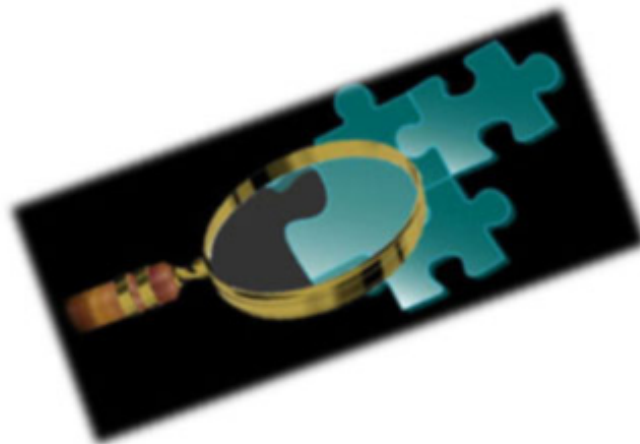




## Quality by Design: Enabling Cost-Effective Comprehensive Component Testing

by [Brian Bryson](#)  
Technology Evangelist  
Rational Software

*Comprehensive unit testing, a key strategy for infusing quality into the software development process, has not yet gained widespread acceptance. This article examines the obstacles impeding its acceptance and introduces new technology from Rational Software designed to overcome them.*



- ▶ [subscribe](#)
- ▶ [contact us](#)
- ▶ [submit an article](#)
- ▶ [rational.com](#)
- ▶ [issue contents](#)
- ▶ [archives](#)
- ▶ [mission statement](#)
- ▶ [editorial staff](#)

Last spring, one of my co-workers learned that his car was being recalled for some component defects. As it turned out, this proved to be little more than an inconvenience for him. He took the car to the dealer, the dealer replaced the defective components at no cost, and he got his vehicle back a few hours later. No harm done, save for wasting a few hours at the dealer instead of being chained to his desk. From the automotive manufacturer's point of view, however, the problem was much more serious. More than 16,500 owners were affected by that recall, and the total cost to the organization would eventually exceed \$114 million dollars. Oops.

It's hardly a stretch to conclude that the manufacturer could have saved a ton of money by finding that defect a little earlier in the manufacturing process. Even if they'd detected it after all the cars were assembled but still not shipped, the savings would have been significant. Plus, they could have avoided most of the embarrassment, customer frustration, and ill will. Now think how much they they could have saved had they discovered the problem on the drawing board . . .

### Quality by Design and Comprehensive Unit Testing

The Quality by Design approach to producing a product encompasses a series of strategies, processes, and practices that infuse quality into the

early stages of development. It is neither new nor unproven. It was used to build the Boeing 777 aircraft, for example. More than 90% of the testing for this plane was completed against computer design models. Only one prototype was actually constructed before the first aircraft was assembled, and savings were in the millions.

In his *Software Project Survival Guide*, published in 1997, author Steve McConnell validates the Quality by Design approach by reporting that fixing defects before coding costs 50-200 times less than fixing those found either at or after product delivery time. These are impressive numbers, and they attracted a lot of notice in the business world. So why don't we all pursue a Quality by Design approach? In fact, why haven't we been doing so for years?

These questions are too complex -- and Quality by Design is too comprehensive a topic -- to cover in one short article. But we can begin to understand both the approach and why there has been great resistance to adopting it by looking at one aspect of Quality by Design in relation to software development: comprehensive unit testing.

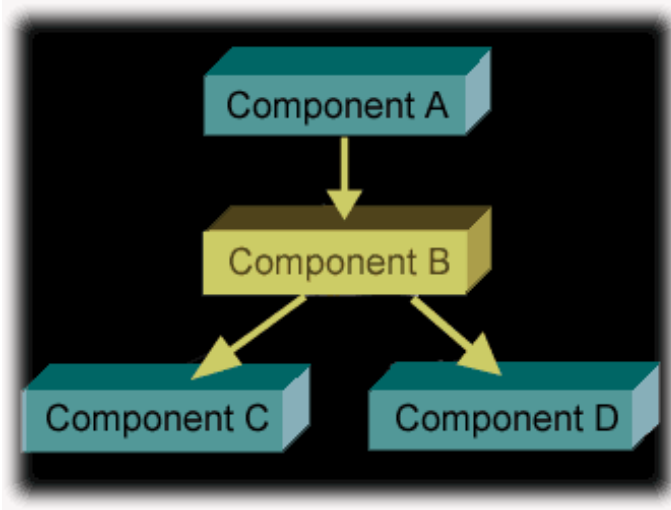
## **Removing Obstacles to Comprehensive Unit Testing**

Although software developers everywhere acknowledge that there are tremendous benefits associated with comprehensive unit testing, the practice is far from commonplace, especially for testing middle-tier, GUI-less components. Why? Because it is time consuming and tedious. And in the past, the costs of overcoming these obstacles frequently outweighed the benefits. One big problem is that most tests are tailored for a particular component; there is little opportunity for re-use. Development teams are under extreme time pressure, so they feel obliged to focus on developing the application itself to stay on schedule. Typically, they view the process of building test harnesses and stubs from scratch, using them, and then throwing them away project after project as wasteful. So they focus their limited resources exclusively on writing code for components rather than on testing them.

Fortunately, it doesn't have to be that way. Rational has just introduced a new technology that enables cost-effective comprehensive component testing. Rational QualityArchitect is a part of Rational's continuing efforts to provide developers with the tools they need to deliver higher quality software in less time. Rational QualityArchitect eliminates the most tedious and difficult aspects of component testing by leveraging the knowledge captured in visual models that automatically generate test code. Developers can focus on the actual test cases they need to create instead of spending time writing error-prone, throwaway test code.

## **Component Testing *Without* Rational QualityArchitect**

To better understand how this new product makes comprehensive unit testing more achievable, let's take a closer look at some of the challenges of testing components *without* QualityArchitect.



**Figure 1: Four Components for Testing**

Figure 1 depicts four untested architectural components. Consider the implications of developing tests for Component B, which is currently ready for testing. Quite likely the other components (A, C, and D) are not yet ready for testing, and even if they are, they may contain defects that would blur the test results and make the job of tracking down problems in Component B that much more difficult. For these reasons, developers typically write their own

throwaway test drivers and stubs.

Now consider the complexity of the test driver requirements. A test driver to simulate the behavior of Component A must drive Component B, make calls into it, provide a range of inputs, and record responses. Meanwhile, all of the functions in Components C and D that Component B relies on must be stubbed out, and they must return appropriate results based on the input from Component B. Sounds like a complicated recipe for alphabet soup, doesn't it?

In addition, even after tests are completed for individual components, there are still significant challenges to overcome. In scenario testing, two or more components are brought together to test a given sequence of calls. If the client software does not already exist, developers need to spend time creating a mock client to drive the scenario. Some development teams report that they spend more than half of their entire development time creating these test harnesses, which are seldom reusable.

## **Component Testing *With Rational QualityArchitect***

Rational QualityArchitect provides the key to cost-effective comprehensive unit testing: leverage the artifacts developers created earlier in the process -- their visual models -- to generate test harnesses. Once developers know how each component has to behave, they capture that knowledge in a model. The visual models that developers can produce with Rational Rose are especially powerful in this respect, as they are used to generate the code for these components automatically. That's why Rational QualityArchitect is packaged as a component of Rational Rose Enterprise.

For unit testing, developers need to accomplish three objectives:

1. Test the individual methods of a single software component.

2. Test multiple methods across multiple components in sequence.
3. Generate stubs for incomplete or non-existent components so that the testing of one component is not reliant upon the existence of other components.

Each of these three tests, in turn, is made up of two components:

- A. The test harness or skeleton code that drives the test.
- B. The test case data.

That's it. Simple, no? Let's look at an example to make this a little more concrete. Say I'm going to test a single Enterprise JavaBean (EJB) component. I need to do two things:

- First, create all the test code (A) that will connect to the server where the EJB resides, then instantiate the bean, call the operations on the bean, and validate the returned results.
- Then, create the test data (B) that is used when calling the individual operations.

Although creating the test data is more challenging, creating the test code is actually more time consuming -- and tedious.

But remember: All the information needed to create the test code already resides in my visual model. The visual model contains a structural description of the component and its operations, plus its operation's arguments and return value types. Whether the visual model was created by analysts and developers during the design stage or reverse engineered based on live components is irrelevant. All the inputs for creating test skeleton code are there for me.

This is where Rational QualityArchitect actually takes over. By examining the structure of a component as set out in a visual model, it can generate all the test code necessary for a single component or for a test involving a sequence of operations across multiple components. QualityArchitect can even generate stub components to act as placeholders until components in development are deployed.

The test code, however, is only half of the solution. I still need test data. Here, QualityArchitect makes my job as a developer easier. No longer chained to the tedious process of having to create test code, I can focus attention on creating interesting and meaningful test data.

QualityArchitect can even help out by generating random test data where specific cases aren't necessary. Where specific test data *is* necessary, QualityArchitect provides a simple spreadsheet-like interface for me to enter the data.

## **Cost Savings, Time Savings, and No Recalls**

*Without* Rational QualityArchitect, the early testing required for

comprehensive unit testing is such a time-consuming and inefficient undertaking that, despite its obvious value, many organizations forgo it. *With Rational QualityArchitect*, early testing becomes truly feasible, because QualityArchitect generates test harnesses and stubs automatically -- and not just once, but incrementally, as the model evolves throughout development. For a developer doing unit testing, Rational QualityArchitect virtually eliminates the time-consuming work of creating throwaway code, leaving only the simple task of populating data into a spreadsheet.

Most important, however, is that all of this testing is done off the visual model, at the very earliest stages of development. In effect, by leveraging pre-existing assets for testing operations, Rational QualityArchitect enables a development team to adopt a Quality by Design approach without stealing a lot of time from basic development work.

Although recalls are not a common practice in the software industry, the consequences of a defective software system can be even more expensive - - and just as devastating -- as those of a defective automobile. The goal of using a Quality by Design approach for every software project is worth striving for. Early testing might have saved a certain auto maker \$114 million, and it can yield similar savings for companies that create ambitious software systems. Boeing has already proved that you can safely test an entire aircraft based on computer designs. Rational QualityArchitect ensures that you can do the same for complex software systems.

---

***For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided. Thank you!***