A Conformiq White Paper

Why Automate Test Design?

The number of software applications, customer service portals, device types, and platforms has reached an all-time high. The need for reliable and efficient testing methods is more critical than ever before. Testing complexity and requirements are growing exponentially. Yet, many of today's testing environments continue to use test design and test execution methods, dating back 20 years or even more. Businesses refusing to seek out and implement next generation testing techniques run the risk of extinction by their competitors who are. Test automation has been primarily focused on automating test management and test execution. Test design still remains largely a manual activity. By also automating the test design process, functional testing efforts can be significantly reduced while at the same time the quality of the testing can be increased. In this paper we will compare and contrast the primary testing methods and the benefits of automating your test design.

Traditionally, test automation has been mainly focused on automating test management and test execution. Unfortunately, test design often remains a manual activity. The test design itself concerns making decisions regarding:

- What to test and what not to test
- How to stimulate the system and with what data values
- How the system should react and respond to stimuli

Therefore, test design is a separate task from test execution. It is done before executing tests against the system.

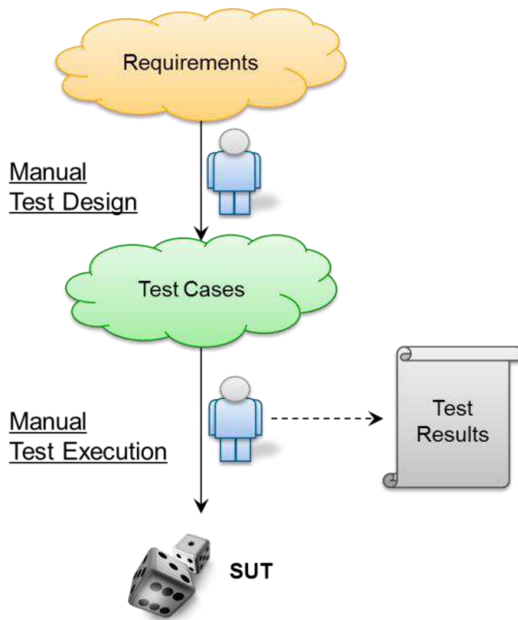
Even today, automated tests are often created and executed only for regression – not to find defects in the new functionality. Traditional and manual test design and manual test execution are still prevailing approaches for testing new functionality.

By also automating the test design, testing efforts can be significantly reduced, while the quality of the testing can be increased at the same time.

Manual Testing Process

In order to see and understand why test automation is valuable and increasingly necessary, let's first take a look at the manual testing process Manual Testing is the earliest form of testing, but it's still widely used today.

Copyright © Conformiq Inc. and its subsidiaries 2015. All Rights Reserved. All information in this publication is provided for informational purposes only and is subject to change without notice. Conformiq, Conformiq Designer, Conformiq Creator, and Automated Test Design are trademarks of Conformiq Inc. Other trademarks and registered trademarks belong to their respective owners.



The *test design shown is done manually* based on informal requirements documents. The test designer goes through the requirements document and manually creates test cases for testing an implementation that is based on the same set of requirements.

The output of the manual test design step is a document that describes the desired test cases. With the test cases, *test execution is done manually*. A manual tester follows the steps of the test cases and interacts directly with the SUT (System Under Test, i.e., the software to be tested) by comparing the values of the SUT output with the ones expected, finally recording the test verdict.

In order to carry out test design, the test designer needs to possess expert knowledge about the SUT and needs to have test design strategy skills. Manual test execution requires less talent, but the ability to follow the steps of the test cases and knowledge about how to interact with the SUT are vital.

The main benefit of the manual testing approach is that it's easy to start with and the initial cost is low.

However, as everything is done manually, there are numerous shortcomings that can be divided into two groups – first, the ones related to test execution, and second, those related to test design.

When looking at shortcomings of the test execution side, the biggest and most severe issue is that there is no automated regression testing, meaning that the whole process needs to be repeated when the system changes. This quickly becomes an error-prone and time-consuming activity. In fact, the process is so costly and time consuming that it often forces teams to cut corners and sacrifice the quality of their testing.

The second set of problems stems from the fact that test design is done manually. Aside from the time factor, the results using manual test design are difficult to reproduce. Because everything is done manually, there is no systematic way to understand functional coverage, it is difficult to judge the progress of testing and also, the quality and coverage of the produced test cases. There is no automatic way to link the requirements, so requirement traceability and coverage is either omitted or established manually.

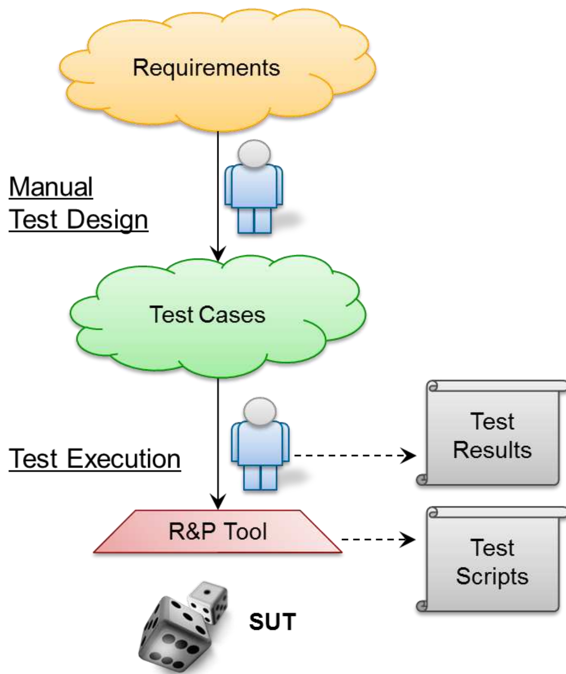
In short, the manual testing process simply cannot scale. As such, manual testing processes are increasingly unable to meet the demands associated with today's testing realities.

Record And Playback

A purely manual testing process can be improved by *automating test execution*. The record and playback method attempts to reduce the time and cost of test re-execution by *recording the interactions with the SUT during the first test execution session and then enabling a playback of the recorded test scripts* so that they can be re-executed at later time.

There are two types of record and playback tools. With the first, the user manually executes the test cases against the SUT, recording the steps for reuse. In the second, the operation of the as-built system is captured and test cases are generated to test it.

Copyright © Conformiq Inc. and its subsidiaries 2015. All Rights Reserved. All information in this publication is provided for informational purposes only and is subject to change without notice. Conformiq, Conformiq Designer, Conformiq Creator, and Automated Test Design are trademarks of Conformiq Inc. Other trademarks and registered trademarks belong to their respective owners.



Initial test execution is a similar activity to the one in a completely manual testing process – the difference being that now the interactions with the SUT are also recorded. When the system changes, we actually have something that we can try to run against the system – the recorded test scripts.

As with the manual testing process, the record and playback approach is easy to use and the initial cost can be low. As the interactions are recorded, one can trivially replay the recording to allow for re-execution of the test scripts for "free."

The main problem with applying record and playback to automate the re-execution of the tests is that it is *extremely fragile with any changes in the SUT.* This inability to adapt to small changes in the SUT often forces test designers to re-record test scripts when there is a small change in the SUT, creating a huge maintenance problem. The problem is so severe that these solutions are often abandoned after a couple of new revisions. Some of the record and playback tools try to alleviate this problem by enabling the elevation of the level of abstraction of the recorded test scripts and allowing one to make changes to them. For example, you can use and create placeholders in the recorded scripts which then can be filled during execution from a data table.

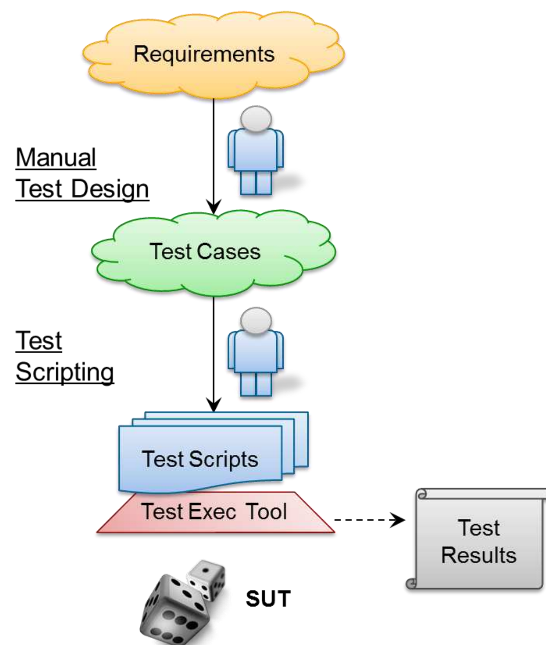
The second record and playback method overcomes many of these issues by automatically capturing the system operation every time it is rewritten. These tools execute the application capturing the operation and generating test cases to test this operation. However, the big issue with these is that the test cases just test the "as-built" application. At this point, it is too late to make design changes, but even more importantly, this method only tests that the application works as it was developed – not how it was intended according to the specification, which might be a significant difference.

Although record and playback aims to address only the problem of re-executing tests, it suffers from the same shortcomings as the manual testing process.

In practice, record and playback is not an attractive approach for addressing test automation because it delivers only limited efficiency gains over manual.

Scripted Testing

In a scripted testing process, automating the writing of test scripts solves the test execution problem.



Copyright © Conformiq Inc. and its subsidiaries 2015. All Rights Reserved. All information in this publication is provided for informational purposes only and is subject to change without notice. Conformiq, Conformiq Designer, Conformiq Creator, and Automated Test Design are trademarks of Conformiq Inc. Other trademarks and registered trademarks belong to their respective owners.

Instead of directly interacting with the SUT, test designers write a collection of executable test scripts, each containing one or more test cases. These test scripts are automatically executed against the SUT. They stimulate the system with tester selected input values. Test scripts can be implemented in many scripting or programming languages and then executed on a framework that can read scripts in the chosen language. The test execution tool records output values, compares observed values against the tester created expected values, and creates a test verdict.

However, since test scripting is a programming task, test designers need to possess or develop additional skills beyond the skills required for manual test design and test execution.

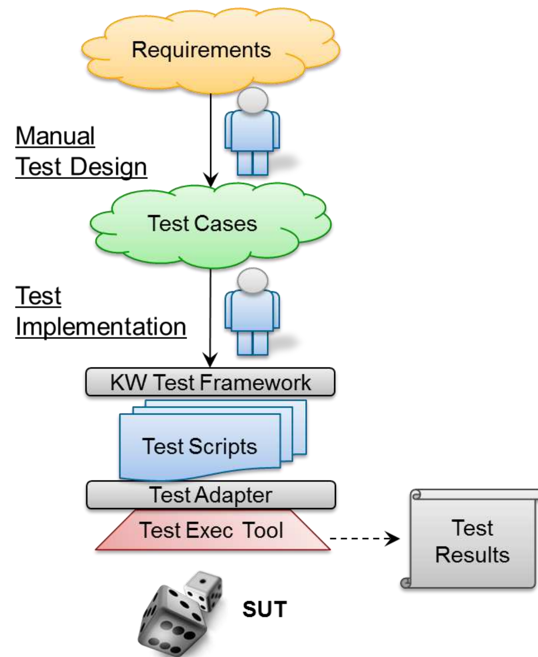
As test execution is automated using test scripting, the initial testing can be run by using the automated scripts. Regression testing can also be done for "free" by simply re-executing the test scripts.

One of the biggest shortcomings of this approach is that scripting is a complex activity that requires a lot of time and effort. But what is even worse is the maintenance problem that scripted approaches have. This stems from the fact that test scripts need to be updated not only when requirements change, but also when implementation details change. How much time and energy is then spent on maintenance depends on the abstraction level of the test scripts. And, again, because test scripting requires a certain level of programming knowledge, implementing nicely abstracted test scripts demands some advanced skills from the test designer.

Although scripted testing focuses on addressing the test execution automation problem, it suffers from similar shortcomings as the manual testing process, including the risks and costs associated with the manual test design, ad hoc coverage, and manual traceability.

Keyword Driven Testing

In order to overcome the maintenance problem introduced by scripted testing, the *abstraction level of the test cases can be elevated* using keyword driven testing.



The main idea of this process is to express the test cases in as abstract a form as is possible, while still providing enough details so that they can be readily executed against the real system.

In **data driven testing** or **data table testing,** there are sets of abstract test cases that do not fix the data values. Instead, the data values are read from a data table during test execution. This allows for the reuse of the same test scripts for testing the system with multiple data values. This will obviously reduce maintenance efforts.

In **keyword driven** or **action word testing**, this concept is taken a bit further and it abstracts the test steps in the test cases by introducing keywords or action words that correspond to some well-defined fragment in the test scripts. This allows non-programmers to implement test cases simply by constructing them using these action words. The action words are mapped to actual test code by a

Copyright © Conformiq Inc. and its subsidiaries 2015. All Rights Reserved. All information in this publication is provided for informational purposes only and is subject to change without notice. Conformiq, Conformiq Designer, Conformiq Creator, and Automated Test Design are trademarks of Conformiq Inc. Other trademarks and registered trademarks belong to their respective owners.

keyword driven testing framework, and the test code needs to be implemented by engineers who can do programming.

The main benefit of keyword driven testing is that it allows engineers to work at more abstract and concise level. Also, non-programmers can implement the tests. As the keywords map to executable code fragments, keyword driven testing offers the same benefits as scripted testing, namely automatic execution of the test cases and automatic regression testing. An additional benefit is that the maintenance efforts are reduced compared to scripted testing because of the possibility of more reuse and abstraction.

However, the test data and test oracles are still designed manually. In addition, test coverage with respect to requirements and traceability, like with all the other approaches introduced so far, needs to be completed manually.

The Next Step...

So what is the next step? How should we solve the problems with the current test automation solutions? All these approaches rely on manual test design, and therefore none of them guarantee systematic and repeatable coverage of the system behavior. This non-repeatability is a huge risk. With manual test design, it is difficult to assess the quality of testing efforts, which often leads one to evaluate the quality and progress of the manual test design process using fairly meaningless metrics, such as number of test cases or number of hours spent on testing.

At the same time, manual test design is also a very expensive process - especially when there are changes in the requirements. In practice, test designers are forced to manually analyze each test case individually in order to see which test cases need to be updated, which need to be removed, and which need to be added in order to fill the coverage gap when there are changes in requirements. This decreases productivity and increases the risk of error. Finally, the requirement tracking in all of these approaches is done manually.

Automating the Test Design

Traditionally in the test design phase, test designers and Subject Matter Experts (SMEs) form an understanding of the system using specification and requirements. In essence, they form a **mental model**. This mental model is *not one of tests, but of the system itself*. In a purely manual test design process, this mental model of a system is turned into test cases in the mind of the test designer. This is an implicit, creative process that is not reproducible and is bound to the ingenuity of individual engineers. If you lack sufficient talent for doing good test design, you're out of luck as a tester and for the project.

As test engineers form a mental model of a system, it makes it seem that test design can be automated by making this model explicit, for example, by expressing this mental model in a form that is understood by a computer which could then generate test cases from this explicit model.

Model Based Testing

Now when we have a computer readable model, we can apply **model based testing** to the problem of test design automation. Model based testing is currently trending and can provide a variety of approaches. In loose terms, model based testing is anything that is based on computer readable models that describe some aspects of the system to be tested in a format and with accuracy that enables either completely automatic or semi-automatic generation of test cases.

The three main approaches to model based testing are 1) **graphical test modeling**, 2) **environment model driven test generation**, and 3) **system model driven test generation**, There are others but, these three are the main approaches.

All model based testing approaches listed above can produce the same end result – that is, they can all be used to generate executable test cases and test

Copyright © Conformiq Inc. and its subsidiaries 2015. All Rights Reserved. All information in this publication is provided for informational purposes only and is subject to change without notice. Conformiq, Conformiq Designer, Conformiq Creator, and Automated Test Design are trademarks of Conformiq Inc. Other trademarks and registered trademarks belong to their respective owners.

documentation. However, the key factor is *what the users need to do in order to get those tests out*.

Graphical Test Modeling

The graphical test model is the simplest of the approaches listed above. It is nothing more than *modeling the test cases in a graphical notation*. Graphical test case modeling aims to provide similar benefits to keyword driven testing by elevating the level of abstraction, which enables more reuse, reduces maintenance costs, and increases productivity. The tools then turn these abstract graphical test cases into executable test scripts.

The models that capture graphical test cases are easy to understand and the complexity to create them is low. Therefore, the approach may appeal to non-programmers, as graphical test case modeling does not require programming skills, which is quite often expected for other model-based testing approaches.

However, as the test cases are modeled, the only thing that we are actually automating is the creation of executable test scripts. Therefore the value proposition is similar as with keyword driven testing. No test cases are created beyond what the modeler thinks of and, when the design changes, the manual effort of remodeling is the same as the original effort.

Environment Modeling

Environment, use case, or usage models describe the *expected environment of the SUT*. These models describe how the system under test is used and how the environment around the system operates. These models represent the tester – not the system that is being tested. The models include *testing strategies* (the input selection) and hand crafted output validators (*test oracles)*.

For example, if we are testing an application running on a handheld mobile device, the environment constitutes the user who uses the device and the radio network. An environment model describes how the environment –the user and the network – operate with respect to the application, including the details about testing strategies and output validators.

This style of modeling is similar to the traditional thinking of testers since these models essentially capture the operations of the tester. The models, however, are more complicated than simple graphical test case models because of the extra expressivity.

Because test generation algorithms for environment models are well known and easy to implement, the tools are relatively robust and efficient. There are a lot of different tools available, both free and commercial, and companies often even create their own tools for generating tests from environment models.

These tools eliminate the need for manually writing test scripts, and some of the tools even allow for the annotation of the model with requirement links, enabling automatic tracking of requirements, which is a highly important and valuable feature. The fundamental problem is that the test design is still left as an exercise to the test engineer. The test engineer needs to manually describe the testing strategies and the test oracle, that is, the stimuli that is needed to send to the system and the expected output from the system under test.

System Modeling

The third main approach to model based testing is called system model driven test generation. Here the idea is that the *model represents the actual, desired behavior of the system itself*. This means that the system model is the mental model that test engineers form, while going through the requirements documentation now made into an explicit model. The model describes how the system should work – not how it should be tested. For a moment, let's go back to our previous example about an application on a handheld device where the application operates with a user of the application and the radio network. As opposed to other approaches in system modeling, the focus is on the behavior of the application itself. We do not focus on how the user utilizes the device or

Copyright © Conformiq Inc. and its subsidiaries 2015. All Rights Reserved. All information in this publication is provided for informational purposes only and is subject to change without notice. Conformiq, Conformiq Designer, Conformiq Creator, and Automated Test Design are trademarks of Conformiq Inc. Other trademarks and registered trademarks belong to their respective owners.

how the radio network operates. Instead, the focus is on the correct behavior of the application on the device. We model the behavior of the application on a high level of abstraction and then leave the problem of test design to the computer. The computer is responsible for figuring out how the environment outside the application operates. The computer figures out how the user should stimulate the application, what kind of interactions should be on the interface of the radio network, and what kind of precise output the application should give to the user – the test stimuli plus the test oracle. Therefore, in the case of system modeling, the *computer generates an environment that drives the real system*.

There are two, somewhat contradictory goals when making a system model. The first is that the model should be smaller and more abstract than the real system – otherwise it takes too much time and money to describe one. It should focus on the key aspects that that are to be tested and omit many of the details of the SUT. The second goal is that it needs to be accurate enough to capture the details that need to be tested.

Creating a system model is a more straightforward and less error-prone process than modeling the test cases themselves or creating an environment model. This is because the mental step involved in designing the testing strategies and oracles is omitted. The modeling process can be compared to a translation problem – the *goal is to translate the specification and/or requirement documents into a computer readable format.* Once that is done, the model is very easy to update when requirements change. This type of model is also much easier to understand by stakeholders and to use as reference for developers. This is a huge time saver for test maintenance.

Conclusion

As we have seen, traditional test automation focuses mainly on two aspects: test management and test execution. With these solutions, the process of test design – the process of deciding how to test, what to test, and what not to test – is a manual activity. Manual test design introduces many risks and takes a lot of time, especially when requirements change.

System model driven test generation is an effective and complementary way of addressing the shortcomings of existing test automation.

First, it automates the design of functional test cases to reduce the design cost and to increase the quality.

Second, it reduces the maintenance costs of tests.

Lastly, it automatically generates coverage reports and traceability information from requirements to the tests and back.

System model driven test generation offers significant benefits in terms of improved quality, improved SUT fault detections, improved traceability, improved maintenance, improved model reuse, reduced cost and time, and improved requirements.

MBT is a more sophisticated approach to testing than earlier generations of testing tools. Operating with these tools requires a different mind and skill set than more traditional testing tools. However, the results of using this process show that proper training and experience, along with a willingness to make the change succeed, can overcome these hurdles.

So once you pass these initial hurdles and start to see the benefits, you will never go back.

The author of this paper, Kimmo Nupponen, has been developing automated test design software for over ten years. He understands what is really needed for real world use and the “under the hood” differences between MBT tool engines. He is the Chief Scientist at Conformiq.

Copyright © Conformiq Inc. and its subsidiaries 2015. All Rights Reserved. All information in this publication is provided for informational purposes only and is subject to change without notice. Conformiq, Conformiq Designer, Conformiq Creator, and Automated Test Design are trademarks of Conformiq Inc. Other trademarks and registered trademarks belong to their respective owners.

www.conformiq.com

4030 Moorpark Ave	Westendintie 1	Stureplan 4C	Maximilianstrasse 35	29 M.G. Road Ste. 504
San Jose, CA 95117	02160 Espoo	SE-11435 Stockholm	80539 Munich	Bangalore 560 001
USA	**FINLAND**	**SWEDEN**	**GERMANY**	**INDIA**
Tel: +1 408 898 2140	Tel: +358 10 286 6300	Tel: +46 852 500 222	Tel: +49 89 89 659275	Tel: +91 80 4155 0994
Fax: +1 408 725 8405	Fax: +358 10 286 6309	Fax: +358 10 286 6309	Fax: +358 10 286 6309	

v0715

Copyright © Conformiq Inc. and its subsidiaries 2015. All Rights Reserved. All information in this publication is provided for informational purposes only and is subject to change without notice. Conformiq, Conformiq Designer, Conformiq Creator, and Automated Test Design are trademarks of Conformiq Inc. Other trademarks and registered trademarks belong to their respective owners.