

Robust



SearchSoftwareQuality.com

Helping you develop, deploy and manage quality software.



SearchSoftwareQuality.com

Helping you develop, deploy and manage quality software.

[NEWS](#)

[WHITE PAPERS](#)

[WEBCASTS](#)

[CAREERS](#)

[GLOSSARY](#)

[DOWNLOADS](#)

[EXPERTS](#)

[RSS](#)

[MODELS/METHODOLOGIES](#)

[REQUIREMENTS](#)

[TESTING AND QA](#)

[MAINTENANCE](#)

[PROJECT MANAGEMENT](#)

SEARCH

[Advanced](#) | [Site Index](#)

Search Powered by



ADVERTISEMENT

[Home](#) > [Software Quality Tips](#) > [Software Testing](#) > Model-based testing for Java and Web-based GUI applications

[EMAIL THIS](#)

Software Quality Tips:



[TIPS & NEWSLETTERS TOPICS](#)

[SOFTWARE TESTING](#)

Model-based testing for Java and Web-based GUI applications

Bill Hayduk
03.19.2007
Rating: --- (out of 5)

Ideally a well-designed automated software quality assurance test will explore all the ways that every feature in a piece of software can be used. It will also reveal how each feature can potentially interact with all other features, exposing all the flaws and bugs that lurk within the programming code. But as you may have noticed, we don't live in an ideal world.

Standard tests repeat a predetermined series of investigations, discovering the flaws exposed by those specific tests. Model-based testing takes a somewhat different approach. Typical implementations use algorithms to figure out all of the usage paths for an application to pare down the number to get a maximum in coverage and a minimum of tests, and then to generate test cases to try the application against.

Skilled programmers work hard to ascertain all of the ways a user might work with an application, but sometimes users surprise programmers. Model-based testing can help, as it is like having a vast pool of users testing the application 24/7 for weeks, with each working in his own idiosyncratic manner. Since you can count on users to interact with the application in ways that the programmer of the automated test didn't take into account, model-based testing can catch flaws that standard tests might miss.

Because model-based testing generates a broad sampling of test cases, it avoids the fallacies potentially inherent in repeating the same narrow run of tests. Repeating the same run of tests can generate a false sense of security, as subsequent runs will naturally uncover fewer and fewer flaws in programming code that may harbor serious problems.

Potential pitfalls in model-based testing

Model-based testing isn't suitable for every application and every situation. These testing tools examine an application's actual behavior in response to actions. In order to develop the inputs that the tool will use to generate test cases, we first have to think about what the application should optimally do and how it should respond to its users and perhaps interact with any other applications. Then we need to describe that behavior to the testing tool.

The inputs to the tool are highly critical, since everything that follows will be based on them. If the inputs are incomplete, do not follow the proper syntax (and therefore are not properly processed by the tool) or are wrong, then everything that follows will also be wrong.

Companies and developers whose work process doesn't include creating complete documentation for applications (and who aren't committed to updating and fully maintaining that documentation throughout the entire development process) may struggle when confronted with the need to define the correct inputs. They may find that creating the inputs properly is impossible to do correctly or is simply too time-costly. Only you can determine whether model-based testing suits your project or company's culture.

It's also important to remember that model-based testing is just one tool in the software assurance-testing arsenal; it's not a replacement for standard automated testing.

Points to consider

Developing input for a Web-based application's GUI is complicated by the fact that every user will potentially interact with the application in a unique way — and some will examine the application specifically to discover flaws to exploit.

Additionally, GUIs in Web-based applications and Java application testing must take into account cross-platform concerns, the applications used to access Web-based applications, configurations, and possible existing flaws in users' Web clients and operating systems that when run in conjunction with the tested application can produce an exploitable flaw that didn't exist in the application alone.

When starting to develop your inputs, you'll also want to think about how the application is intended to be used, how it can be used (not always the same as the intended use) and how it can be abused. And since a big part of a Web-based application's success will be judged by its users on performance issues, consider testing scalability and reliability as well as looking for security flaws.

One of the benefits of model-based testing can be more thoughtful programming, since programmers and/or testers must understand in depth how an application is intended to work in order to develop model-based tests. Model-based testing can create more of an awareness of how actions interact and what effects these interactions have on code. Thinking about how to test an application can result in better coding.

Thankfully model-based tools don't require programmers/testers to define every possible interaction a user might make; instead you define the states, actions and transitions for your application. The tool then generates test cases. Be aware that your tests will only be as good as your input. Garbage in, as always, equals garbage out.

Next you'll need to set initial test objectives. Since there are potentially an infinite number of tests that can be run in model-based testing, the best approach is to risk-prioritize the work. That means critical application areas are designated as highest risk, medium risk, etc. in descending order of perceived risk.

Remember that different stakeholders in the project usually have different ideas about risk prioritization and the prioritization process may take longer than you anticipate. Since many developers wait to test an application's GUI until after the code base of the application has stabilized, make sure to factor in the necessary time to agree on risk priorities and develop the inputs. As you run the tests, analyze the results and fix bugs, you'll continue to modify the model until you've met your testing objectives.

Model-based testing is not the proverbial magic bullet, nor will it necessarily save time or money over standard automated tests. Its value is its ability to generate non-repetitive, real-world tests, assuming that good inputs can be produced within a project's time and financial constraints.

About the author: Bill Hayduk is founder, president and director of professional services at [RTTS](#). Over the past 15 years, Bill has successfully implemented large-scale automation projects at many Fortune 500 firms. He has managed projects in most verticals, including banking, brokerage, multimedia, ISVs, government, telecommunications, healthcare, education, pharmaceutical and insurance.

Software testing resources

[User acceptance testing and test cases](#)

[Misuse cases: Understanding the hacker's approach](#)

[How to evaluate testing software and tools](#)