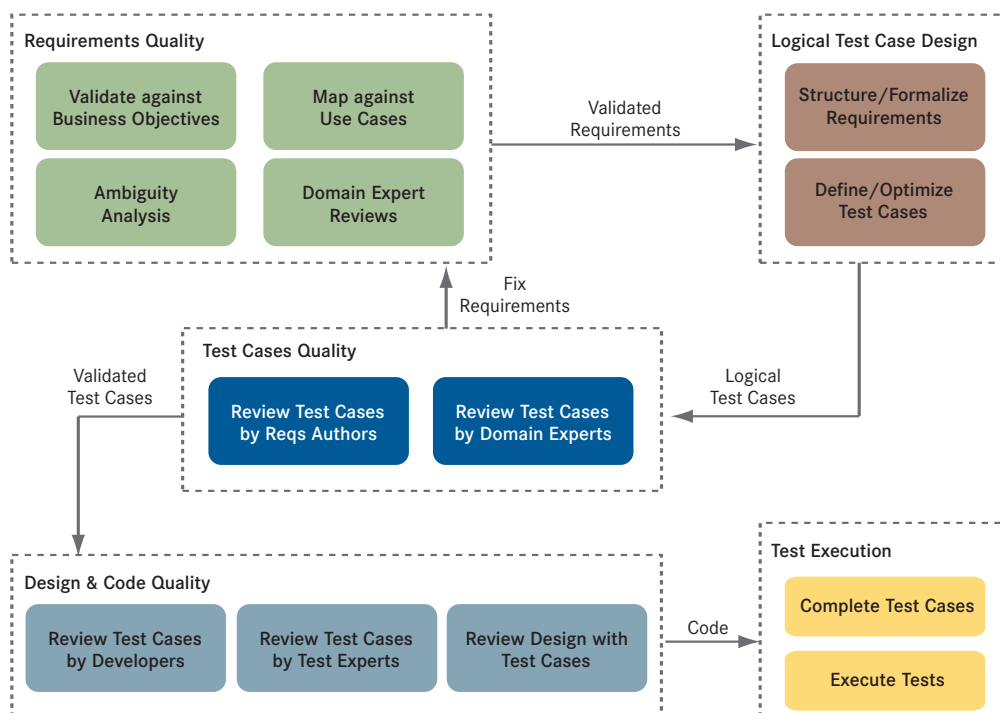


Figure 2: Requirements Based Testing process flow

Rather than focusing on the “traditional” testing activities, which are typically performed after code is delivered, RBT emphasizes the full lifecycle quality activities. These activities are aimed at increasing the quality of requirements and test cases, as well as encouraging the utilization of requirements and test cases in other phases of the software lifecycle, such as design and development. This focus ensures that when tests are finally executed, they are truly associated with initial business requirements.

The following sections provide additional information about each of the RBT categories of activities.

Ensuring quality of requirements

Because the success of a software project can be directly associated with a good understanding and articulation of its requirements, RBT places a great deal of importance on the quality of requirements.

To ensure that business needs are eventually met, RBT includes the validation of requirements against business objectives. This also optimizes project scope by ensuring that each requirement satisfies at least one business objective. If there is no match between the requirements and business objectives, refinement is necessary.

To enable involvement of end users and domain experts, RBT recommends a review of the requirements by these stakeholders. Feedback of users and domain experts is used to refine the requirements before additional work is done.

To validate completeness of the requirements, RBT calls for mapping requirements against a task-oriented or interaction-oriented view of the system, often captured by use cases. If one or more use case cannot be addressed by the requirements, then the requirements are not complete.

To guarantee the consistency and clarity of the requirements, RBT applies language analysis techniques to detect problematic phrases in the requirements and fix them. A problematic phrase is language that is ambiguous, unclear or inconsistent. Such phrases can be interpreted in multiple ways by different people, which leads to confusion and errors down the road. Ambiguous terms also result in requirements that are not testable.

Designing logical test cases

Textual requirements are very important, given that almost all business analysts and domain experts use natural language to express requirements. However, using textual requirements makes it more difficult to achieve and validate high test coverage. When requirements are expressed in a nonformal fashion using natural language, testers can use only their intuition to claim that the set of test cases they designed covers 100 percent of the functionality captured in the requirements. In other words, with nonstructured textual requirements, there is no formal way to show complete test coverage, and therefore “corner” or even major scenarios often remain untested until defects occur in production.

To meet the objective of systematically achieving high test coverage, RBT introduces a process step aimed at creating more formal and structured representations of the requirements. Once this is done, it is possible to define “logical” (skeletal) test cases and ensure optimal coverage of requirements, as these logical test cases are eventually evolved into the actual set of tests that are run against the system.

Multiple techniques can be used to provide structure and formality to natural language requirements. The purpose of these techniques is to reveal cause-effect relationships embedded within requirements; that is, to express requirements as a set of conditions (causes) and resulting actions (effects). Cause-effect charting is one of these techniques. Another way to achieve similar goals is to express requirements as flowcharts, since they naturally depict precedence dependency between actions as well as conditional branching of activities.

Once requirements are expressed in a structured manner, it is much simpler to derive the equivalent test cases for the requirements. A set of logical test cases can be defined (manually or automatically), which is 100 percent equivalent to the functionality captured in the requirements. However, this set of test cases may include many redundant cases (that is, overlapping with other test cases). To optimize the number of test cases but still provide full coverage, techniques such as decision (truth) tables can be applied if cause-effects charts were used to structure the requirements. If flowcharts were used for that purpose, then generation of the optimal set of test cases means finding all unique paths on the flowchart, for which there are known techniques.

Ensuring quality of test cases

Once test cases have been designed, there is a probability that some errors still exist in them. To detect errors in test cases and fix them, RBT incorporates reviews of the test cases by the authors of the requirements and by end users and domain experts. This phase gives these important stakeholders an opportunity to review the requirements and the test cases in their structured form, and catch any defects that were introduced in the transition from natural language to formal form.

Ensuring quality of design and code

RBT goes an extra mile to integrate the quality process into the development process, using the structured, validated, high-quality test cases that were generated in the previous phases.

To ensure that the design is robust enough to satisfy the requirements, it is reviewed against the logical test cases (as they are simply a different representation of the requirements). If the design cannot meet requirements, then either the requirements are problematic (and additional refinement is necessary) or the design requires rework.

To ensure that developers understand what will be tested, and to provide them with a structured view of requirements, RBT recommends that test cases be reviewed by the developers (coders).

To ensure that the delivered code conforms with the requirements, RBT recommends that individual code modules be reviewed against the structured requirements that trace to them. It is much easier to match the algorithmic expressions captured in code to the formal view of requirements than to compare it to their nonstructured view.

Completing and executing tests

Using the logical test cases, testers can now complete them by defining data inputs and navigation flows, potentially using test automation tools. To compare the actual behavior of the system to the expected behavior, the complete set of fully defined tests can now be executed against the code.

Traceability

While not exactly a “phase” in RBT, traceability has an important role in its success. With RBT, maintaining traceability information between requirements and test cases and tests is crucial. This information is required for monitoring progress and coverage, as well as properly managing the impact of changes in requirements. Without it, it is more difficult to determine which test cases or tests should be changed if a specific requirement changes.

Summary of Requirements Based Testing process

The table below summarizes the RBT goals, associated objectives and how they are met.

Goals	Objectives	Method Used
Ensuring quality of requirements	Ensure that business needs are eventually met, enable involvement of end users and domain experts, validate completeness of the requirements, guarantee the consistency and clarity of the requirements	Review of requirements against business objectives, review of requirements by domain experts and business users, mapping requirements against use cases, ambiguity analysis of requirements
Designing logical test cases	Provide structure and formality to natural language requirements, derive the equivalent test cases for the requirements, optimize the number of test cases	Express requirements using cause-effect charts or flowcharts, automated or manual generation of test cases, decision (truth) tables or unique path detection
Ensuring quality of test cases	To detect errors in test cases and fix them	Review test cases by requirements authors, domain experts and end users
Ensuring quality of design and code	To ensure that the design is robust enough to satisfy the requirements, to ensure that developers understand what will be tested, to ensure that the delivered code conforms with the requirements	Review of test cases by designers and developers, use test cases in code review
Completing and executing tests	To compare the actual behavior of the system to the expected behavior	Full definition of test cases, execution of test cases against delivered code

Implementing Requirements Based Testing with Borland

Borland supports the RBT process with a wide array of products, professional services and educational offerings. Borland technology provides a broad basis of support, with products that enable effective gathering and validation of requirements, efficient management of requirements and their traces to other work products, and efficient development of test cases to verify and validate the requirements. Borland training provides skill development in planning quality activities, performing a wide variety of work product reviews, and effectively using the suite of Borland technology. Borland consulting offerings enable organizations to identify the primary gaps in their lifecycle quality management practices and iteratively implement appropriate processes to remove the gaps, melding in the needed skills development and enabling technology along the way.

Borland RBT solution architecture

Borland requirements management and definition technology supports many of the important concepts of RBT. The Borland® Caliber® DefineIT™ product can be used to elicit and structure requirements, and to generate and optimize test cases. Once requirements have been properly defined, Borland® CaliberRM™ is the solution of choice to manage requirement changes and their traceability across the lifecycle. Integrations of Borland's requirements tools with Borland® Together® and Borland® SilkCentral® Test Manager products enable designers and testers to efficiently participate in the RBT process (see Figure 3).

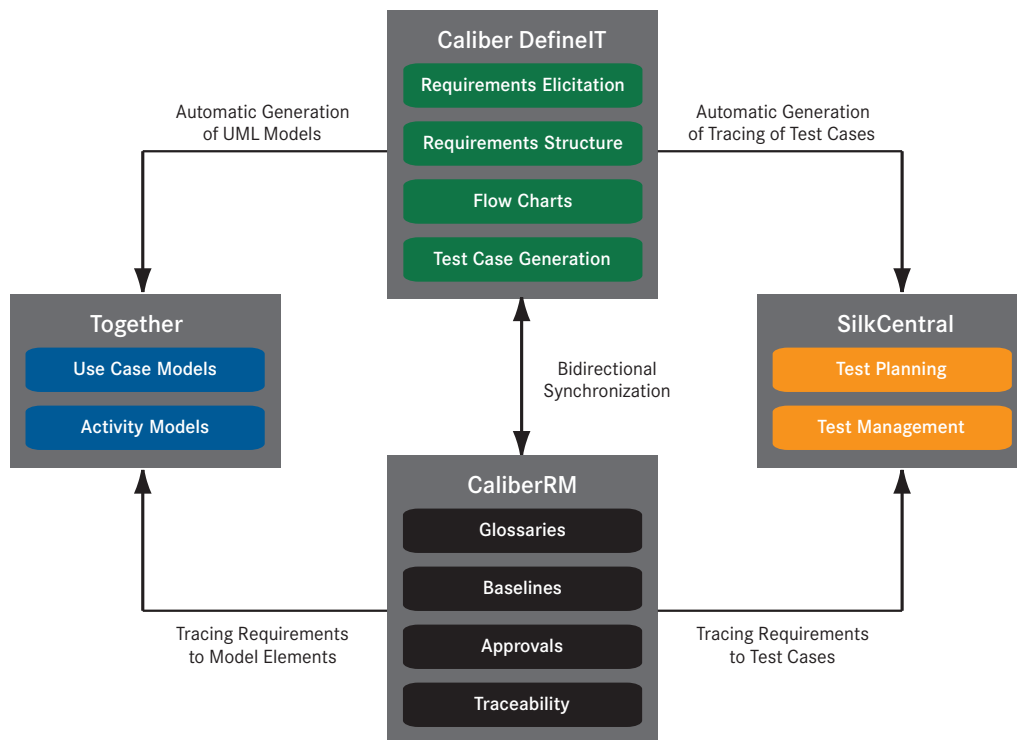


Figure 3: Borland Requirements Based Testing solution architecture

The following sections explain how organizations interested in RBT can automate significant parts of the process using Borland products and integrations.

Reviewing requirements against business objectives

Borland® Tempo® enables business and IT teams to collaborate by capturing and managing business objectives, as well as ensuring that a proper process for selecting and kicking off IT projects is in place. Borland’s process and training offerings provide guidance in how to gather and describe business objectives, business requirements and system requirements. Once requirements are articulated, RBT practitioners can use Tempo to evaluate them against agreed-upon business objectives.

Detecting ambiguities in requirements

To help requirements authors detect and resolve ambiguities in requirements, Borland provides skill-building workshops in peer review techniques for carefully examining key work products (such as requirements, designs, code and test cases) to eliminate defects and rework in the development lifecycle. For some teams, very formal reviews (inspections) are appropriate; for others, simpler walk-throughs or technical reviews may be appropriate. Borland workshops help teams to decide when to plan and execute each type of review.

To support these reviews, CaliberRM enables teams to include glossaries of terms that help work product developers and reviewers minimize ambiguity. CaliberRM glossaries enable business analysts to define multiple dictionaries, which include words that are considered “bad terms,” and those that should not be used when defining requirements (because they are too general or ambiguous). Glossaries also make it possible to define common terms that are often used in requirements, such that all roles have a clear and uniform understanding of their meaning. Once glossaries are in place,

CaliberRM automatically highlights requirements text that matches glossary terms, and provides users with helpful tool-tip data.

Applying formality and structure to requirements

Borland Caliber DefineIT helps business analysts structure requirements while capturing them through the definition or elicitation process. DefineIT structures requirements as “scenarios,” each of which defines a specific high-level interaction with the application, and consists of several steps. Every step represents a fine-grained action within the scenario.

DefineIT shows two structured representations for a scenario: a “textual” representation of the scenario’s steps and a graphical representation of the scenario’s steps, which is a flowchart that captures the sequence of steps within a scenario and their branching logic. The textual and graphical representations of a scenario are kept synchronized at all times (that is, actions performed on a specific representation are immediately shown in the other one).

Since requirements are structured in DefineIT, it is possible to validate their logical consistency. DefineIT provides a validation pane that shows messages about potential logical problems with a scenario. As an example, if a decision point is added to a scenario, but all branches lead to the same step, then the decision point does not make sense. DefineIT notifies the user about this as well as other problems, and thereby helps to ensure that the scenario is defined in an accurate and logical manner. The validation pane is updated in real time, as users manipulate the scenario.

Reviewing requirements and test cases with various stakeholders

RBT includes the review of requirements and test cases with multiple stakeholders, including domain experts, end users, designers and developers. These review sessions should provide both nontechnical and technical audiences with a clear and concise understanding of expected system behavior. Their input should be captured and used to iteratively improve and refine the requirements. Borland peer review workshops enable stakeholders and team members alike to have the skills to perform such reviews effectively.

The main goal of DefineIT is to close the “language gap” between business and IT, by facilitating a more structured dialogue between the two teams. Business stakeholders are often domain experts, speak the “domain language,” but lack understanding of technical terminology. IT stakeholders are well versed with the technical terminology, but often lack expertise and understanding of the problem domain. DefineIT synchronizes the two teams by facilitating an accurate and complete “knowledge transfer” from analysts and business stakeholders to the technical team.

Caliber DefineIT elicitation capabilities are designed specifically to elicit requirements from various stakeholders through a well-structured review process. Once a scenario is modeled in DefineIT, the storyboarding feature of DefineIT can be used to show it to the application’s end users. This helps to validate that the captured scenario is indeed in sync with what stakeholders expect.

Storyboarding a scenario with DefineIT essentially means walking reviewers through multiple “paths” defined by the scenario steps and decision points, while capturing the reviewers’ feedback and comments, which can be used to improve the scenario.

Mapping requirements against use cases

The integrations of CaliberRM and DefineIT with the Borland Together modeling product enable business analysts and architects to collaborate and ensure that requirements and use case models are synchronized, such that the software design is based on validated and accurately specified requirements.

This integration of DefineIT and Together enables users to export a DefineIT project to Together as a set of UML-compliant use case and activity diagrams. This capability essentially transforms the DefineIT scenario representation

into equivalent UML models. The resulting diagrams include links to reference materials provided in DefineIT, such as images, and these can be directly viewed from within Together.

Users can also export DefineIT projects as Business Process Modeling Notation (BPMN) diagrams through OMG's Query View Transfer (QVT) specifications. DefineIT users can export a project as XMI, and architects can use the QVT capabilities of Together to transform it into a BPM diagram. After the BPMN diagram is created, it is also possible to use Together to generate BPEL for execution.

Generation and optimization of test cases

Caliber DefineIT can automatically generate a set of test cases for a specific scenario within a project. By doing this, DefineIT enables the functional test coverage of the application.

DefineIT also helps minimize the number of generated test cases. Each generated test case represents a unique path within the scenario. Selecting a specific test case highlights the associated path within the scenario diagram.

DefineIT generates a textual description of all the steps that need to be taken within the test case; and for each, specifies the Actor, the action taken and the expected result. Generated test cases can be exported to Borland SilkCentral. It is also possible to export generated test cases to HTML, which creates an interactive and navigable website where the QA team can view test cases and their associated artifacts.

Tracing requirements to test cases

Both CaliberRM and Caliber DefineIT enable testers to associate their test cases with requirements, to ensure that their test suites ultimately provide full coverage for the complete application scope. This capability is supported via integrations with the Borland SilkCentral Test Manager.

The DefineIT integration with SilkCentral makes it possible to automatically transfer requirements and test cases generated by DefineIT to SilkCentral. Exported requirements appear in the selected project tree, and for every requirement the test cases and test steps are exported as well. Once export is done, test cases are automatically linked to their respective requirements within SilkCentral.

CaliberRM provides similar integrations with SilkCentral. Additionally, it provides a robust set of traceability visualization and analysis tools.

- **The Traceability Matrix** is a powerful interactive table that shows all traces between requirements, model elements, code and tests in a matrix format.
- **The Traceability Diagram** is an interactive tool that shows a visual graph of trace dependencies between requirements and other lifecycle elements including test assets.

CaliberRM is able to visually mark all traces to or from requirements that were changed as "suspect." This capability is key to impact analysis, as it helps testers quickly find and access all test artifacts that are affected by a change in requirements. They can then access impacted test cases in SilkCentral and modify them to cover for the changed requirements.

Completion and execution of test cases against code

Once "skeletal" test cases are generated by DefineIT and exported into SilkCentral, it is possible to provide detailed test definitions and run a series of functional and performance tests using Borland SilkCentral, SilkTest and SilkPerformer. If the RBT process has been followed, at this point the testing team can be assured that test coverage is optimal.

Summary

The table below maps Borland's products and services to the required RBT process components.

RBT Process Component	Supporting Borland Product
Gathering and documenting requirements effectively (all levels)	Requirements Gathering, Definition, Management and Engineering workshops
Review of requirements against business objectives	Tempo, Peer Review Workshop, CaliberRM
Review of requirements by domain experts and business users	Peer Review Workshop, Caliber DefineIT / elicitation
Mapping requirements against use cases	CaliberRM and DefineIT Together integrations
Ambiguity analysis of requirements	Peer Review Workshop, CaliberRM glossaries
Expressing requirements formally using cause-effect charts or flowcharts	Caliber DefineIT / flowcharts
Automated or manual generation of test cases	Caliber DefineIT / auto test-case generation; SilkTest and SilkPerformer training
Optimization of number of test cases using decision (truth) tables or unique path detection	Caliber DefineIT / flowcharts; SilkTest and SilkPerformer training
Review test cases by requirements authors, domain experts and end users	Peer Review Workshop, Caliber DefineIT / elicitation
Review of test cases by designers and developers	Peer Review Workshop, Caliber DefineIT / elicitation, SilkTest and SilkPerformer training
Completion of test cases, execution of test cases against delivered code	Borland Silk™ product line and training
Traceability of requirements to test cases	CaliberRM and DefineIT integrations with SilkCentral

Borland RBT process consulting support

Borland's Lifecycle Quality Management (LQM) solutions are supported by a range of process development and skill-building offerings, as well as Borland technology. The specific solution for any customer situation depends on the current state of LQM and the goals for the organization. Borland uses its Accelerate framework to architect and deliver such solutions.

Specific elements for LQM include the following:

- **Executive Workshop for LQM.** Several-hour session for organization executives to review industry best practices for LQM and set business goals for their organization with respect to improvements in LQM.
- **LQM Gap Analysis Assessment.** In-depth analysis of the organization's current LQM processes, skills and technology against industry best practices, to identify where the organization might benefit most from improvements; duration depends on the number of areas examined and size of the organization (generally 3 to 10 days).

- **ActionPlanning Workshop.** Three-day workshop for the team, which implements the recommendations that come from the gap analysis.
- **LQM Consulting.** Guidance for the development and deployment of improvements to skills, processes and technology, as well as assistance with organizational changes.
- **Peer Review Workshop.** Two-day workshop teaching a range of types of work product reviews and how to select an appropriate style for different purposes; includes hands-on experience doing formal inspections.
- **Silk Testing Suite product and practices training.** Full set of training offerings to enable effective use of SilkCentral Test Manager, SilkTest and SilkPerformer.

Conclusion

By integrating quality activities with the overall software development cycle, Requirements Based Testing can help organizations to systematically deliver predictable levels of software quality. With RBT, quality is not treated as an afterthought, nor are testing efforts perceived as a bottleneck. Borland solutions and process consulting services—alone or complementing offerings from other leading testing vendors—can significantly assist with rollouts of Requirements Based Testing.