Five Symptoms Your Test Automation Is Dying

An Doan

an.doan@outlook.com

Abstract

Are you thinking about starting an automation project? Are you chest deep in test automation? You might not know it yet, but your automation project may be on a collision course with an undodgeable Hadouken Fireball. Certain telltale symptoms can help identify problems early. This paper discusses the key symptoms, what causes them, why they are terminal, and what you can do to diagnose and treat them. These anti-patterns, if left unattended, will degrade the quality and efficiency of your test automation, and may cause hardship and eventual failure of the automation project. As with all illnesses, having a symptom or two does not mean imminent failure. Awareness of the symptoms, however, including how to identify and treat them, can help avoid automation pitfalls.

In the context of a software test automation project, this paper looks at five key symptoms that may be evident, and could cause automation projects to fail.

Biography

Developing test automation by day, speaking and organizing automation meet-ups by night, An Doan is an 8th year "Dark Developer" specializing in writing dark code, code that breaks other code. He develops test automation frameworks and quality processes. As co-founder, and idea incubator of the Portland Selenium and Test Automation Users Group he helps the community share and develop creative ideas in the realm of testing and test automation.

1 Introduction

Testing early and testing often is the key to delivering a high quality piece of software. That makes automation the most important testing technique in allowing QA to test early and often. Unfortunately, it is also the most difficult get right, and often fails to be of any true value. Think about the following questions:

- Does the team support test automation? The company?
- Do you have resources at your disposal to procure automation software, hardware, and people?
- Are you trying to train manual testers?
- Are you trying to convert your manual test cases into automated test cases?
- Does your primary concern revolve around making automation business readable?
- Are you using automation to fulfill a business need or solve a business problem?
- Are you dependent on record and playback tools?
- Does your automation framework seem clunky?
- No automation guru on your team?

If you answered 'yes' to more than a few of these questions, your test automation may be dying. In this paper, you will learn about the five symptoms that result in test automation failure, how to identify these automation anti-patterns, and how to treat them.

1.1 Test Automation

Test Automation is the art of using software to test software. Anywhere where there is any kind of test against a product, it has the potential to be automated. A great example of upper limits of automation is the Tapster Mobile Automation Robot made by Selenium contributor and Sauce Labs co-founder Jason Huggins. It automates the interaction of a human finger with a mobile touchscreen. The task is daunting, but the rewards are well worth it.

2 The Symptoms

2.1 The Missing Project

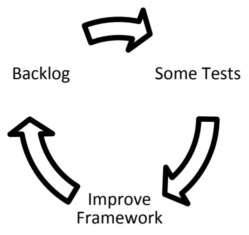
You have been brought into an Agile team working on a new mobile application. The manager wants to do automated testing, but it is not captured anywhere in the Agile backlog and workflow. No one is bothered by how it happens or what the results look like, they just want test automation. There are other teams within the company that might be able to help but they all seem to be doing the similar things, solving similar automation problems, and reinventing the same wheel repeatedly.

2.1.1 The Symptom

Creating great software requires planning and organization. Likewise, test automation is software development, and as such requires the same level of planning and organization. Test automation is complex, sometimes difficult and requires careful planning and precise execution.

You have this symptom if your automation framework is not designed, developed, and maintained as a project.

For small teams test automation should be a part of the main development project. For an environment with multiple development teams test automation should be its own project with its own team. Its mission would be to create automation and automation tools to support the other teams.



2.1.2 The Cure

Treat this symptom by finding a leader, treating automation as an Agile project, taking small steps, and making iterative improvements.

2.1.2.1 Leader

First, you need to find a leader. This person need not be an automation guru. However, this person needs to be able to solve the problems of today, while anticipating the solution to tomorrow's problems. Solving the problems will involve learning new tools, seeking out new resources, as well as new technologies. In order to track and solve problems effectively, we will borrow some proven practices from the Agile world.

2.1.2.2 Project backlog

The second step in getting your automation organized is to create an automation backlog. You should use your development team's backlog if you can, or use a separate backlog. Getting your automation visibility within the team is important but do not let it get in the way at this point. What is important right now is to get tasks down on paper.

In Agile, an Epic "captures a large body of work. It is essentially a large user story that can be broken down into a number of smaller stories." (Atlassian), and a User Story "captures what a user does or needs to do as part of his or her job function" (Wikipedia). In this situation the user is the team's automation engineer, the person or persons responsible for writing automated tests.

The following Agile Epics should be written, prioritized, broken down into smaller Stories, and solved one by one. As these Epics are interrelated by the automation project as a whole, remember to think about each Epic and how they would be solved in relation to each other. For example: sharing test results is related to seeing test results, the audience to which the reports are shared influences the format in which they are presented.

- "I want to write an automated script"

  This Epic is about choosing a test tool. Who will write automation? What tool or programming language will be used? How will the automation be written; i.e. what framework and design patterns will be used? When will the tests be written? Before, during, or after the application's development?

- "I want to see the test result"

  This Epic is about test reporting. "If a tree falls in the forest, and no one hears it, did it fall?" How will the team report automation progress and success? How does the automation report failure? When it fails, it should be easy to determine the failure. Who will read the reports?

- "I want the script to run on a remote machine"

  This Epic is about how the tests run. The automation does not give much value if it takes over the user's computer and the person has to sit there and watch it run. If that's the case you might as will test manually. The automation should run on another machine, so that the user's time is freed up for more development, complex testing, or exploratory testing. The chosen test toolset should have the ability to run on a remote machine. It may come with this feature out of the box, or be added through customization.

- "I want to run the automated script automatically"

  This Epic is about the mystical unicorn that is Continuous Integration (C.I.) and Continuous Deployment (C.D.). A few have it, we all want it, but we are not too sure if it exists or how to get there. Having automation, and having the ability to run it automatically and repeatedly is a key step in achieving C.I. and C.D.

- "I want to share the result"

  This Epic is about publicizing the test reports. What good is our success if we do not celebrate it loudly? Are the test results sent out in email? Are failures automatically entered as bugs? Is there a big dashboard on the wall that turns from green to red? If a developer causes a test to fail, which flavor of doughnuts do they need to buy?

2.1.2.3 Take Small Steps

Now that the automation road map is laid out, plan to accomplish these stories in small steps. Start with the 5 or 10 most important smoke tests. Automate those, get a simple report out of it, get it running automatically, and share your success. Think simple, but expect continual improvements. The tools used in the beginning might not be the correct tools for the job, so be open to change and improvement.

2.1.2.4 Iterative Improvements

The last step is unfortunately not the last step but an "advance to go, collect $200." An Automation Project is continual. Just like the way your application is continually improved, and your team continually grows, your automation must improve and grow. When new features are added to the application it may be necessary to add or change the tool or framework used for automation. Upgrade your toolset continually, leave your good tests in the old framework or tool, and as you add automation for the new features implement them using the new framework or tool. Eventually as your application is improved, so will your automation. More tests may mean more test hardware. Longer reports may mean reporting improvements. The automation should have the ability to improve as needed.

2.2 Training Manual Testers

The company has just started a new training program aimed at improving testing and thus improving the quality of the products delivered. A new automation title is created and any manual tester who wants this new title needs to attend programming and automation training. The training program was well received, and initial impressions were positive. Tests were automated left and right and all seemed well. Almost every team had at least one automated test. As the easy tests were completed the newly trained automated testers struggled to automate the more complex test cases. The automation had grown to the point where it was beyond of the scope of the original training. Only a few truly knew what they were doing, and the rest followed like lost sheep.

2.2.1 The Symptom

Another symptom of a dying automation project is the prospect that a manual test team can be turned into an automation team through training. Borrowing a line from Liam Neeson's character in the film "Taken"; automation engineers have a "particular set of skills; skills … acquired over a very long career." This does not mean that a manual tester will not ever become an automation engineer. It means that not everyone will succeed. On top of that, it will take experience to be successful. There will be a low success rate when training manual testers.

2.2.2 The Cure

A manual tester trained in a dozen automation examples will most likely only be able to write tests similar to those dozen examples. A select few will have the particular skills, to become an automation engineer.

- Programming background – Automation engineers should have a background in various forms of programming and programming languages. Learn enough recipes until you can create your own.
- Creative Nature – Automation engineers need to be creative. It requires a bit of problem solving and a bit of out of the box thinking. This comes with experience.
- Inquisitive – Automation engineers are inquisitive by nature. They are always learning, always seeking out new technologies, attending conferences and meetups; and learning from others.

There are two types of training, externally motivated and internally motivated. The first one creates manual testers that try to automate, and the later creates automation engineers.

2.2.2.1 Externally Motivated

Externally motivated training, is training that comes from an ulterior agenda. It is something the company wants, not necessarily what the tester wants. It is the company realizing too late that exclusive manual testing is costing them too much. It is that one requirement that will get them the next career bump.

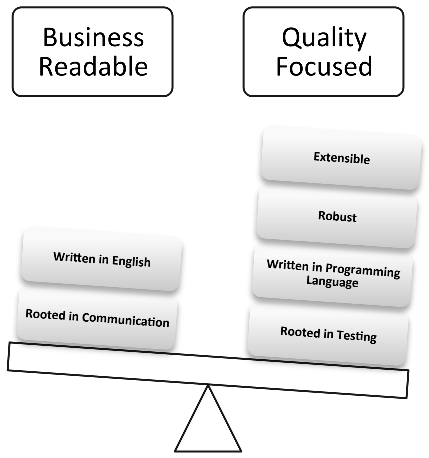
2.2.2.2 Internally Motivated

Internally motivated training comes from the tester. It is born from the desire to test, the desire to test better and faster. This desire will eventually lead the tester to automation as a superior supplement to manual testing. The tester will seek out books, classes, meetups, and conferences.

2.3 Business Readable Trap

The QA director has just recommended the use of BDD and Cucumber. All development teams are expected to write their acceptance criteria in Gherkin, and all test automated is expected to be written in Cucumber. The stakeholders however had little time on their hands, so the testers were left to write the Gherkin on their own. The developers just wanted to write code. They did not think testing was their responsibility so they gave no regard to the Gherkin. At this point, the automation engineer not only had to write the test automation, they had to write the acceptance criteria in Gherkin. It only mattered to the team that the build was red or green, and the Gherkin was just some text that was recorded in a log. Cucumber was supposed to make automation easier and faster, but it just felt like just extra work.

2.3.1 The Symptom



A business readable domain specific language will not make automation easier or better. Are you automating for the primary purpose of quality or are you automating for the primary purpose of fulfilling business requirements? If it is the latter then the automation project has fallen into the business readable trap. Automated testing is a solution to a testing problem to help prove that the application works well and correctly. The primary purpose of automation is not to solve business problems but to prove that business requirements have been fulfilled. Tread lightly when trying to solve business problems with automation, it is a symptom that can lead to automation death. Testing is about quality and automated testing should remain deeply rooted in quality.

There are tools out there, like Cucumber or FitNesse, that allow English, in the form of business requirements, to be used as a programming language for writing automation. In the ideal situation, the business owner or the product owner would write these. However, that is usually never the case and the QA team member has to write the business language in addition to coding the automated test. This is where automation starts to fall apart. Over reliance or over requirement of business language can lead the team to the point where automation is about making the requirement green and no longer about testing the application.

These types of test tools tend to become brittle as test cases become more complex, and will fail to scale when there are many tests. Their primary focus is on the presentation of the business requirements and making them green. As a result, they fall short when it comes to the actual automation and testing. Being written in plain English the tools often lack fully featured code development environments that help with writing and debugging the test. These tools also have poor support for parallel test execution, the ability to run multiple tests at a time to speed up overall test execution times. This makes it difficult to incorporate the tool into a Continuous Integration workflow. At this point, more resources are being spent trying to cope with the tool’s shortcomings rather than creating actual automated tests. This is a sign that it might be time to upgrade to a better toolset.

2.3.2 The Cure

The only way out of the Business Readable Trap is to upgrade the toolset to one that is rooted in quality and is about testing first. Do not just use a business readable automation tool for the sake of using the tool. Think about testing first and foremost. Identify a testing problem, then select a tool to solve that problem. Business readable tools are often solutions looking for problems.

2.4 Record and Playback

The team knows the value of automated testing, but there is a lack of expertise on the team. Testers get their feet wet by trying out some record and playback tools. They seem simple enough: start the tool, perform the manual test, it is recorded, and the test can be repeated by playing it back like an old-school cassette tape. It was an immediate time saver. As the number of tests grew it took longer and longer to complete the test, and even longer to fix the test when enhancements were made to the application. The solution was obvious, run tests in parallel to test faster and create reusable pieces of automation to reduce repetition and maintenance, but it does not seem possible with the record and playback tool.

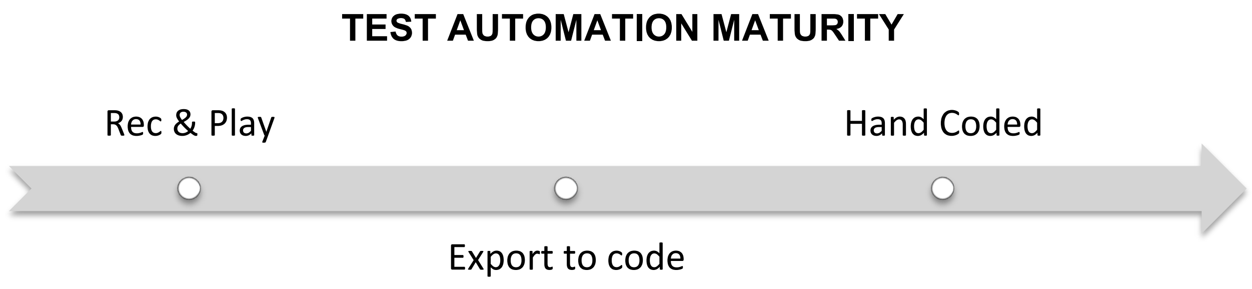
2.4.1 The Symptom

Strong reliance on record and playback tools will limit what you can automate. You have this symptom if you depend on record and playback tools to automate.

While record and playback automation tools are great for beginners, their utility quickly diminishes in more advanced situations. As a beginner's tool, it works well for a small number of simple to mediocre test scripts. Once you advance to dynamic websites, advanced interactions, or parallel test execution even MacGyver will not have enough duct tape around to hold the tool together.

2.4.2 The Cure

There are three levels of maturity that an automation framework should have the ability to advance through, and should advance through. Before creating too many tests at any level, forethought should be given in how the framework can be advanced to the next level.



2.4.2.1 Rec & Play

The most basic form of test automation is record and playback. These tools record the user's interaction with the application. The recording can be played back to repeat the interaction. These are simple to use and creating a test is as simple as performing the manual test.

However, these tools can only record basic tests. Record and playback tools will most likely not support applications with dynamic elements, or advanced user interactions. When it is time to execute the tests, most will not support Continuous Integration tools like Jenkins or TeamCity. As your test suite matures and the number of automated test grows, you will want to run your automated tests concurrently in parallel. Again, most recorded and playback tools will fall short of your needs.

2.4.2.2 Export to Code

As an added feature, some record and playback tools will allow you to export the recorded test out to code. This allows you to modify and used the recorded test in a native programming language rather than the proprietary language used by the tool. This opens up the support for Continuous Integration tools, and concurrent testing, but you are still limited by the type of interactions that the tool supports. If your application is dynamic or complex, corrections or heavy modifications will need to be added to the exported test. Even then it might not work.

2.4.2.3 Hand Coded

The most advanced form of test automation is hand coding, where the bulk of the automation is coded by hand. Test automation is customized to the needs of the team and the characteristics of the application under test, in order to maximize utility and re-usability. This form of test automation is best, and can automate almost any type of application. It is also the most complex and needs to be treated like software development. All test automation should strive to achieve this level of maturity.

2.5 Tumors

As the automation guru, you have been asked to assist another team with automation. The previous automation engineer left behind mountains of documentation and a comprehensive suite of test cases. Investigation reveals that the automation takes hours to run, and when it finishes the reports are difficult to understand. Trying to incorporate parallel testing and improved reporting into the existing framework, turns out to be too difficult and time consuming. It is like trying to fit a square peg into a round hole. The automation toolset just did not support parallel testing, and the reporting improvements would require too much customization make it right. There are better tools out there, but what about the last two years worth of automated tests? Are they to be rewritten?

2.5.1 The Symptom

A poorly architected framework can grow out of control. This results from putting too many common functions into a single common library. It can also occur with keyword driven frameworks that have too many duplicate keywords or phrases. Test suites take too long to run, or we get reports that are just a wall of words. Technical debt can grow into tumors if the automation framework is not regularly evaluated, maintained, and refactored. This can result from poor initial design, lack of forethought, or due to team member turnover.

2.5.2 The Cure

Fight this symptom by using a Page Object design pattern, and a clean framework API.

2.5.2.1 Page Objects

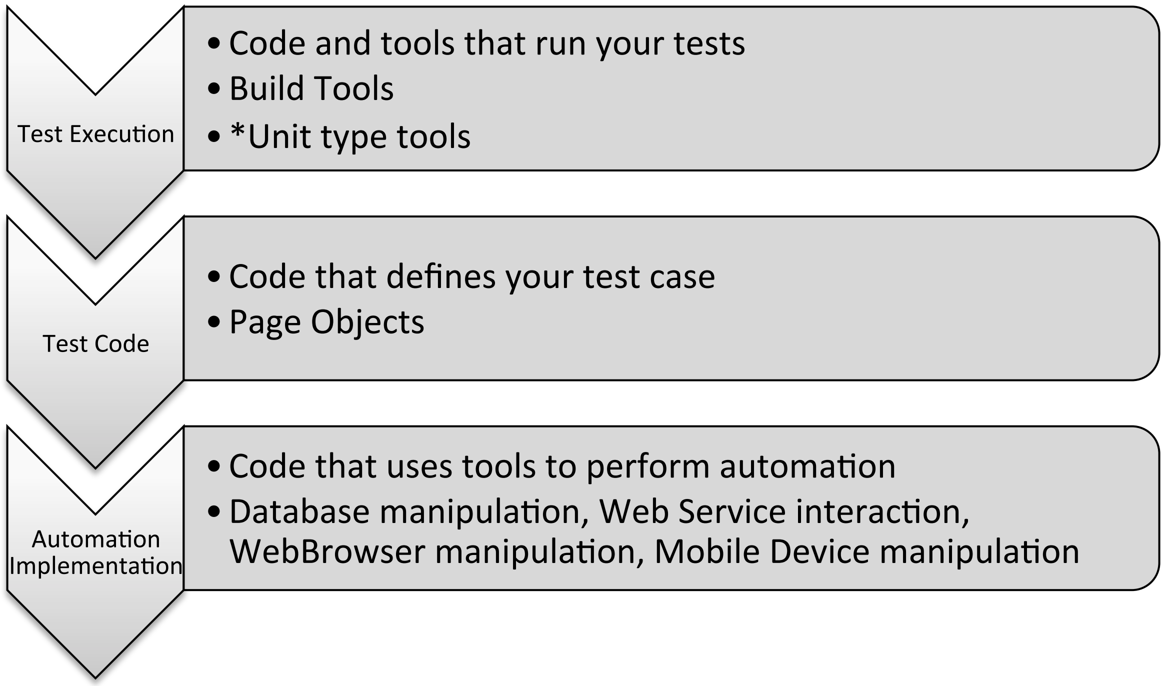
Your automation tests should follow the Page Object design pattern. The Page Object design pattern is an object oriented design pattern where each page or sub-section of a page in your application is represented by your test code as an object. The things on the page, known as the "nouns", are the pieces of code used to identify the things on each page. The actions you can perform on the page, known as the "verbs", are the things you can do on each page. That way the code is clearly organized and when enhancements or new features arrive in your application under test, updating the automation is as easy as going to the appropriate page object and updating the "nouns" and "verbs". This is a widely used automation technique, mostly used by the web automation community, but it can be applied to all types of automation.

2.5.2.2 Framework API

Whether you inherited it or created your own test automation framework, it needs to layered, modular, and the reports need to be actionable.

2.5.2.2.1 Layered

To maximize longevity and maintainability of your test automation separate your test case code from your implementation code. Create layers between your test code and your test automation tools. Depending on your chosen development language, the layers will be implemented as separate files, objects, or classes. This will make your code easy to maintain and allows you to enhance, upgrade or change your test tool without drastically affecting your test code and test execution. These layers are not hard and fast, and the lines can blend, but the more you can separate these layers, the better your framework will be and the longer it will last.


Test Execution
• Code and tools that run your tests
• Build Tools
• *Unit type tools
Test Code
• Code that defines your test case
• Page Objects
Automation Implementation
• Code that uses tools to perform automation
• Database manipulation, Web Service interaction, WebBrowser manipulation, Mobile Device manipulation

Test Execution: This layer encompasses the code and tools that are used to run your automated tests. They are tools like Gradle, Maven, JUnit, TestNG, NUnit, and Rspec. These tools usually sit on top of your test cases and are responsible for running your test cases. Keep this layer restricted to test parameterization and execution. Keep automation out of this layer as much as possible. For example if you need to set up a database prior to running a test, create the database setup in the Automation Implementation layer and make method calls to it from the Test Execution layer. That way if your application upgrades to a different type of database, only the Automation Implementation layer is affected.

Test Code: This layer should contain only your test code and it should use the Page Object model described above. The test case will make method calls to the Page Object, and the Page Object will make method calls to the Automation Implementation. The benefit of this is that you have a single entry point for multiple automation tools, and you can swap out automation tools without affecting the test case.

Automation Implementation: This layer performs all of the automation. It pulls in many tools like web, mobile, web services, and databases and provides a single common interface for the test code. For example, you can use Selenium for web browser automation, and Appium or Robotium for mobile automation, but from the perspective of your Test Case, it is abstracted away and they are the same. The Test Case does not need to worry about how the automation is implemented; a click is a click regardless of the platform.

2.5.2.2.2 Modular

The code that performs the automation should consist of many tools and should be modular. For example, a single API can represent both desktop web browser automation and mobile web browser automation; that single API can leverage different modules to perform the appropriate automation on the different web browser platforms. When it comes time to upgrade, the desktop module can be upgraded independently of the mobile module while maintaining a consistent entry point for your test case writers through the single API. The result is a lightly coupled automation framework that can change to meet the testing needs of the constantly changing application under test.

2.5.2.2.3 Actionable

When a test fails, the report that results should be readable and actionable. What good is a test if it fails and no one knows about it, or no one understands why it failed? It should be as specific as possible and attempt to identify the location of the error. Use English whenever possible and avoid stack traces. If testing a UI, include screenshots in the report. The report should also be saved somewhere to allow historical reference to make it easy to compare a failed report to a previous passing report. To increase its visibility the report should also be compatible with Continuous Integration and build tools.

3 Conclusion

As detailed in this paper, there are many ways for a test automation project to die. Each one is treatable. The automation should be treated as a project, a never-ending project that undergoes constant evolution like google.com. There should be an experienced lead that keeps the automation framework two steps above the needs of the team. Instead of trying to train manual testers, try to encourage them to attend conferences and meetups; it is the best way to learn automation if they truly want to learn. Understand that it is a long learning process, so it will not happen overnight. In the end hiring more specialized automation resources will still be required. Unless the business is on board and is asking for business readable automation, do not do it. It is just an extra layer of work that will slow down automated test development. Record and playback tools are good starter tools and a good learning resource, but do not stop there. Record and playback tools are not made for serious enterprise level automation. Keep an eye on technical debt caused by automation. It can poison the automation from the inside out if it is left for prolonged periods. Monitor and treat these symptoms and the automated tests will be more successful and easier to work with.

References

Atlassian. “Working with Epics.” https://confluence.atlassian.com/display/AGILE/Working+with+Epics (accessed August 22, 2014).

Huggins, Jason. “Tapster from hugs.” Tindie https://www.tindie.com/products/hugs/tapster/ (accessed July 27, 2014).

Luc Besson and Robert Mark Kamen, Taken, Film. Directed by Pierre More. France: Europa Corp, 2008.

Tapster. “tapsterbot.” Twitter, https://twitter.com/tapsterbot.

Wikipedia contributors. “User Story,”Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/User_story (accessed August 22, 2014).