Gojko Adzic

Specification by Example

How successful teams deliver the *right* software

MANNING

Table of Contents

Chapter 13. RainStor. Specification by Example: How successful teams deliver the right software, ISBN: 9781617290084

13

RainStor

RainStor is a UK company that builds high-capacity data archiving and management systems. RainStor is an interesting case study because they deal with a technical domain where complexity comes from high data volumes and high-performance requirements, combined with advanced compression and data management algorithms.

The company has fewer than 30 employees, and about half of them work in Research and Development, so they have to be efficient in building and supporting their software. All the developers and testers work as part of the same Scrum team, although they're now thinking about splitting it into two.

Their journey to Specification by Example was almost organic, without any big plans or buzzwords, driven mostly by the testers. When I interviewed Adam Knight, senior test and support team leader at RainStor, he said, "Nobody else in the company knows what Acceptance Test-Driven Development means." Although their process has almost all the key elements of Specification by Example, they just think about it as their homegrown way of developing software. They illustrate requirements using examples, automate them into executable specifications, and validate them frequently to build a living documentation system. The changes they implemented allowed the development team to triple in size over three years, at the same time making them more effective.

Changing the process

Three years ago, a new CEO decided to implement Scrum and expanded the team of four developers by hiring two testers and a test manager. Although they adopted iterations and daily stand-up meetings, the process was actually a mini-waterfall, according to Knight. He explained:

> We had requirements as a large document put in at the start of the sprint. It was supposed to be both the requirements document and the technical specification. There was too much technical detail in it. It was set in stone at the start of an iteration. The development would go ahead against that, and testing was based on the contents of that document. The document was not maintained with the decisions made during the development process, so towards the end our test cases differed from the implementation.

In addition to the problems in coordinating development and testing, they had a problem with the way tests were executed. Although they had some automated tests, testers were running most of the validations manually. As the product grew, it became obvious that manual testing wouldn't scale. Even if they added more people to run manual checks, their software handles very high volumes of data, and manually checking queries that return tens of thousands of records wasn't feasible.

Knight took over as a test manager in late 2007. He wanted to make testing more efficient and to support development, preventing the need for manual testing as the product developed. They implemented a simple automated test harness, which allowed them to push testing much earlier in the process. They could define tests at the same time as they were developing the relevant feature. This helped them align development and testing.

Functional test automation gave them immediate value, because they no longer had testing tasks pile up toward the end of an iteration. It also gave developers quicker feedback as to whether a piece of work is done, removing the interruptions to the flow caused by testing spilling into the next iteration.

Once the team aligned testing and development, they started noticing problems with scope creep and knowing when they were finished. They often had to rewrite the requirements after development started. Boomerangs were coming back from the previous iterations in the form of stories titled "Finish off ...." During the summer of 2008, Knight brought in David Evans as a consultant to help them understand how to improve.

As a result, they started to describe scope with user stories instead of using large, detailed technical requirements up front. This enabled them to start thinking about acceptance criteria from a business perspective and deriving the tests from that, instead of receiving requirements in the form of functionality to implement. Knight said that this allowed them to understand the scope better and have a clear picture of when they're finished developing a feature.

They started to break down stories into smaller deliverable items, which gave them more visibility of what could realistically be delivered in an iteration. This helped the team to better manage the expectations of their business users.

The team then started to use examples to illustrate conditions of satisfaction, even for requirements such as performance. Knight explained:

> We used well-defined acceptance criteria for performance measurement. For example, the system has to import a certain number of records within 10 minutes on so many CPUs. Developers would either get access to dedicated testing hardware or testers would run tests and provide feedback.

Focusing on user stories allowed the business users to engage better in defining the expectations from an upcoming piece of work, and illustrating those expectations with examples allowed the team to measure objectively whether they've achieved the targets.

As their customer base started to grow, they were getting more customer-specific scenarios to implement. In late 2008 the team decided to reach out to the customers as the final stakeholders and involve them in the specification process. Knight added:

> Generally a customer would have an implementation they would like to put in. They would give us the requirements, and we'd work with them on getting realistic data sets and expected targets. We'd put this in the testing harness and use that to drive the development.

Putting the customer-specific scenarios with sample data into the system as acceptance tests ensured that the team met their targets. It also meant that the team didn't have to waste time coming up with a separate set of acceptance criteria and prevented any wasteful rework caused by potential misunderstandings.

This process worked best when they could involve actual customers with realistic requirements. RainStor primarily works with reselling partners, who sometimes request functionality without a specific business case. "That is the most difficult kind of requirement," said Knight. In such cases, they push back and ask for examples, sometimes organizing workshops with the customers to go through high-level examples on relatively developed prototypes. They use those high-level examples to drive the scope later. Working on paper prototypes also helps them look at the system outputs first, promoting outside-in design.

The current process

At the moment, the Research and Development team at RainStor works in five-week iterations. Sprints start on a Tuesday, with a sprint kick-off meeting in which they briefly go through the stories planned for the iteration. They use the rest of the day to elaborate those stories. The developers, the testers, the technical writers, and the product manager collaborate to flesh out the requirements and define some basic acceptance criteria for

each story. The testers write down the acceptance criteria based on the notes they took during that meeting and publish them for the whole team to see.

Once the conditions of satisfaction for a story are published, development and testing start in parallel. Developers work on getting the existing specifications with examples to pass and testers work on creating more detailed test cases. For some stories, they might not have any examples automated at the start. In such cases, the developers initially work on delivering the basic functionality while the testers automate the simpler examples. The testers then go on to develop further tests, and the developers deliver the functionality that ensures that these tests pass.

As the functionality is implemented, the testers perform exploratory tests and start to run automated tests against the new version of the system. When the full functionality is implemented for a story, the testers ensure that all the tests pass and then integrate them into the continuous validation system.

Top three ah-ha moments

I asked Adam Knight to single out top three key lessons he learned about Specification by Example. Here's what he said:

- As you develop a harness of automated tests, those can become your test documentation if you set them up properly to reveal the purpose behind it. Metadata made the tests much more readable. We produced HTML reports that listed tests that were run and their purpose. Investigation of any regression failures was much easier. You could resolve conflicts much more easily, because you could understand the purpose without going back to documentation.
- The acceptance criteria and specifications with examples created as part of the story process became the requirements. You can have a lightweight story to get started. Once you have the tests, passing those tests tells you that the requirements have been met. You don't have to refer to anything to find the requirements. That allowed us to spot if future requirements were conflicting and what the changes were impacting. It allowed us to maintain the requirements on an ongoing basis. We were always in a position to know how the product sits against the requirements that we implemented. If a test starts failing, we know which requirement wasn't met.
- The test and test results were part of the product. You need to store them in the version control with the product. We test different branches and versions and we need to execute the tests appropriate to the branch.

Some tests work with very large data sets or check performance, so they divide the continuous validation into three stages: regular builds, overnight builds, and weekend builds. Regular builds take less than one hour. Slower checks run overnight. Checks with very large data sets, often customer scenarios, run only over the weekend. Because of such slow feedback, they add tests to the overnight or weekend packs only once they're stable. When developers release parts of the functionality, they run tests on their machines if possible. Testers run tests that require specialist hardware and offer feedback to developers.

In the last week of the iteration, they close up any unfinished issues. The team ensures that all the tests are running in the appropriate automated pack; they chase stakeholders about open issues and fix them. On the last Monday of an iteration, they run final regression tests and hold a retrospective.

Because RainStor is a relatively small company, their vice president of product engineering is responsible for analysis, along with many other things. He's not always available to attend the specification workshops, so the testers sometimes step in to help and take over some of his analysis tasks. Testers are responsible for gathering a list of questions and getting clarification before writing specifications with examples.

Key lessons

Although the development team size more than tripled over the last three years, RainStor still has a relatively small team. The same people have to develop a product, support the existing customers, and help grow the customer base. They have to be effective to do so with such a small number of people. Here's what they accomplished and how:

- Implementing executable specifications eliminated the need to maintain two sets of documents. It helped align testing and development and eliminate a lot of wasteful rework.
- Switching to user stories helped them engage their business users better.
- Deriving scope from business goals with high-level examples ensured that they build the right product and not waste time developing unnecessary features.
- Engaging customers into collaboration on specifications helped them make the process even more effective, because they get acceptance criteria from the very start, which ensures that they achieve the targets.

They gradually improved over the course of three years without any big plans or enforcing any particular process. Similar to many other stories, they always looked for the next big thing to improve, looked for ideas in the community to see which ones could help, and then figured out how to implement them in their particular context. This led them

to implement several unique practices such as running tests manually until they become stable and creating a living documentation from metadata with a custom-built tool.

Their particular context makes it unlikely that they'd be able to use any of the more popular tools to get the same effect, so they built a tool that helps them do the job efficiently. They started with the process and built a tool to support it.

For me, the key lesson here is to focus on the important principles when improving and to use popular community practices just as an inspiration.