Matti Vuori, 30.8.2013

RATA project report

Locating the defects from robot test run observations – or where is the problem?

Contents

1. Introduction....2
2. Potential sources of errors....2
3. But if there are no anomalies… 3
4. Strategies for dealing with difficulties in testing....4

1. Introduction

When a robot assisted test systems makes a test run and there is an anomaly, great care must be used before reporting a defect, as the anomaly may be caused by many things in the test system or the device. Understanding the possible problems in test execution is essential because robot assisted testing has its own unique pitfalls, but also shares many issues with traditional test automation and model-based testing.

Here we will look into the potential sources of the anomaly and give guidance for locating the real problem – and how to work around the problems in testing.

2. Potential sources of errors

Here is a list of potential sources of anomalies or errors perceived in a test run:

- **Defect in the SUT or the application under test**. The best case is that there really is a bug and it is located in the last operation done before the test system noticed that something was wrong. This is easy to verify. Because a test robot does things in similar way to a human tester, just repeat the actions manually and see what happens. If an error is detected, it should be a straightforward defect. Further debugging can continue by checking the application log for details, instrumenting the system for a second test or by turning into traditional testing (including any state-of-the-art techniques such as fuzzing). But if the error can't be repeated, one should first check that the test was actually executed correctly and then try to detect from a test trace or logs when and where something started to go wrong, resulting at some point the detection of an anomaly. That can sometimes be difficult.
- **Error in the test model or test script**. An error in the test script or test model is very common. After all, they are program-like artifacts and will contain errors especially early in their lifespan.
- **Errors in test generation**. A somewhat similar issue is errors in the tests dynamically generated from the model. They are in nature a bit different though: errors in models and hand-crafted scripts are caused by the modeler or test designer, but errors in the test generation are caused by defects in the test generator software and need to be worked around or the generator to be fixed, which obviously may take some time.
- **Outdated test script or model**. A special case it that when the application is developed, the test models and scripts will become outdated and need to be updated. That can be a big problem in especially agile development. This is actually the biggest obstacle to any automated testing. One needs to be prepared to that and to have resources for doing the updates. Even the smallest things matter here – a change in a button text is enough to break the testing. This is not usually a problem in traditional test automation, where the tests can refer to some internal ID of all UI objects.
- **Defects in adaptation**. Of course, the adaptation layer is very prone to have defects.
- **OCR and icon recognition problems**. Detecting items on the device is difficult and unreliable. A good test system will have facilities for finding out what texts and icons it thinks there are in the SUT and from that one can detect what should have been recognized. But this can be a difficult thing to solve and can require skipping some UI actions or making quick-and-dirty fixes somewhere. Configuring the OCR and icon recognition for testing an application family is very important, as is teaching the OCR system all fonts that are used. Of course, this is also influenced by the imaging technology used (quality of the camera and image preparation software).

- **Accuracy of robot actuation**. If something was tapped but nothing happened was it really tapped? The area that receives taps can be smaller than it looks like. It may be, that mechanical tolerances or other reasons cause the tap to just so slightly off that it is not registered. Of course, this could raise a usability bug: is too much accuracy required from the used? On related note, if the robot aims to mimic human behavior, it should have variation in all its actions. Furthermore, just as with humans, a poorly designed robot finger can press two buttons or a wrong button. Calibration of the whole system is obviously essential. If all actions are off to some direction by a couple of millimeters, many things will fail.
- **Timing**. If the robot works too fast (or sometimes too slow), the SUT may not be able to accept the inputs. Of course, a good test system should have means for visual checking whether the SUT is ready for some action, but sometimes simple delays are needed. But if the robot works slowly things may change in the SUT – a dialog closes automatically, for example, if it is not tapped during a couple of seconds.
- **Test environment problems**. Screenshots and accuracy of ORC can be hindered by too dim or uneven lighting. If the camera sees a reflection of a light bulb, OCR has a hard time recognizing the text in that area. Similarly instability of the table the robot is placed on can cause vibrations that in turn cause lower accuracy of actuation.
- **Disturbances during test run**. Of course, during a test run, many things can happen that we either have not foreseen or perhaps cannot handle by the test system or decide to not handle, to keep thing simple and to be able to concentrate on more pressing matters. Those may include alarms, timed applications popping up, running out of battery power, getting a phone call to the device and so on. We just need to assess what might happen, try to minimize the effect of those that we can, and live with the rest.

3. But if there are no anomalies…

It may sound surprising, but a common problem with test automation is that the test scripts seem to get executed, but no problems are actually reported, just passed tests. It may be that the test system cannot connect to the SUT but reports successes for the test cases. With robot assisted testing we can at least hear that the robot is doing something… and as with any mechanical automation, possibilities for visual monitoring are needed – a web cam to see what is happening.

Sometimes the cause for the false passes is disabling the internals of test scripts because nobody has time to update them. When the scripts start rotting, soon the whole testing phase is worthless. The tests need to be in good shape all the time.

A related problem relates to test design: if the tests do not really test much, but just go through motions, they will pass but do not test anything relevant. So make sure that the tests really test the system. That's the whole point…

4. Strategies for dealing with difficulties in testing

Of course, fixing test system problems in a firefighting mode does not make much sense in the long run. To avoid that, we need to have some general strategies to dealing with robot assisted testing. Those include:

- **Ensuring testability**. As with any type of testing, good testability is essential. The developers should consider the testability. Special testability reviews should be considered in many cases.
- **Test-friendly SUT configuration**. Perhaps the SUT can be configured so that robot testing will be more easy and reliable. For example, some animations could be turned off for testing and difficult background images could be replaced by simple one-colored backgrounds. Of course, one needs to consider all effects of such configuration changes.
- **Inherent difficulty in testing a test condition**. Perhaps the test conditions are just inherently too difficult to test with a robot. One should remember that robot assisted testing is never the only testing done, and one should only use that where it is practical. Other things can be tested by other means: manual testing or by other types of test automation. Common difficult issues include a very dynamic user interface, complex gestures, non-determinism, artistic user interface design.