Matti Vuori, 18.3.2014

RATA project report

About the applicability and benefits of robot assisted testing

Contents

1. Introduction

Robot assisted testing is a quite new thing – as is any test automation to many people. Robot assisted testing is not only new in the sense that is not used much in the industry, it is also a quite new concept and there are not many experts even to ask about it. This is in contrast to traditional software test automation, where there may be a couple of experts in many companies.

Because of that, we have written this short document. It will briefly explain about general goals and challenges of test automation and outline the potential benefits and good applications for robot assisted testing.

The context here is the testing of applications in devices – all the verification and validation of software systems during their development and readying them for releases. We will not be discussing testing of for example physical characteristics of mobile devices here.

2. Why do we do test automation?

There are various reasons why we do test automation. Some of them may be obvious, but let's look into them.

The first thing about automation is always letting the machine do most things that a human used to do. It is understood that machines in general can be cost-effective in doing many things that can be "programmed" and where the great abilities of humans are not required, for example using our knowledge or the world in making decisions and doing ad-hoc planning of how to proceed. A machine can only do what it has been told to and it can only react on ways that it has been instructed to. When it meets surprises, it gets stuck, but a human can analyze the situation and figure out how to proceed – perhaps after discussing with other humans.

So the traditional use of test automation has been the testing with simple scripts that produce some results that the machine can look and decide whether the test passed or not.

The real issue here is repetition. It takes time to automate things and automation requires plenty of effort. With the effort required to create test automation for a simple data entry task a tester could herself test that thing several times! So if testing something is only needed a couple of times, automation makes no sense. There are, luckily, things that need to be tested several times. Regression testing of applications is one such thing. After some part of an application has been changed, other parts of it are tested for regression – checking that nothing has been broken in the process of making improvements. Such tests can be run quite many times – sometimes every few hours, for several years… Of course, the problem here is that the things that are regression tested can change and that can lead to a nightmare of updating the automated tests.

At the other end of the repeatability scale is the rapid testing of new features in agile software development. After new features are implemented they need to be tested fast, to get quick feedback. But then the features may be changed, developed further – or dropped altogether. Thus, the tests are never repeated. That is the benefit of humans: we can start testing very fast, with no overhear, and we don't leave behind us baggage that should be reused. We "just do it". Of course, some of the tests that we do in rapid manner will continue their life as regression tests, but not nearly all. Because we have already carried out some serious manual testing, automating the good parts of it is easy.

Sometimes the tests that are automated are not simple either. Simplicity of the tests is usual at the user interface tests, but “below the hood” there may be tests that are automated exactly for their complexity that humans cannot handle. Consider the testing of some data exchange protocols. There may be so many variations in their use of data fields that the only way to test them is to use an algorithm to generate and execute a large number of tests. That way we can gain great coverage – if there is enough time, a good portion of the variations can be tested. Humans could never do anything similar. We can also add randomness to the test creation and execution because that will often aid in bringing up problems,

But even in those cases the decision to automate is not so clear-cut. That is because humans can see potential problems and focus on critical issues, which reduces the number of tests needed. That may be done with careful analysis and test design or by exploratory testing that relies on the test observation to derive new tests. That is why some companies have turned into exploratory testing even in regression testing of larger information systems – after running into the maintenance problems of automated testing. Be careful to note that all this is very different than using testers to execute in robot-like manner the same manual tests again and again… That is rarely done nowadays – or at least should not be.

3. What does it require to automate tests

Sometimes it is easy to create automation tests. When testing is done at source code level, the developers can write test code that executes the product’s code just like it would be executed by some component in the product. That is done especially in “unit testing”, where developers test the source code that they have written. Similarly, if the application has been written with good testability in mind, we can in simple test code instruct it to push buttons, insert inputs in form fields and check what happens as a result of that.

However, that is not always so easy. The first problem is that this relies on mechanisms that happen “under the surface” – what the tests think is happening might not be quite the same as what really happens in the user interface, when a user does similar things. For that, we might employ a system where an external application at higher level more accurately simulates a human user. Here we again run into practical problems. There still needs to be instruction for how to identify the buttons to “push” and where to type the inputs. But things change and user interfaces are often changed rapidly during product development. The tests can soon turn out to be not working anymore and need constant maintenance – effort, time and money.

Often the tests need to be run from another computer. For example, mobile application usually need to be executed and the execution monitored from a PC and getting access from the PC to the device and to be able to control the applications can be difficult. It should not be so, but in practice it is. Sometimes it takes a great effort to build this “adaptation” and connectivity the testing requires.

4. The problems with additional components

Each time we add something additional to the system under test, it is not the same system anymore. If there is a component that controls the system or monitors it behavior, the behavior will change at least just a little can sometimes change a lot. The testing systems make the system just a bit slower, they change the memory configuration and cause new paths of execution to happen. We do not execute the system anymore like it would be executed in real use. The term for that is “intrusive testing” – the things needed for testing really intrude the system under test,

Just as every application always has some defects – this is the basic premise of testing – the testing components have defects too, causing plenty of extra work in analyzing the observations and finding the real defects. The main hazard is that the test components may block some read defects and may not even test the locations in the system under test where they reside.

So what does the testing tell us? It still tells us plenty, but our trust to the findings becomes lower. We need to complement this "instrumented" testing with other means of testing, such as manual testing.

So, it would be very nice it we could test a system exactly as it is "from the package" with no external components, with no debugging tools, with no interfaces to a test control system.

That is exactly what robot assisted testing promises us.

5. …and why robot assisted testing is different

With a robot, we can test a system as purely as possible. We do not need to add anything to it. The robot is an external observer like a human and just uses its vision to see what state the system is, then based on it and its test scripts or models, decides what to do next. Checking of the results of each test is done again by looking at the device to see what has happened.

This is in principle fantastic! We can use the system just as a user would. What we see happening is the very same thing that would happen if a human would use the system. Everything happens just as fast or slow as in real life. Every little hiccup results from the system under test and not from some badly behaving test component.

The more critical the system under test is the more valuable this is. For example in testing safety critical systems this can be extremely beneficial. When we later in the development cycle do maturity testing to see how often we run into problems, it is great to have a test arrangement that as closely as possible mimics the reality. When we run a complex test with a robot for hours, it is much like a human would use it for ours just looking how long it takes before something goes wrong. However, human should not be used for that kind of testing.

During-development functional testing, things may not be so simple. Verification of what has really happened may not be easy from just a user interface – but that is the same situation as in manual testing. A human tester can more easily sidestep to see for example in the file system what has happened. The robot can do the same, but it complicates tests more.

Wit robots we have choices! We can have some instrumentation, when needed, but we need to be careful with that so that the "baby is not thrown away with the bath water". It is better to leave the instrumentation to debugging: when we find some anomaly which we do not understand or cannot trace it to a defect in software, we can for example execute software in a debugger during the robot run or turn on more logging and other monitoring during a debugging test.

So we still have a whole range of alternatives – and having alternatives and a full "toolbox" to use is a great thing for good testing.

Because of these characteristics it is easy to think of robot assisted testing as being close to manual testing, but still there are big differences:

- It can only react to things that have been programmed into it, whereas a human can detect all kinds of funny things – something wrong in some unrelated area of a display – and immediately research them further with some ad-hoc exploration.

- A robot cannot thus do any exploratory testing which relies almost purely on observations and little on detailed pre-planning. It can only do things based on pre-written scripts.
- If we do not know in detail what should happen when some action is executed, a robot is at loss, but a human could make deductions from what she sees.
- A robot can also more accurately for example measure timings – a task that is impossible to humans, who adapt to any slowness rapidly.
- Usability testing is obviously impossible with a robot, because it tests how things work from the viewpoint of a human.

The scope of this paper is in software testing, but we must not neglect some issues about those types of hardware testing that are very much related to the applications used.

Consider testing of how long the battery lasts in a mobile device. Clearly, we would like to have a controlled test arrangement for finding out that. We would also like that there are only such components in the device that cause it to use power. That means that we would prefer having as little instrumentation in the device as possible. A robot is good at this. We can let it carry out a long test sequence – hours or days that simulates user actions. After the battery runs out we are left with good information about the battery's performance. If we use a little system monitoring, we will nicely find out which applications and system software components use most of the battery power and thus can continue to optimize those if needed.

6. Robots can work around testability problems

Testability is the ability of a system to be tested. For years it has been understood that each new platform should be developed so that it is easy to control its applications with a test automation system, meaning that we can for example execute all UI controls of an application from another application. In reality this is often forgotten when systems are bought to market.

In some cases, the testability readiness might exist, but the manufacturer refuses to make the necessary components and tools – or just knowledge -- available that it requires.

Even when these issues are considered carefully and openly, there are problems due to the complexity of the systems. There may be several layers in the user interface, each requiring different interface to access:

- The platform UI technology.
- Special technologies used for some applications or components.
- UI implemented dynamically by scripting in the application, for example by JavaScript.
- Embedded components that sometimes simply cannot be accessed programmatically.

Luckily, a robot requires none of that. It can just open a device and start tapping away, independently of the implementation technologies!

An additional problem with any testing or debugging subsystems is that of security. Testing interfaces introduce new ways for crackers to enter the system and control it – perhaps through a network connection. That is why it makes sense to keep the systems as closed and controlled as possible.

7. Speed of starting to test

In today's hectic business, the speed and ease of getting testing started are essential. In the traditional world of test automation, a company might need to have a very capable developer spend some time – that depends on the practical problems that are sure to arise at some point – developing the adaptation between the test execution system and the system under test. Then come the problems of accessing all user interface elements from the test scripts.

That is all time that could be spent for doing something more valuable. Of course, all that needs to be done from scratch only once for a platform (if it cannot be bought from somewhere), and revised for all platform changes.

With robot testing, the ramp-up can be faster. We have the ability to start reading the screen and tapping things "straight out of the box". Still there may be some platform-based preparations to do, such as teaching the optical character recognition system the platform fonts so that it can detect texts in the screens. It is much faster and less prone to nasty surprises than instrumented test automation.

8. What testing robots can and cannot do in software testing

In Table 1 we list some test types and tasks and their applicability for testing with a robot. The table is arranged so that the more applicable test types are at the top.

Table 1. Applicability of robot testing to various test types and tasks.

Test type / task	Applicability	Benefits or problems	Notes
In general		*All the benefits and pitfalls of generic test automation* *A robot test system is needed…* *Visible UI testing has maintenance issues, but also better validity*	
Device validation testing (like response to gestures)	Very good Impossible by other means	Accurate, always the same	For example how a tablet responses to basic gestures – impossible for a human to test
UI response measurements	Very good Impossible by other means	Accurate, always the same	Use in basic validation, but also to measure that response times do not get slower with development (regression) or in long term testing
UI-level functional testing	Very good	Does not require instrumentation / internal adaptation – better validity Tests what the human user sees and can do Testing of multiple UI languages and UI themes will cause more work Device orientation changes and such require more advanced (expensive) robots	May use scripts or model-based testing; even monkey testing, but exploratory testing is obviously impossible

Test type / task	Applicability	Benefits or problems	Notes
UI-level testing of safety critical systems	Very good	-- " – Does not replace testing by humans, but may give a good "technical baseline" with good functional coverage	One must not get fooled with the coverage numbers; they are reached with quite mechanical actions
Long term UI testing / maturity testing	Very good	Similar to instrumented tests, but more accurate measurements possible	
Device performance testing (UI level)	Yes	Can be much faster than human in work with an UI and doesn't get tired	
Compatibility testing	Yes	Obviously a robot could test for example with several devices, but currently all the setup makes it impractical and software-based testing in a device farm is more practical	
Mechanical assistant for instrumented tests	Yes	Of course, a robot could just be an "assistant" in tests that are done fully in software. It could turn the devices, initiate their sensors, push buttons, remove memory card or charger plug etc.	These are all things that a human should not be doing
Usability testing	No / low	A robot does not have a human mind Can detect low-level problems with controls though (hitting buttons, icon differentiation)	Mostly should have a functional testing / control validation goal and not confuse with usability testing
User experience testing	No	A robot does not have a human mind or culture	
Business process testing	See UI-level functional testing		
Security testing	No / Weak	This needs a human expert tester	
Information system performance / stress testing	No		Some manual testing should be done while the scripts are running, but humans should do that, as it is needed for even weak signals and holistic observation
Unit testing & low level integration testing	No		These are done by developers with "under the hood" testing tools

9. Conclusions

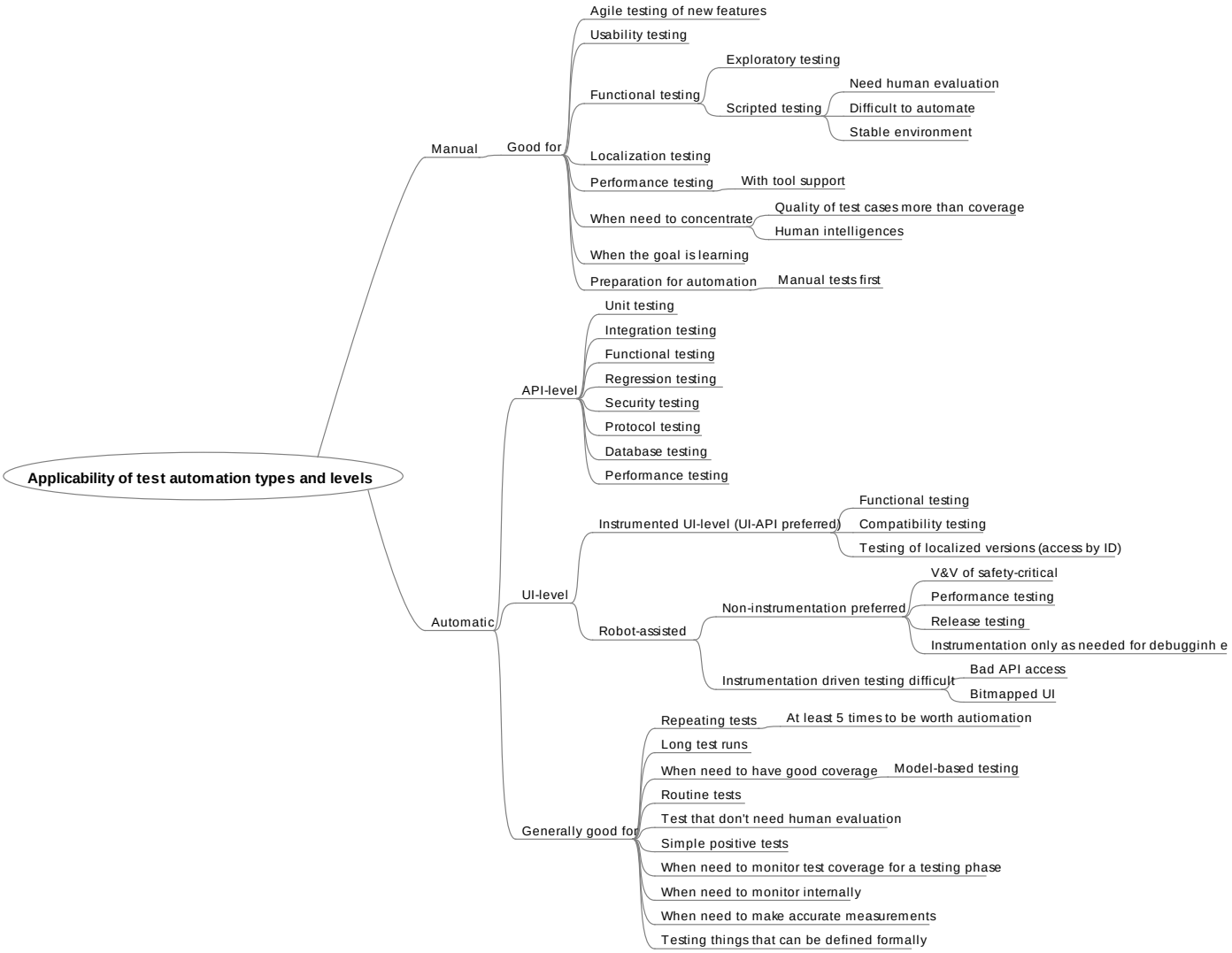
The robot assisted test automation clearly provides many benefits, such as having no need for instrumentation and the ability to test a system exactly as it is configured for production use. This provides remarkable benefits for many kinds of testing. Compared to instrumented test automation, these benefits are essential:

- More realism to testing and therefore more trustworthy results.
- Less work for testing arrangements, creation and maintaining of testing tools and scripts. This saves money and work.
- A better simulation of the user, thus better testing of user interfaces.
- More real-life like platform for device measurements, resulting in more accurate measurements.
- No need to compromise security because of testing interfaces. This is very valuable for managing security risks with mobile devices,

Yet, robot assisted testing is only done at the user interface level and obviously cannot replace test automation at lower abstraction levels – “under the hood” of the system under test. And it cannot replace many types of human testing.

There never are any silver bullets in testing, all testing technologies just add to the available toolset that can be used in testing, giving us the means to do richer testing and select the most suitable testing techniques for any testing task.

APPENDIX 1: Applicability of various types of test automation



APPENDIX 2: Issues around robot assisted UI testing