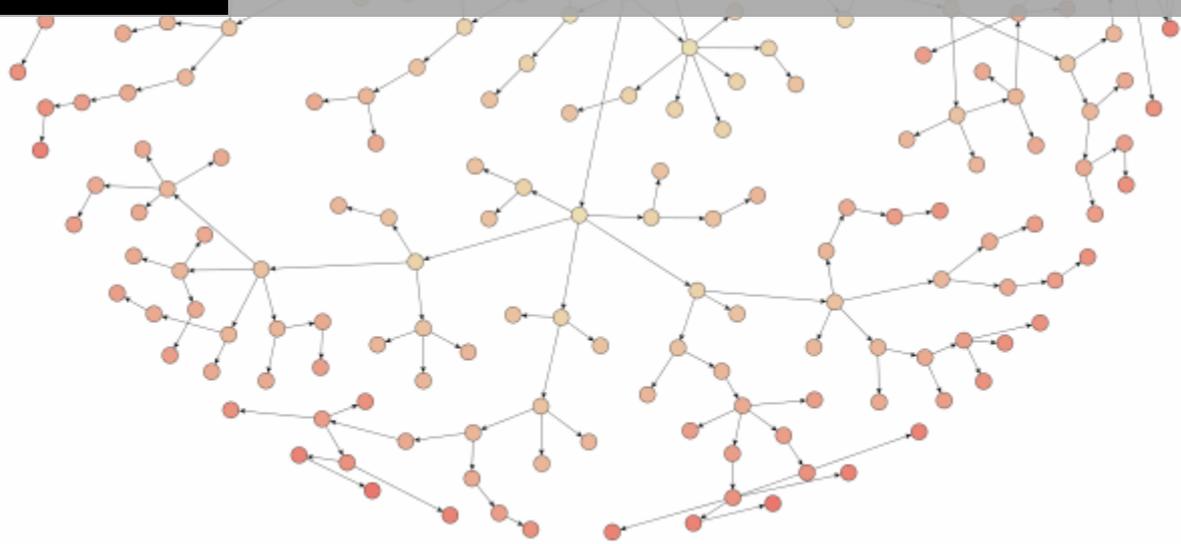# prolore

WHITE PAPER

MODEL BASED TESTING

The answer to the Agile Testers pray

About the author

Daniel Andersson started with test automation 1996 and has extensive experience from different tools, environment and businesses. Daniel has also developed several test automation courses and seminars.

Introduction

This white paper discusses the Model-based testing (MBT) and its use primarily from an automated test perspective. It aims to introduce the test automation specialist (and other test professionals) to MBT. It is intended to be clear and straightforward in a way that also enables management to get the big picture.

So what is Model-based testing?

According to Wikipedia it is:

"Model-based testing is software testing in which test cases are derived in whole or in part from a model that describes some (usually functional) aspects of the system under test (SUT)."

Although several definitions exists, most of them carry the same basic idea about an model describing a system of some sort and some way of using that model to get test cases (or rather test sequences).

Why is it gaining so much popularity?

If you attended the major test conferences for the last years you probably noticed that MBT has been on the agenda for some time now. The interesting thing is that the topics regarding Model-based testing have shifted from "New and Hot" to more mature reasoning and something that is self-evidently logical and commonplace.

Listening to the representatives from the biggest software houses we got on this planet, talking about Model-based testing, you immediately realize that this is now second nature in their organizations and causes no more stir than words like Agile or Virtualization.

Now, many of us is not there yet and there is all reasons for drilling down into why Model-based testing is the shining star in the test automation skies:

- Model-based testing is easy to understand from both the business and developer communities
- Model-based testing separates (business-) logic from testing code
- Model-based testing increases the test coverage with the same effort as "Classic" test automation
- Model-based testing is the fastest way to get use of automated test
- Model-based testing enables us to switch testing tool if needed or support multiple platforms using the same model
- Model-based testing focuses on requirement coverage, not how many test cases you executed last week etc.
- Model-based testing proved to positively affect the maintenance problem, the nemesis of all test automation.

Prolore AB   Page 2 (6)   Web: www.prolore.se
Västmannagatan 43   E-mail: info@prolore.se
113 25 STOCKHOLM   Org Nr: 556683-3421

How does it work, very briefly?

For simplicity and understanding throughout this paper consider the following simple model:
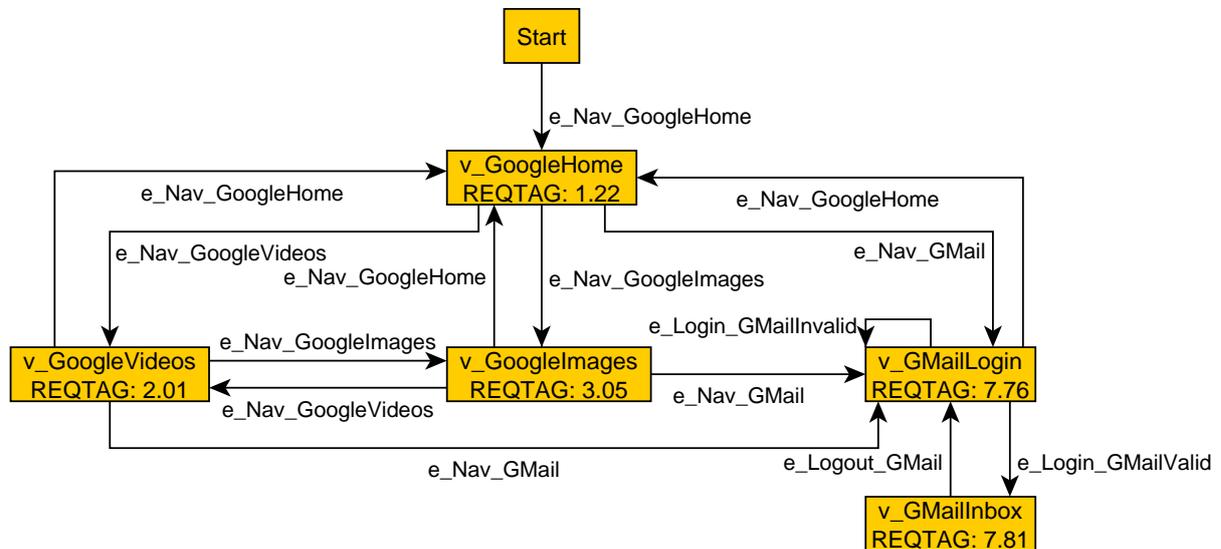
The model describes the functionality of a system, or rather the functionality we want to test. You see the possible paths through the system visualized by arrows (aka edges) and nodes (aka vertexes). Traversing the model is quite simple; do something in an edge and verify that your actions turn out right in the following vertex, and on to the next one….

OK, so how do you draw a model and then what? Well, we need some tools… For simplicity and making the budget process easy we start with free tools. The tool chain used in this paper will be yEd (from yWorks) for drawing models like the one above, mbt.tigris.org for test sequence generation from the model and any scripting tool at hand (preferably with functionality to call a web service, built-in or programmable) like Selenium (or the not so free HP QTP or IBM Functional Tester).

Now, the model created in yEd will be saved in .graphml format and mbt.tigris.org are able to interpreted that to generate test sequences (more on that later). To actually run the sequence we need to implement the functionality represented with the Edges and Vertexes in our scripting tool.

One beauty of this approach is that the code will be fairly small functions of "do:ing", that would be Edges, and mainly "verifying", Vertexes. I say mainly because one usually need to do some things to be able to verify, like open a customer dialog and search for latest transaction or create a database connection to check a certain post in a datatabel etc.

Prolore AB   Page 3 (6)   Web: www.prolore.se
Västmannagatan 43   E-mail: info@prolore.se
113 25 STOCKHOLM   Org Nr: 556683-3421

OK, I get it in general, how do we start with the model?

Well, the model is actually the most important part in MBT since it contains most of the thought-works and business intelligence. There are several questions that need to be answered (and probably a lot more than covered here) so let's walk though some of them:

- What should the model represent?
  I suggest that you start with the model representing the end usage/user functionality of the system. That will usually resemble your traditional functional test case approach. There are several other approaches for you to explore later in you new MBT career…

- To what level do we model?
  That is an interesting question! And a potential trap! Instinct will tell you not to model to "shallow", an extreme case would look like Edge_StartGoogle, Vertex_GoogleHome, Edge_NavToImagesAndVideosAndLoginGMailWithValidAndInvalidDataAndNavToHome and finally Vertex_ValidateAllWeJustDidt!

  Such a model will not benefit from anything. The data you want to verify is very far back and even more serious is the total lack of possible, "random", paths through the system

  The opposite is not too good either, by cutting all and everything into small pieces you will end up with a too big model representing fairly little functionality compared to its size and loosing the overview benefits of the model. OK, so what is the appropriate medium way? I'd say use your knowledge, experience and gut feeling and fine-tune from that. Remember that you haven't scripted anything yet so changes are fast and cheap!

- Who creates the model?
  That is another interesting question! I suggest that **you** do it and start with interviewing the SME:s (Subject Matter Experts) to gather the main business processes and the possible optional paths through the system.
- For first-timers: Please remember to start with a limited scope! The SME:s could be senior testers with fair experience of the system (if it exist) and business or power users from the LOB (Line-Of-Business).
- Secondly you could talk to the developers, showing your model and ask if they see any potential misses or unnecessary coverage based on their knowledge about what's under the hood. Eg. that you don't need to cover Y functionality in your model since the X functionality already verify the same thing (tricky question: do you really trust the developers? ☺).
- If the system is non-existing, you probably would consider user stories, product backlog items, use-cases or the equivalent requirementish information.
- During this process you will collect the first benefits of working with MBT! You will be one that discover the misalignment of understanding between developers, testers, SME:s, product owners and the alike, and you didn't even start to automate any tests yet! Now, that is proactive testing! (couldn't we get this benefit with "classic" automation? Sure, just try getting the parties mentioned to read your test cases….compared to browsing through a model on the

wall and informally discuss the functionality. I'll bet even your boss will glance at the model and nod in pseudo-understanding….)

So I have a decent model, can I automate now?

Sure, let's just do this a little step-vice.

First you need to get all the names of the edges and vertexes into your chosen scripting/test automation tool. Copy directly from the graph or let the mbt.tigris.org engine extract all unique names.

Secondly you need to code and/or record. Every edge and vertex will be a small function (with the name exactly as in the model) in your scripting tool covering the functionality you intended while drawing the model.

And now we run the Model?

Yes, finally! And now you have the opportunity to utilize the full potential of working with model-based testing because of how you run it, or rather how you configure the mbt.tigris.org engine to generate the test sequence. You could for instance have the condition to run the model until every edge and vertex is passed at least once or run it for 1000 steps or randomly in 3 hours or…whatever your testing goals are.

You will most probably execute the model in some of the random modes because that is what's make the test coverage dynamic. That will also require some change of thinking since most of us take some pride in knowing exactly how and what to test. But now we have the random thing in our testing and we do not beforehand know if we will start by verifying googleImages, googleVideos or the GMail. The randomization however will cover more ways of doing things in our SUT without re-coding or expanding the amount of scripts.

Remember that we're here to verify the requirements (outspoken or implied) and if the Model cover them they are tested and we report our progress accordingly. In the model above you see that the vertexes have REQTAG:s. These could easily be collected from the results (depending on your chosen scripting tool) and visualized in a graph, spreadsheet or intranet. That would make it easy for the stakeholders to see what's been tested and the status of our SUT.

For more technical information about how to run a model please visit mbt.tigris.org!

Prolore AB   Page 5 (6)   Web: www.prolore.se
Västmannagatan 43   E-mail: info@prolore.se
113 25 STOCKHOLM   Org Nr: 556683-3421

How does all this apply to Agile development?

If one of the key ingredients in Agile development is to allow more or less constant change we need to find a way to test automate that embraces that way of thinking. The model enables us to make swift changes along with minimized maintenance of the scripts. Real life experience from my part and a couple of colleagues imply a 50-50 split for a change that needs only re-modeling to re-modeling and some re-coding. Another factor that makes re-coding easy, if at all needed, is the functional structure of the automated test scripts. It is easy to pinpoint exactly where to change and since the small functions is just that, fairly small, they are easy to grasp and update!

Even Model-based Testing is an enabler for test automation in the agile world the benefits should not be disregarded in more traditional ways of working. After all who doesn't want the benefits of this approach, agile or not?

So this thing is like Jesus, right?

Hmm, sure there are some issues that need to be addressed, the main one is to let go of the traditional thinking. You will most probably fight old expressions like "Lets run some scripts" to "Lets run some models" and try turning management questions like "How many test cases have we automated" to "How much of our requirements are covered in the models".

Other from that MBT of course struggle with the same questions automated test always have to face; choice of tools, what to automate and what not to automate, who to do the maintenance etc. But I will bet that the agility and clarity of the MBT will lessen these issues and make way for a much higher success rate and a more wide spread use of test automation!

Prolore AB   Page 6 (6)   Web: www.prolore.se
Västmannagatan 43   E-mail: info@prolore.se
113 25 STOCKHOLM   Org Nr: 556683-3421