Case A

The organization

This case takes place at a large and established financial services organization. Since their inception, they have operated as a project-based organization, with many long-lasting waterfall software development projects. One year ago, they have decided to adopt a Continuous Delivery approach to software development and delivery to be able to better keep up with upcoming competitors.

The team and the product

The team consists of 4 developers, a tester and a scrum master. They follow Scrum and work on a mission critical batch processing system that calculates monthly payments to clients based on a large set of business rules.

The system is quite complex, it is said that it will take new hires around a year to get really familiar with it. It has been around for more than 10 years, and there are very few people in the organization that know about all its nuts and bolts.

The testing process

Most testing is done by the tester in the team, in cooperation with business representatives who have a final say in accepting the software changes. Because of the repetition in their work, and the ever growing pressure to release faster, the team is looking to implement test automation.

The developers already adopted unit testing, and they pride themselves in maintaining a high level of code coverage, especially around the complex part involving all the calculations and business rules. Despite this, a lot of regression defects slip through and make it to the tester and the business representatives. As a result, the business does not have a lot of confidence in the work delivered by the developers.

Automation challenges

To increase the level of trust of all parties involved, the team decides to implement automation of acceptance tests. One developer is tasked with the development of a testing framework that can operate and test against the system under test. Unfortunately, the tester is too busy with extinguishing fires to contribute a lot, despite them having a fair level of experience with automation.

When the framework is delivered, the developer gives a demo to the team and writes some documentation on how to write and run tests using the framework. Unfortunately, that’s where this part of the automation efforts stalls. The developers rather work on features, the tester is too busy putting out fires, testing new features and communicating with business representatives to make a contribution.

Case B

The organization

This case takes place at a semi-governmental organization. Having long been a traditional project-based organization, they've started to adopt Agile and Scrum principles about a year ago.

The team and the product

There are several development teams that all work on different applications in the IT ecosystem of this organization. These applications are not operating independently, since they share data and transactions by means of an Enterprise Service Bus (ESB). All teams consist of at least two developers, at least one tester, a business analyst and a scrum master. Most teams work on premise, but part of the software development and associated testing activities are nearshored to another country.

The testing process

Development and testing for the features where the scope is limited to the team itself is largely going fine. Developers develop, testers test and not a whole lot goes wrong. It's the features and bug fixes that cross the boundaries of the team that create problems. Each development team has limited knowledge on what happens when they change a message or a process that impacts other systems, because these are developed in other teams.

At some point, there has been a team (the 'ESB testing team') that was responsible for testing across the entire IT ecosystem, but a year ago management decided that each team should from then on be responsible for testing the impact of the changes and features they developed.

Automation challenges

For testing the changes across the entire IT ecosystem, an automated test suite exists (a remnant from the time when the ESB testing team still existed). There's one test automation engineer that dutifully maintains it. It does only cover those parts of the processes and the ecosystem that they know about, though, meaning that coverage is still rather limited.

The organization really wants to focus more on integration and end-to-end process testing (tests that traverse the ESB), because the potential impact and damage due to bugs can be high: there's a lot of money involved with the day-to-day business of this organization. However, lack of time and lack of knowledge about the entire ecosystem is working against them.

Case C

The organization
This case takes place at a small organization that develops software for higher education institutions (universities and colleges). They are currently moving from working project-based and waterfall to adopting Agile, with the ultimate goal of becoming capable of practicing Continuous Delivery.

The team and the product
There are three development teams (or rather 'feature teams') that all work on different parts of the main product that this organization offers. Each team consists of front end developers, back end developers as well as one or more dedicated testers and a scrum master.

The testing process
Because of the size and the complexity of the releases (the teams used to release once every three months), testing has until now been a long-winded process. Another complicating factor is the fact that, apart from the core product, all kinds of customizations are being developed and delivered to cater to individual customer's needs. This means that various versions of the product are in use (and need to be supported) at any given moment in time.

Automation challenges
In this case, virtually no test automation has been implemented, yet. The teams are struggling to get started, especially because nobody in the organization has experience with implementing a test automation strategy from the ground up.

Some experiments are carried out with low code test automation tools, because these provide the most gentle learning curve for the teams. These tools serve them well, but the teams are still in doubt whether or not it is all worth it. Building and maintaining the test automation takes a lot of their time and results are coming slow. The fact that the teams keep being under pressure from management to make deadlines (they haven't gotten rid of those yet, unfortunately) and the many customizations they need to test do not help, either.

Still, they do see that without a solid test automation strategy, they will likely not be able to reach their goal of Continuous Delivery.

Case D

The organization

This case takes place at a medium-sized telecommunications company, delivering triple play (Internet, television, landline telephone) subscriptions to consumer clients. All software is developed in house.

The team and the product

At the moment, there's a very traditional organizational structure in place. Software development is done on one floor of the building, testing is done by a separate testing team that is located on a different floor. The testing team is responsible for testing the entire business process around provisioning new subscriptions, upgrading and renewing them as well as cancelling, a process that involves interacting with at least 5 different applications, as well as some time traveling (for example to skip cool off periods).

The testing process

Because there are quite a few variations in products and processes, there is a significant regression test suite that is dutifully maintained and executed by the test team. Each release, the team has to go through the entire regression test set (around 150 test scripts, of which most take 10 minutes or more to complete due to the length and complexity of the processes). To save time and money, the test manager decides to implement test automation.

Automation challenges

Experienced test automation engineers are tasked with the automation of the existing regression test scripts. Because of the variety of applications under test, a multipurpose vendor-based tool is selected to help perform this task. To keep a close eye on the progress of the automation of the regression test scripts, an Excel sheet has been created in which the status of the test automation efforts can be administered. Red means 'to do', orange is 'in progress' and green means 'done'. A test script is considered 'done' when it has run in the test environment without errors for three nights in a row.

The project scope is being strictly monitored: the list of regression test scripts to be automated is frozen and the project will be considered finished when all scripts have been automated.

The test automation experts start this new project being excited. Progress is being made and soon enough the first short regression test scripts can be promoted to the 'done' status. After a while, though, things are slowly starting to take a turn for the worse: scripts that used to run fine now fail. The longer regression test scripts turn out to be hard to complete; with each test run these scripts fail at some point, to the frustration of the automation engineers. In the end, around 60% of the original set of test scripts can be marked as 'done'. The project is brought to an end, but in the end nobody is really happy with the end result…

Case E

The organization
This case takes place at a small testing services provider that performs testing and test automation services in house for its clients.

The client and the product
For a specific project, a number of test automation engineers were tasked with automating regression tests for a webshop that sells electronic cigarettes and accessories (charger, refill capsules, etc.) to customers in the United States. As this is a highly regulated product, it's very important to check with every deployment that all business rules that are related to persons of a certain age, living in a certain state being (un-)able to purchase a given product are correct. The fines for (either accidentally or purposefully) selling a product to a person who (by age, by zip code or both) should be prohibited to do so are significant!

The testing process
There are about 6500 relevant and unique combinations of age group, state and product category. A full regression test run should cover all of these combinations, making regression testing a time-consuming process. As the system under test is a web store, they want to release early and often to stay ahead of the competition, though, so in practice, regression tests were often skipped. However, since the client base is growing, so is the risk of violations of regulations around electronic cigarettes, a risk the web shop is no longer willing to take.

Automation challenges
Since the web shop is developed by a team from yet another organization in another part of the world, all that the test automation engineers have access to at the moment is a version of the web shop deployed in a test environment and documentation about the business rules that need to be verified.

Before the demand for a regression test that fully covered the aforementioned 6500 cases, a test suite with a much smaller coverage was built. Part of that test suite was an end-to-end test, running against the test environment, that performed a search on the web site, put a product in a shopping cart, proceeded to checkout and filled in all the necessary forms to place an order, so that the test automation engineers could demonstrate that the web site supported the primary buying process, or at least that it did under those specific circumstances.

The test automation engineers then opted to make that test script data driven, i.e., perform it 6500 times, with different input parameters for product, zip code and date of birth, so as to check the correctness of the business rules in that way. They soon ran into trouble: one iteration took about 45 seconds from start to finish to complete. When they wanted to run it 6500 times, it took them (give or take) 292500 seconds (= 4875 minutes = 81,25 hours = 3,4 days). And that's when the entire test run finished flawlessly, which it never did..