Sigrid Eldh and Serge Demeyer

The Next level of Software Test Automation

Preprint of a bookchapter, to be published by Springer

February 23, 2025

Springer Nature

Contents

List of Figures

Part I
Software Test Automation

Chapter 1
Reasons for Doing Software Test Automation

Abstract In this chapter we address the critical importance of test automation, presenting 41 compelling reasons why it is essential for modern software development. We address the substantial cost savings achieved through test automation, highlighting its role in reducing manual testing efforts and minimizing defects. We emphasize how automation serves as a key enabler for continuous exploration of new technology as well as providing opportunities for insights into the testing process. The discussion also underscores how test automation induces a broader quality perspective. Finally we list some considerations to take into account when embarking on a test automation trajectory.

Learning Outcomes

After reading this chapter, you should be able to . . .

1. Argue the advantages of introducing test automation.
2. Provide counter arguments on excuses to postpone test automation.
3. Consider the hurdles that you will encounter when embarking on a test automation trajectory.

1.1 Reasons for Doing Test Automation

Test automation is the way forward! The pros and cons are often debated and sometimes people need to be convinced. In this chapter we provide the arguments why you must adopt test automation. Even though there are costs associated, the return on the investment will be higher, if you do it for the right reasons and in the right way. Read the arguments, select those that is most applicable from your point of view, and then get started! As shown in Figure 1.1, gives an overview of the three major categories of arguments in favor of test automation: Save, Enable and Quality where each has subgroups. The figure also shows the considerations you need to take into account when embarking on a test automation trajectory.

However, there is one argument which rules them all: *Test Automation is fun!*

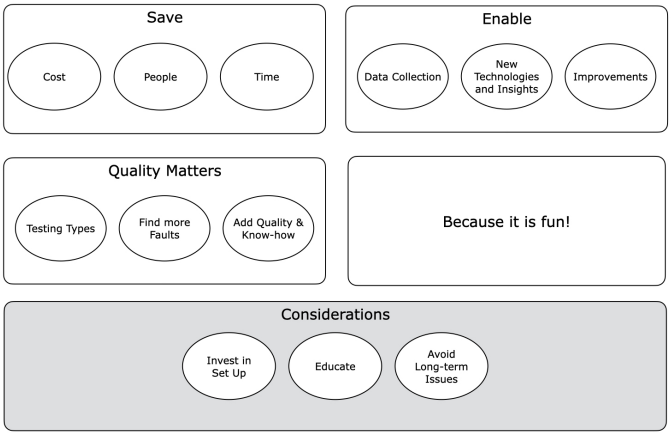

Fig. 1.1 41 reasons overview

1.1.1 Save Costs

Reason 1. Automate to save costs (limit waste, reuse, patterns)

The first and most important reason for test automation is to save costs on repetition. When the initial set up is done, the cost of producing new test cases is low, and regression testing for every new fault will save you a lot of time and money (not to mention the minds of your testers and developers). If you automated your entire build and test flow, the process throughput will be fast e.g., test design, test execution, test result (verdict) will be automated in a flow. The initial work to set test automation up for the first case, is often much more work (higher cost) than anticipated. Doing "the first test case" feels much slower than doing it manual, but - when this has been achieved, the production of more new test cases is much easier and faster as the similar test patterns can be reused, and a library of test cases will eventually be created – that can serve as a basis to combine and explore into new test cases. And simply – having a variable as input – means that you can test “all” inputs for one test case – without having to write many new test cases. It is therefore easy and saves a lot of effort and cost to expand your testing.

Reason 2. Reduce test repetition for different contexts and environments

For software that needs to exist in many contexts integrated with in many different environments, there is a huge saving to automate the test cases, since repeating the test suite in different integrations and context will be a minimal addition. It will also be very clear in what way these repetitions of test differ, pending on the surrounding. This could be the case for embedded software, and software executing on different hardware and hardware combinations, different browsers etc. You might not need to

re-test the entire software in the "new" environment, if you have a plan for what you test in what configuration. Limitations and resource tight aspects should though be considered. It is often easier to just check what adaptations are needed, and then you check that those specific adaptations are tested.

Reason 3. “Any aspect” of test can be automated

In principle, anything in the testing can be automated, but the return of investment (ROI) to spend this money vary on some of the phases in the test process, where still humans are better to understand what needs to be done. How much you save depends on what type of testing you are doing and in what phase, and with what tools, in what system. In the end, this comes down to your skills and knowledge. Therefore, it is important to analyze the effort it takes and what the benefits are. A good goal is to automate the entire test flow, which you can view both in the development and test process, from requirements creation to release of the software, as well as from an individual test case life cycle: test design, execution, verdict, and result. More important is that you can start testing as soon as you know what the system is to be doing (what correct is!), which is often revealed in the requirements phase. Most money you save in regression testing, but you can also save time if you are generating your test automatically from a model or when conducting performance testing. In principle, computers are great with repetition and with managing large amounts of data!

1.1.2 Save People

Reason 4. Save cost on training

If you have a common way to write and store test cases, you do not need to re-invent the wheel when moving testers around, i.e., all test cases are stored similarly, and can be followed in detail exactly how they are to be executed – so you can learn how to write test cases and what they do. "Then you lose the creativity of the people and get stuck in a structure!” is then a claim, but this is not true, because the creativity is not in the templates, it should be within the actual test case. If you have a good template or even a set of good test patterns, half the job is done when you automate. If you do it manually, you waste time understanding people’s different structures and way of writing the test case – some have details, and some not – and you never really know what was done in the test case.

Reason 5. Reduces risk of know-how staying with few persons

The cost of training testers is high since few universities offer sufficient test courses. Therefore, know-how of testing is expensive. Often testers possess the best system knowledge. This is then transferred into the test cases. Thus, the quality of the test itself will be dependent on the know-how of the tester. The fact is if you automate the test – all details can not only be reviewed and spread to everyone, but

it is also possible to learn from each other – how good tests are written. If you are testing manually- it is often very tedious and seldom are all details written down, leaving room for confusion, variation, and different results. This can be both good and bad. Therefore, a written down test case in detail spreads know-how between testers – and any test can constantly be improved.

Reason 6. Common Ownership of the software and tests

If tests are automated, anyone that has access can go in and understand how the tests are conducted for this area and learn how to test and execute this piece of software. This makes software a common ownership, and there will reduce dependencies on people who are experts on a specific task. Common ownership (if not too big) enables a common view of quality – what can be expected from the tests and the software. The risk with common ownership is that "no-one" is responsible for quality and tests. In smaller teams, one can take turns to share responsibility. In larger organizations – it is probably wise to assign people for certain responsibilities and make sure the person not only have the skill and the time to assure this responsibility, but also sufficient power to ensure common set goals are fulfilled. This can often lead to "improvement programs" in specific quality areas, taking care of technical debt, or making sure people have incentives to produce quality software.

Reason 7. Avoid boring your testers, developers, and users

Manual repetitive work is really brain-draining, boring and makes people feel more like a robot than an intelligent tester. By automating repetitive tasks in testing, you only need to do it once. Automation is also more creative and detailed than a manual test (who might miss details). To succeed, there is a need for a diverse plan and a more long-term strategy that address what to be automated, i.e., checked and how to handle tests that still are not automated. Be wary of automating the wrong things or tackling automation in a similar way as you perform your manual tests. This is often ineffective and thus bore the people involved in the project. It is also important that both developers and testers are aware of their responsibilities and the plan, to avoid overlap or untested areas.

Reason 8. Special skills for Test Automation

Test automation requires not only testing skills, but also requires the ability to develop automated test cases and incorporate them in the frameworks adopted within the organization. This is something that many people foresee when they wish for more automated tests. This means that it is probably not your average tester who take on this responsibility, because you need special skills that sometimes can be more advanced compared to normal development. This is also why many in agile development focus so much on the unit tests – for a developer, this comes natural as automating test, and testing the source code on code level is just programming as usual! Instead, a well-educated tester representing the user should be used instead. In any case, you need someone that tells you what is correct and defines clear goals and requirements – functionality that traverses the system, good use cases and user stories. Not only do you need quick feedback from your test suite what did work on

not – all in a team can use this fast feedback – and see progress in the development, what works and what does not.

Reason 9. Automated testing can mimic any user

If you are manually testing your system, you will only represent the users that are like your testers. If you automate, you can mimic "any type" of user and behavior of the system through different test design techniques, and then reuse these automated tests over and over. Of course, there are still some types that are difficult to represent, for example hackers, but we still will cover more types compared to manual testing.

1.1.3 Save Time

Reason 10. Save time in test execution and on the critical timeline

By automation of your tests, you will save time when testing is on the critical timeline, e.g., before release." Assuming fault free software this phase can be reduced to almost zero, compared to the manual task of testing the same functionality. Of course, reducing time to zero is impossible since it still takes time even for automated test to execute. And sometimes tests will fail, thus the fault should be located and fixed – and then your automated test might have to be updated again. Automating your tests can be done at the same time code is developed, and as such minimize the time on the "critical" timeline to release. You will simply have more time to develop and fix faults, as tests are run as soon as the system builds and is executable.

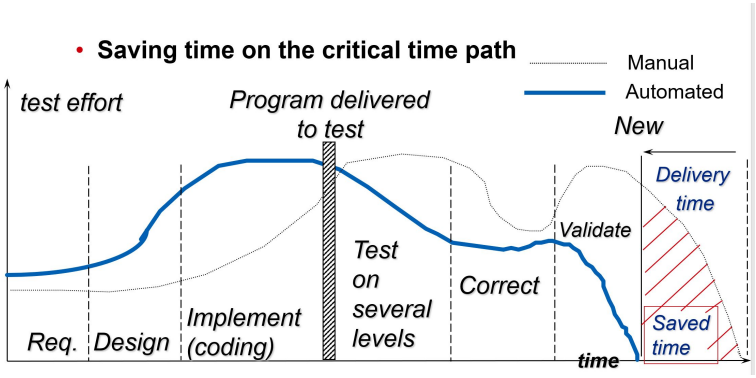

Fig. 1.2 Save time on critical timeline

Reason 11. Saving time in regression test

If you find a fault/bug in your software, the cost of re-testing the entire system (with the new software change) with automation will be negligible. It is common to

regression test your software multiple times if you have multiple bugs. Over time, these cost saving will be substantial. However, covering the whole system with your automated test suite is very challenging. For some systems, dependencies, timings, and resource impact can have a lot of hidden dependencies that are not triggered by the test suite. Therefore: know your system, know your bugs and know your tests (even if they are automated). Note that if you test in an agile way – you automation should run "green" – i.e., without faults before release in a best scenario. It is a waste to retest everything, since not all is broken, you only need to re-test the bugs which have been fixed.

Reason 12. Makes software development efficient through instant feedback

With test automation you can speed up the test time because it allows you to get frequent feedback on the code quality, as a health check and thus make the development more efficient since the feedback will be instant if something is not working any more. Fast feedback is important to remember what you were doing and have a speedy way to continue your work.

Reason 13. Speed up development by removing unnecessary variations

By reducing variation of test set up, i.e., faults, things will be much faster through the pipeline. Variation slows down project pace according to the ideas of Lean. Variation can mean a lot of things, but most of the time it is something that disturbs the normal workflow like handling defects, having different ways to set up the environment or to manage, write and report tests and test results. The smaller the variation is in "how" all people do things, the less impact on the project. The variation should be within the test case (and how they test the software). Automated test cases that are executed continuously in conjunction will small integrations, instead of big bang integrations, will make the variations smaller and thus not impacting the development pace. Therefore, test automation needs to be an integrated part of the development process.

1.1.4 Enable Data Collection

Reason 14. Measure the software quality through sampling or coverage

Testing is a way to collect information about the quality of your software, through a "sample" of executions of the software, a combined judgement on the quality. The collection of numbers of test cases passed and failed and of how many test cases existing becomes a number. Through automation of the e.g., code coverage and collecting data about your software system and your test, this number can serve as an estimation of the quality of the software. It can only be done manually if you only have a few test cases, because counting it is very time consuming – especially if you repeat it often e.g., in regression testing.

For code coverage is impossible to do without automation. "But code coverage is not a good measurement of test!" This is partially true, but it is a lot better than

"opinions" and gives at least some indication of how much you tested on the unit level. Over time, these metrics (who are often cheap to introduce in an automated context) could be added up with other quality metrics to provide an overview that is more statistically correct. One possibility is to introduce "weights" to reflect importance based on human knowledge. Note that there are many different types of code coverage – and not all are poor, but all metrics can be misused. See more about code coverage in chapter 2. Test automation simply enables an easy and factful way to sample the quality of your software and use code coverage to measure how good your tests are.

Reason 15. Collect data from all aspects of process

You can collect data from the process, from the execution and from the software. This can be used to analyze quality over time, predict release times and quality, and aid in focusing effort through analytics of e.g., usage of the software. It is easy to say but collecting data for a multitude of purposes in different formats from different tools, in real time, aggregating it and storing it can take a lot of effort to set up, and is of course very time consuming and error prone if you need to collect this manually. Through automation, and once set up – this can much more easily be collected into useful dashboards, that look at trends, progress and quality indicators of the software, and also help the organization to predict resource efforts, time until release, and improvements since last version in quality etc. A problem is to know which data to collect and save - and for how long, but also to know when data should be discarded. The consequence could simply be that discs are filled up with "nonsense" if you collect the wrong data. Taking regular baselines – a download of your data is also probably a good thing. Sometimes there might even be other legal and security aspects that need you to collect this data.

Reason 16. Certification of software and test

If test automation is used, it is possible to get the test suite, and the software certified to a standard or safety regulated area. It is much more difficult to show that a manually tested software complete, accurate or sufficient. Even though certified tests are often "old" in the way to execute software and as such, not so common anymore (except for specific protocols) and sometimes standards, they still exist for safety critical systems

Reason 17. Legally sound practice

If you automate your software and follow good practices of taking baselines, it is always possible to go back and see exactly what you have tested when and how at a specific time. Then you can prove legally, that you did test the system to a certain standard at a certain point in time. This is almost impossible to prove (in a court) for manual testing. This is especially important for safety-critical systems. In some cases, the test report contains sufficient information, if you have done a proper job. You also need to baseline your tests, and test tools along with the software versions, what test was executed, at that commit that caused a specific bug. Taking baselines for example at every commit of test cases, tools, framework, and the software also

aids also to seem who did what and when. It is very hard if the tests and software is constantly changing to keep track of the history and learn from it without automation.

Reason 18. Automated test enables good practices

If you have construct templates and a common way to write test cases, you can get a good documentation at the same time you write your test cases. Common ways of working can lead to a good quality aware way to produce your software. Test automation enables all to create a common and good practice for test, since it is also easy to see if some part of the software is not tested, something that is much easier to hide with manual testing. The important thing is to analyze and structure the work before you begin the automation. If this is done in a good way, tools like e.g., Doxygen can help you extract the correct information when you need it. At the same time the documentation is always up to date and thus the cost for maintenance of test case documentation is reduced. But as pointed out, it is necessary to define a good structure from the beginning or otherwise these cost reductions will be lost.

1.1.5 Enable New Technologies and Insights

Reason 19. Towards autonomous systems – self-running, self-testing, self-fixing

The future is here, a way to reach the ultimate goals., i.e., self-running tests and self-healing faults. It might sound like utopia, but it is not a long way there. Today, we have self-driving cars so self-testing system is on its way! On part of achieving this goal is of course to start building automate tests and test systems, in an automated framework of build and test, with automated baselining, data collection and automated fault localization, debugging and fault fixing. The main issue – as always with testing is: You need to know what is correct – what you are aiming for.

Reason 20. Use of new and automated test design techniques for better testing

This means if you have automated approach to your test cases, it is easy to, for example, automatically do a specific test design technique, e.g., introduce a set (variable) of parameters, instead of only testing one parameter to expand a test for checking boundary values, or using the more advanced search-based testing (evolutionary) testing method that will automatically identify all boundary values that fulfills the test case for this criteria. It is easier to loop across the entire software for similar test, e.g., to check a user interface field is correctly tested. You can today spawn all “routes” through a user interface, but beware, with different data in the variable fields – the test can also easily “explode” to many unnecessary clones. With automation optimizing a test suite or identifying values that makes the execution pass or fail is also possible. Not to mention testing performance – almost impossible to do without automation. Note that some techniques can equally applied manually as well, even if it is tedious. Your effectiveness relates to how it is done, and of the domain!

Reason 21. You can evaluate the quality of your test suite

By automating your test suite, you make it possible to use techniques that evaluates your test suite quality, e.g., mutation testing. This might be necessary for testing safety-critical systems. Here we see the same risks as with creating automated variations. First, we might create variations and mutations that does not add a substantial value to the testing. There is also a risk that all the non-valid mutations require too much manual labor so that the gains will be lost. New automated techniques are of course making progress in taking these risks away, but it is an option if you need quality software – or maybe just a part of the software.

1.1.6 Enable Improvements

Reason 22. Parallel development

If you automatically test your software, you can create test code at the same time as the code is created, which means you can parallelize the software development to a faster enabling. This is also known as Test Driven Development (TDD), and one of the elements for successful automation.

Reason 23. You can test much more of your software

You can easily test through much more of the possibilities your software, e.g., more data, more paths, more sequences, by just making setting up your automated test cases to automatically vary, loop, and traverse your software. When you include the effort of setting everything up to get started, the total amount of effort might be the same (or higher), so you should consider what type of system, tools, knowledge etc., you need to test. If all aspects of the system are different, the cost of testing can be very expensive.

Reason 24. Makes automated test verdicts and test oracles possible

For some domains the problem is not to "execute", but to determine if the result was correct and to judge the result. This might save a lot of time if this is clearly defined. It is also an important step in the fully automated test flow. We must be aware that people are (much) more effective in judging test results, and they are the test oracle in most cases, they know what is correct. This knowledge can be difficult to transfer into automated tests.

Reason 25. Non-functional system testing

To put stress, high load and overload, or a high volume of data and/users into a system manually is almost impossible, but through automated testing this is both a feasible possible solution. How could mimicking 5000 users be better than an automated test that re-peats test execution (test cases) for 5000 users? Mimicking many users plainly pounding small scripts often runs into problems, since it is "dumbly" repeated and creates unrealistic loads and congestions. Therefore, we

should be aware of how we design the operational profiles when doing this kind of testing. Other types of non-functional testing than performance testing could be to test interoperability, availability, robustness, reliability, stability, usability, or security. Some area easy to set up with automation, other are more demanding.

Reason 26. Continuous delivery

A comprehensive automated test suite enables modern practices like continuous delivery, which makes it possible for your business to respond quickly to user feedback. Test automation is essential when setting up an environment for continuous delivery. Without the proper security net for quality, this type of environment can quickly become disastrous if the delivered quality is not satisfactory.

Reason 27. Highlights software design issues

If something is hard to test it is likely the software design is not good enough. Also, if you struggle with failing tests, the root cause can be design issues. Test automation will not only find code defects, but it can also highlight design and architectural issues as well.

1.1.7 Quality Matters for Different Types of Testing

Reason 28. Early test of requirements

If you automate your test design, by defining and describing a model that produces test cases, you will have an early check that you have a good and complete understanding of your software. The additional tests of the actual software will be minimized as most normal test cases are generated by the system. Capturing, specifying, and exemplifying requirements is hard. It is often difficult to determine that you have captured all aspects (including e.g., fault handling) until the entire process is done. This means it is also important to model negative tests (tests that address what is not allowed) to create a more complete model. It is better to develop iteratively, than trying to capture all your requirements up front.

Reason 29. To secure the user interface

Making a user Interface secure and fool proof, automation can test combinations in ways no human can think of in a reasonable time at a much cheaper cost. This is often called “monkey testing” and doing it manual is both tedious and repetitive. Automation will provide better coverage of different combinations. If no human can think of them, why would anyone need to test or execute those weird combinations? Even if we do not think of them immediately, users tend to do a lot of strange things and therefore this is a good approach.

Reason 30. Testing safety-critical systems

For safety critical systems, like trains, planes, cars, emergency, and medical devices, to mention a few examples, automated testing is a way to make sure that

you have done a "complete" test of your system. The more aspects of test that is automated, the higher the possibility of a safe system. You still need to design your system specifically for these aspects because testing does not fix poor understanding of functions, requirements, or architectures.

Reason 31. You can ensure reliability

If you automate your tests, you can use technologies to measure the automated test of the software, and automatically inject all kinds of faults into the system. With other words, you make sure the software can handle all kinds of possible faults you can imagine, and then ensure the software is reliable.

Reason 32. Add more variation in your tests

By automating your tests, you can easily create variations of the test case, and explore new areas, that a human would not think of. Loops, random variations, permutations of order, and variable input data are simple additions to your test cases that can find more faults and add coverage. With automation – you can gain good variation of testing which leads to an improved coverage, and this leads to many test variants that we otherwise would have missed.

1.1.8 Quality Matters to Find More Faults

Reason 33. Find more faults

By test automation you can find more and new faults, even in software that has been manually tested for years. When you bring in automation, you will find a new set of faults in most systems. But are they important faults? As with all defects, it is necessary to classify them and do root cause analysis for those with high impact.

Reason 34. Find security bugs with e.g., fuzzing

With automated "fuzzing" technology, and testing for specific combinations of faults, you can find security problems in your software. Even though hackers always find ways to circumvent these issues, it is good to try to secure as much as possible from the beginning with help of automated tests.

Reason 35. When human lives are at risk!

When manual human error could not be tolerated and human lives are at risk, there are automated “test dummies” that allow the device under test to be hooked into a complex test framework. Here the dummy simulates the conditions a real human might experience. The automation framework allows many scenarios to be tested that could be risky for a real person. An example is found in some medical systems where the loss of human life is a sufficient risk to require test automation. Consider a smart device that is to be implanted in a human to keep the person alive. There will be many tests before implanting the device in a real field trial. The automation framework allows many scenarios to be tested that could be risky for a real person.

1.1.9 Quality Matters as you will Add Know-How

Reason 36. Secure quality through repeatability

If you do the same thing repeatedly (regression testing), the re-test will be quick (and if you had a fault all can repeat). We seldom find faults more than once in a code. It is true that a lot of faults are found when doing the automation, i.e., the first time it is executed. But, if software lives longer than a few iterations (and software does), it is often worth it. Claims say that about three to five regressions are often enough. Also, if you are doing CI/CD and continue to fix and update the system, it will be worth it. You need to make sure you did not break the other parts of the software when you do a new change somewhere in the existing software. It is tedious to test everything again, when you have had a fault and especially of you do not do a complete soft-ware dependency analysis, to identify the affected parts.

Reason 37. Gain confidence

You will be more confident about the quality when you had executed all your automated tests. By automation you have the possibility to execute your test cases faster and more often, which will lead to better belief of the quality of your software. This will gain trust in your test suite.

Reason 38. Recognize when automation is the best choice

Testers need to be thinking about situations where the manual approach is not a good option for project test planning. Too often, many testers default to manual testing because it is what they are most familiar with. There are occasions where one should use manual testing, i.e., to learn a new system. Meaning – all testers and developer should run some of the test cases manually to get a hands-on experience and feel for the system – and to better understand what the system does. Manual testing could be an option if software will not have a very long life, e.g., you are making a demo of a specific aspect, or exploring new aspects and you are e.g., prototyping or trying out different designs. Beware though, that many so called "prototypes" are not thrown away and both these and demo's easily end up as the embryo in your development. Then instead you started your development journey with technical debt at probably a very hectic time of growth, where test automation framework instead could have saved you to get a fast incremental growth of software. Planning your test, understanding your goals, and making sure there is enough skills in your team for both test automation and manual test is therefore very important.

Reason 39. Automation is a necessity for fast systems

When the system computation cycle time (output to required input) is faster than a human can respond, we need automation. This is evident in different domains, e.g., systems for airplanes, robotics, telecommunication, and aspects of self-driving automobiles. It is also a simple and powerful way to determine what to automate.

Reason 40. Automation is a necessity in complex and multi-functional environments

When the system has complex user-machine activities where failures can have significant financial or resource impacts. Just listing the kinds of users, the Internet of Things (IoT) software systems have, you are likely to see testing a smart city becomes a huge manual task. A possible list of IoT smart traffic light users includes citizens on the road, police, database systems, communication systems, city managers, operations staff, power systems, machine learning route systems, and smartphone apps. Automating the testing for the many varied devices in a smart city is the good choice, or the only choice.

Reason 41. Because it is fun!

Yes, this might the most important aspect. It is much more fun to create automated tests compared to doing manual checking. Therefore, we should ask the question of "who would like to do a boring job?" Can you find our 42nd reason? Please mail us new unique reason for automation!

1.2 Disadvantages of test automation? Not necessarily!

As we said in the introduction: there is an upfront investment in test automation. But even after this initial investment there are longer term implications to consider to avoid failure in your test automation trahectory. We have divided them into three areas: (i) Invest in set up of your test automation, (ii) Education and (iii) Avoid long term issues.

1.2.1 Invest in set up of your test automation

Consideration 1. Extra time and resources

If you have an ongoing software development, both your testers and developers are fully booked. You will temporarily need extra time and resources to set your test automation up. One way to do this, is to do it gradually, and plan all aspects into the regular way of working. Once your environment is set up – you can then gradually add all new test cases into the automation, and slowly transform your regression tests into automation. You will soon find that the more you automate, the less time you need to spend on test execution. But setting up your frameworks and tools and prepare for automation – this simply takes both time and effort. Therefore, it is much more effective to automate when you start a new software.

Consideration 2. Evaluate tools and integrate

There are a lot of free and available test tools and test tool frameworks that can easily be used. You also need to consider what process, domain, and development environment you have. It is important to think about test tools in the context of how

you are working. Testing is "not just" a testers concern – you should consider how this fit with your software development at all levels. If you are embarking on an agile CI/CD (Continuous Integration/Continuous Deployment) way of working – you should consider having an integrated test tool with the build tool – build and run (tests). Today there exists many opensource build and run frameworks, e.g., Jenkins, Maven and GitHub that easily works together with common test tools. You need to evaluate what frameworks that fit through e.g., pilots, or based on understanding your way of work, and spend time teaching your organization. Best is of course if you have people with this experience, and frameworks that are easy to set up and use – and that works well in your test environment. Also note that often one tool is not enough to handle all your testing needs. Therefore, it is important to integrate your tools so they can automatically build and run smoothly along with your software and test case creation.

Consideration 3. Cost of tool licenses and other tool issues

If you do not go completely open source (which is "more or less free") but could also mean that you are giving away some part of your software (check licenses deals carefully) or build your own automated test tool– as there is a cost of buying a tool licenses. You should also consider if the tool scales to your environment, and how easy it is to integrate the tool in your environment. Some tools require you to learn a specific "script" language, like e.g., TTCN-3 to write your tests. Or – you can write your tests in the same language as your software. Many use e.g., Python or TCL for there test scripts, but there are so many more languages out there – with lots of other advantages – When you do a cost calculation of a tool, also consider languages the tests are written in, popularity and aim of these tools. In the future – you might be able to automatically capture a text written test case into a format that a computer can use, that then can be optimized.

Consideration 4. Create "new" test cases that suits the automation

Beware that you write your new test cases fitting for the automation you have. If you have a safety-critical system, you might consider more formal ways of defining your tests and generating them. If your focus is to automate your developers' tests – consider the create your test suites in the same way you create your source code. Use case testing, functional testing and behavior tests can also differ in how they are automated compared to e.g., the unit tests. Often different type of test has tailored tools – and might not be easy to move tests across different tools. Define the tests so they suit the way you are working. It is a new way of thinking when you write your automation. An automated test case can both traverse the system, loop, and test with a large set of data for the same variable. If your already have textual manual test cases – and are in thinking of automating these, remember that you know that manual test cases are really "one" way of writing the test case path through the system often with hard coded input. With automation, you could easily expand input into variables, and test much more that you could otherwise. You also should look over the overlap or "cloning" that you might have in your manual test cases.

Consideration 5. Create test case templates and test libraries

When you set up your automation, the first test case will take long time, but can then serve as a "template". You should always think on things you repeat that could go into a library instead of copy-pasting this part of the test case. Utilizing test templates that has a set up pattern, a clear input associated with the expected outcome (output), a test oracle (some form of observation or comparison that determines the outcome) and a set of actions. After this you might also need a tear down pattern. This is to be able to re-run your automated test cases, which could include deleting and removing content created in the test case. A well written test case template describing a set of patterns, what you need to document (in e.g., a header of the test case code), can save a lot of time for the testers to automate and identify similar test cases.

Consideration 6. Savings depends on number of regressions If you are to do a calculation of how much time you save, it is easy to say how many times you "fix bugs" and how often you are repeating your testing (regression testing). On average 90% test execution time savings can be expected from each regression after you automated. The advantage is often that all tests can be re-executed in an automated regression suite, but if you were manual, you would only test the fixed bug. But you also need to deduct the cost and time from the set up. Most businesses that keep updating and improving on their software finds the time to market very valuable, and as such, having a quick quality check is more valuable than the initial cost of set up. As we discussed, looking from it in pure business terms – the quality increase, the speed increase, but also the pure fact that people manually cannot compete in the long run, will well compensate for the extra investment that has to be done.

1.2.2 Educate

Consideration 7. Learn programming and writing smart automated test cases

Manual testers might need to learn programming, as some have mostly been "users". Use your developers – who should be skilled in programming to teach testers, but also aid in how the test case can be automated. Let the expertise of the manual testers on the system behavior oversee the transfer. This would even out the skill-gap, and not ignore the manual testers competence, as well as expanding the developers know how on how testing is done. This transfer and joint work are not the easiest to make, and professional and (external) support and courses is often needed on new test code patterns, different scripting languages, new (test) tools, test libraries and much more. People can often rise to the challenge if mutual respect and time for support and education is given. It is easy to forget that testing is also a unique knowledge that should not be ignored. Writing good, automated tests at all levels of testing is not a simple task, and many developers have not learned how to effectively write test cases. Take time to learn new test design techniques to do smart automation.

Consideration 8. Learn new tools and new ways of working

The way of working, the development context, and the new automated test tools should be part of a training program. Best is of course if you have a champion in the tool, that can lead the way, answer questions, and support the others. Today most test tools (if not self-created) come with many instruction videos – and the internet is a great source of specific information on how to solve issues. Transferring to automation can be a change in the development and test processes, and the new responsibilities, roles and way of working should be taught and discussed to yield the best outcome.

Consideration 9. Change behavior, process and sometimes attitudes

The most difficult part of changing a historical manual testing team to run test automation is the change of behavior, process, and attitudes of these teams. It is natural to humas to resist change – as we often feel more secure when things are stable. Being motivated to change makes everything so much easier. Most issues are solved if we can allow ourselves to take the time. Changing into a modern test automation way of working is a substantial change. And there are many that can guide and help how this change is happening – and challenging and developing yourself and your colleagues will have a positive reward for all involved.

1.2.3 Avoid Long-term Issues

Consideration 10. Loosing know-how of test and the software/system

It is easy to lose know-how of what the test cases contains, when your test suite grows. Then you lose the feeling for how the system and software behaves and reacts. If you don´t regularly review your own and other test cases, you will soon lose sight of what your test suites are doing. To remedy this problem, regularly train both your developers and testers on how to run some test cases manually as well as review and refactor your test cases to keep them up to date. Make search easy to find similar test cases to be used. If you cannot easily go in and "fix" a problem or update a test case – it is a sure sign you have lost that touch. If you do not know what you are testing – you might be in trouble!

Consideration 11. Innovation of new tests – lack of creativity in testing

It is easy to lose your innovation and creativity of doing new test cases. Without checking your test design, you end up repeating the normal test case, and you only follow existing test patterns. Therefore, it is important to be active and do new test design techniques, new approaches, and new ways of variations of existing test cases that explores new code. We also need to learn how to be creative to find new bugs. Also make sure you learn from existing bugs in the system. One example is how hackers find security vulnerabilities by trying new combinations of issues that no

one thought of. Make sure that you constantly improve and refactor your test suite to avoid getting a stale test suite.

Consideration 12. Chasing the "wrong" bugs

It is easy to focus your time on the wrong things in automation. Many spend a lot of time on finding e.g., flakey test bugs in the test suite. Many of those bugs are simply because we did not test enough, assumed wrong timing or order in the context of our test case, did not use memory test tools etc. There are a lot of reasons – often related to the test environment set up (hardware) that might be the cause for the test suite does not go smooth.

Consideration 13. Over-confidence in the test suite

It is easy to get over-confidence in the test suite. The higher number of automated tests you have – the more likely you are to be "blinded" that you have now "tested" your system. But beware, a high number of test cases is not a guarantee of a good test suite. There are probably still bugs lurking and unfound, and there might be a lot of overlap between your test cases (called cloning) – so you are testing the same thing over and over. The only paths you know works, are those you automated. But there always plenty more and test cases grows old. You should for sure have at least as much turn-over (or change) of your test suite as your source code. Make sure your test suite is decent, by using fault injection techniques and/or mutation testing to evaluate its quality.

Consideration 14. Thinking you do not need to invest more in testing

The last, and maybe most important point is the danger of thinking you do not need to invest more in your testing. As software lives longer, and if it is used, you will probably find new bugs that needs to be fixed – and if you do not test it – this can cause the entire system to break. This phenomenon is called software rot. If you keep changing small and only testing the fix, after a while – without refactoring and updating your test suite, the test suite will no longer be useful, and the software system deteriorates with more and strange bug coming alive. With automation, this phenomenon takes longer time to happen, and is also not as common compared to manual testing. Remember, that if your software is alive and running, it is useful and then it must both have bug fixing and testing support.

1.3 Conclusion

Automate efficiently and effectively – Find more bugs, get better measurements, enable new techniques, and make serious improvements to your software as you save costs, people, and time – and make sure you avoid those long-term issues. Test automation is fun!

Study Questions

1. You are a team member of an agile team (5 individuals) working an app for following the stock exchange market. You are already passed five 2-week sprints into development of the initial prototype; a minimum viable product. At the end of the sprint you want to convince your fellow team members to start automating tests.
   Which one of the 41 reasons will you use first to convince your colleagues?
2. You are a team member of an agile team (5 individuals) working an app for following the stock exchange market. You are already passed five 2-week sprints into development of the initial prototype; a minimum viable product. You convinced you colleagues that test automation should be adopted in the next sprint. But to actually commit resources you need to convince the Chief Technology Officer (CTO).
   Which one of the 41 reasons will you use first to convince the CTO?