Quality management
White paper
September 2009

Rational® software

IBM

Deliver robust products at reduced cost by linking model-driven software testing to quality management.

Contents

Closing the productivity gap between development and testing

With embedded product designs becoming more complex and product lifecycles shrinking, development efficiencies are essential. Fortunately, emergence of the Unified Modeling Language (UML) and Systems Modeling Language (SysML) standards has provided the opportunity for breakthrough gains in streamlining the development process. These standards enable engineers and software designers to significantly improve their productivity by transitioning from a code-based development process to model-driven development (MDD).

Using models, software engineers can more clearly understand and analyze requirements, define design specifications, test systems concepts using simulation and automatically generate code for direct deployment on the target hardware. Development teams can standardize processes and automate repetitive tasks to boost productivity and enhance regulatory compliance through self-documenting data and workflows. As a result, engineers and developers are able to deliver more complex and intelligent designs in much less time.

These benefits of model-based development are well documented. But software testing is another story altogether. The testing process is still largely code based—creating a serious productivity gap in the development process. Designs can be produced much faster than they can be tested.

It’s not that model-based testing tools don’t exist. They do and have been proven to save time by enabling developers to test in the same language in which they design—UML. Another advantage is that when developers start with scenarios as requirements, they are testing against what customers and marketing staff have agreed too.

Highlights

A major roadblock to adopting model-driven testing is lack of integration to quality assurance.

So why are developers continuing to code their own test scripts? A major roadblock of model-based testing is the lack of integration to quality assurance (QA). Why aren't model-based tests available to the QA team? The main reason is because QA professionals don't understand UML and have little reason to learn it.

This white paper explains how organizations can overcome this obstacle. It highlights the benefits of model-based testing and describes a model-driven approach designed to create tests that can be executed from external sources such as QA. No matter who executes the tests, results are available to all affected practitioners—closing the loop from requirements to verification and validation of use cases.

The benefits of model-driven software testing

Year-over-year studies by the Embedded Market Forecasters organization show that model-based testing helps companies deliver complex designs on time while meeting predesign expectations for performance, systems functionality and features. These studies compare actual data gathered from surveys of practitioners in three types of development and testing projects: legacy, transitional and enhanced. The legacy projects used no model-based practices at all, either in development or testing. The transitional projects used model-based development, but code-level testing. And the enhanced projects used models to drive both development and testing.

Highlights

When models are used to drive both development and testing, project teams are able to deliver better outcomes in a shorter period of time—a key reason being early detection of defects.

The most recent comparative data shows that enhanced projects produce designs that cost, on average, tens of thousands of dollars less than those produced in legacy and transitional environments. Enhanced designs also were completed more quickly—requiring up to six weeks less time from start to shipment. And MDD and model-driven testing (MDT) together resulted in better design outcomes (see figure 1).[1]

	Code-based with no MDD (legacy)	Code-based with MDD (transitional)	MDT with MDD (enhanced)
Performance	76.4%	80.0%	88.8%
Systems functionality	76.7%	80.0%	81.5%
Features and schedule	72.0%	74.0%	70.3%

Figure 1: Projects using both model-driven development and testing completed significantly more designs with outcomes that met predesign expectations.

Early detection of defects was one of the key reasons that enhanced projects were able to deliver better outcomes in a shorter period of time.

The role of model-based testing in early detection of defects

According to the National Institute of Standards and Technology, 80 percent of development costs are spent on identifying and fixing defects.[2] The majority of defects are most likely introduced during the early stages of design and development when errors can be fixed more easily and at a fairly low cost. Unfortunately, most defects tend to be found late in the process, sometimes after product release.

Highlights

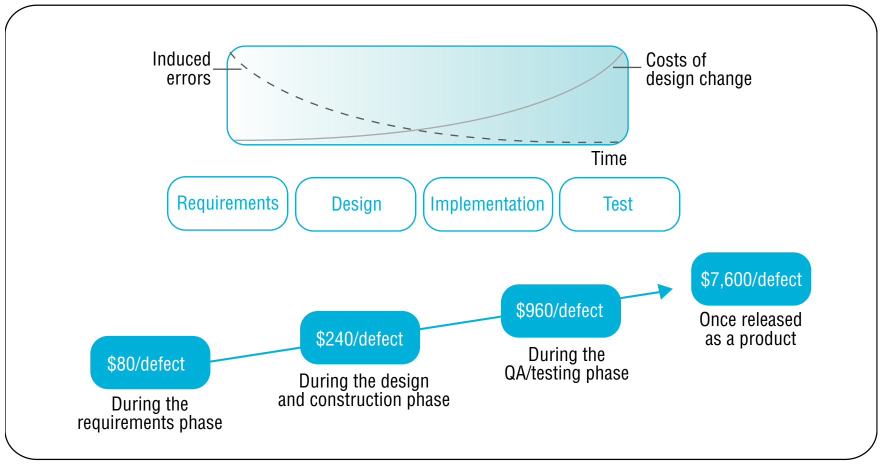

Figure 2: The costs of correcting defects increases geometrically throughout the development process.

The obvious response is to perform tests as early as possible in the development process. That way, errors can be found and fixed before they propagate.

If the development environment uses models but the test environment relies on code, there is no effective way of navigating from test cases to use cases to validate features against requirements.

Verification and validation are the two fundamental strategies for software testing. Verification answers the question, "Is the feature developed in the right way?" Validation answers the question, "Is it the right feature?" When the development and test environments are separate—because the first is model-centric and the second is code-centric—no one can navigate from the test cases all the way back to the use cases to validate requirements.

Highlights

Using the design model to create more efficient and higher-quality test cases can result in better coverage of requirements.

If testing can be extended forward in the process to include the design model, quality managers and software engineers can check that each feature meets requirements as soon as it is implemented in the model. And, if the testing process uses model-based validation activities to automate test code, developers are able to be more accurate and save significant time verifying code coverage, detecting any memory problems or finding other run-time issues.[3]

In short, extending traditional code-centric test case development into modeling test architectures and test case behaviors, and using the design model to create more efficient and higher quality test cases, can result in better coverage of the requirements and ultimately higher-quality deliverables.

Creating model-driven tests

The IBM Rational® solution is based on the UML 2.0 testing profile, which integrates testing into UML, enhancing it with concepts such as test architectures and test behaviors. Test architectures extend the exiting UML 2.0 structural concepts to describe the elements involved in a test and their relationships. Similarly, the test behavior extends the existing UML 2.0 behavioral concepts to encompass all observations and activities during the test.[4]

Highlights

The IBM Rational solution offers a single model-driven environment where both design and testing can coexist.

The testing profile offers a taxonomy for testing artifacts that integrates well with UML, offering a single environment where both design and testing can coexist. All design artifacts, test artifacts and test reports can be integrated within the same browser. Design and test artifacts are always in sync and navigation between design and test artifacts is easy. Requirements can be linked to design components, which can then be linked to test cases and to test execution reports for comprehensive traceability (see figure 3).

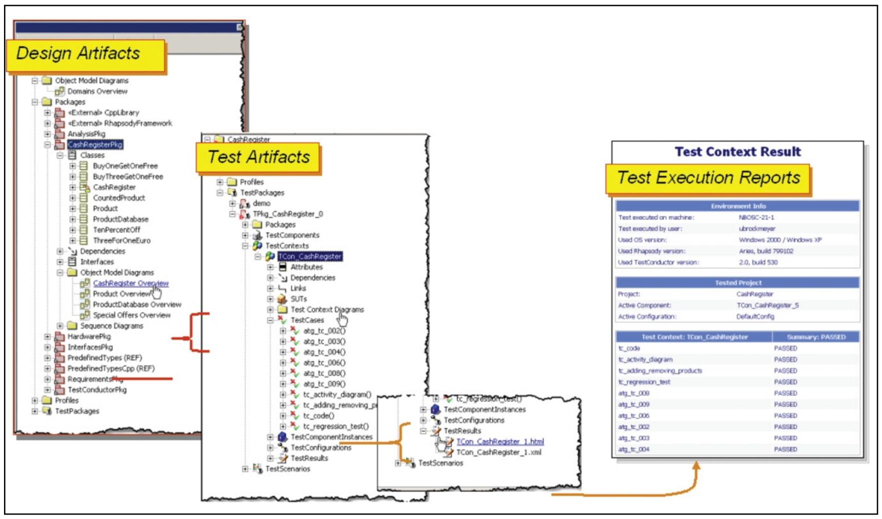

Figure 3: The UML testing profile enables model-driven testing where the design process and test process are fully integrated—providing an efficient approach to software validation and verification.

Highlights

Developers are able to design and test their models using the same diagrams.

In UML you design your applications through diagrams. The diagrams include sequence diagrams and activity diagrams. MDT with IBM Rational Rhapsody® software allows tests to be drawn as these diagram types, which means that the developer doesn't need to create a new language in order to build tests. The software can execute these tests against the design and provide comprehensive test results. The tests execute on the desktop and on the target. This can work quite well for developers. It may not, however, work as well for QA.

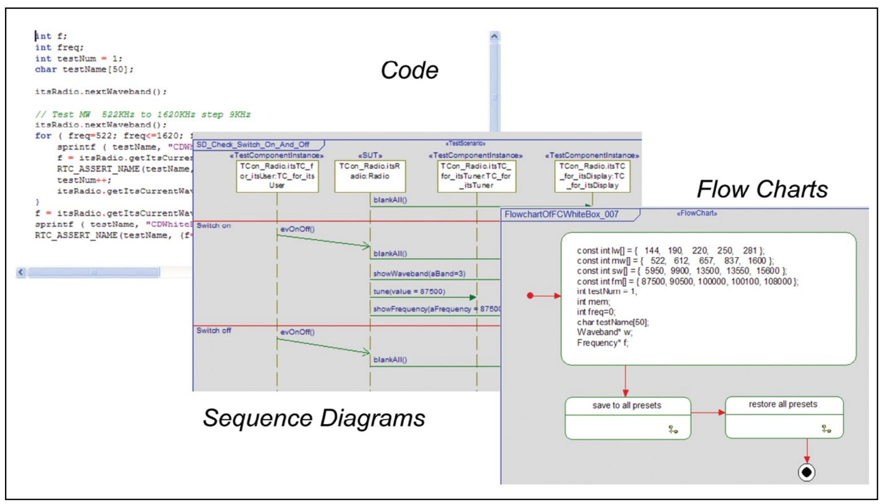

Figure 4: Model-driven testing with IBM Rational Rhapsody software enables you to test your models the same way you design them—using diagrams.

In a traditional MDT environment, these tests need to be executed and the results reported inside the UML tool. This means QA teams have been left out of the loop for executing and getting results from tests.

Highlights

With the IBM solution, the model tests created by developers can be stored in a central repository and executed by QA managers who are unfamiliar with UML.

The missing link: making tests available to QA

In view of the benefits, the primary reason organizations fail to adopt model-driven tests is because these tests are hard to reuse. Other people in the product development process don't know UML and they don't have access to the machines being tested. There is little incentive to spend time developing tests unless they can be used throughout the development process and product lifecycle.

IBM Rational Quality Manager software links with IBM Rational Rhapsody TestConductor Add On software to make tests available to QA. The integration between these tools allows tests created in Rhapsody TestConductor Add On software to be executed by people unfamiliar with UML. The integration provides a central repository to run tests from almost any source, store the results and enable access to those results.

Store and rerun tests from almost any source

QA teams have to test whether or not the product or system meets requirements from all sorts of different perspectives, not just software. Multiple teams are developing multiple components at any given time and creating tests. QA managers also write their own tests.

To realize the full benefit of MDT, QA managers should be able to reuse tests to make sure that completed system components work together as designed and meet business objectives. With the Rational MDT solution, both model-based and code-based tests can be stored in a central repository. They are always available for the QA manager to execute at any point in the development process and product lifecycle when it is necessary to validate that any functional additions and changes operate as expected and don't cause other parts of the system to break.

Highlights

The QA manager can view automatically generated test reports on the Rational Quality Manager dashboard and click on a failed test to navigate to exactly where the problem lies in the design model.

For example, a new component is ready to be added to a system under development. As part of the requirements validation process, the QA manager executes a range of tests from Rational Quality Manager software, which includes technology that is designed to identify and select the minimum number of tests required for a given level of coverage. Via the Rational Quality Manager dashboard, the QA manager can view automatically generated test reports that indicate what portions of the system pass and what parts fail. By clicking on a failed test, the QA manager can navigate to exactly where the problem lies in the design model and the associated code.

In this scenario, the test that failed is for a software component. Though executed from Rational Quality Manager software, the test is actually run in the Rational Rhapsody TestConductor Add On software. The appropriate developer is alerted automatically. The defect tracking functionality enables him to see all the ways the test failed and the context in which it failed (what the QA manager did that caused the failure.) And when the QA manager is notified that the defect is fixed, she can rerun the initial tests to see whether or not the system fails again.

Essentially, model-centric validation activities are used to drive code-centric, automated verification activities. The result is an integrated validation and verification testing process that leverages the model information and automates much of the code-centric verification activities.

Highlights

The IBM approach to model-driven testing can help your QA managers and testers realize productivity gains similar to those developers enjoy with model-driven development.

Integrating testing into the design process

Developers who have migrated from a code-centric approach to a MDD approach have been able to significantly increase their productivity. By leveraging the IBM approach to MDT, the QA and testing organization can realize similar productivity increases with the ability to:

- *Trace and easily navigate between requirements, design artifacts, test architectures, test cases and test execution reports, all from within a single browser.*
- *Link test cases, regardless of how they are captured, to their test architecture.*
- *Execute the same test cases on the host development platform and on the target without modifying them.*
- *Graphically monitor the progress of executed tests and graphically identify causes of failures.*

For most companies that depend on software to differentiate their products and services, it's really not a question of whether to transition to MDT. It's a question of when. With IBM, you can transition in steps and at a pace that makes the most sense for your business.

For more information

To learn more about how IBM can help you transition to model-driven testing, contact your IBM representative or IBM Business Partner, or visit:

IBM Rational Test Conductor

ibm.com/software/rational/products/rhapsody/developer/features
(Click on the "Validation and Testing" tab.)

Rational Quality Manager

ibm.com/software/awdtools/rqm/index.html

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
September 2009

1 Jerry Krasner, *The Economics of Defective Software*, Embedded Market Forecasters, July 2008.
2 National Institute of Standards and Technology (NIST), "Software Errors Cost U.S. Economy $59.5 Billion Annually," NIST news release, June 2002.
3 Martin Stockl, *Validation vs. Verification: When Resources Are Limited, Which One First?*, IBM Rational, March 2009.
4 Moshe Cohen, "Transitioning from Code-Based to Model-Driven Software Testing—Part 1: The Basics of the UML 2.0 Software Testing Profile," Embedded.com, November 2007.

RAW14161-USEN-00