Gojko Adzic

Specification by Example

How successful teams deliver the *right* software

MANNING

Table of Contents

Chapter 15. Sabre Airline Solutions. Specification by Example: How successful teams deliver the right software, ISBN: 9781617290084

15

Sabre Airline Solutions

Sabre Airline Solutions offers software and services to help airlines plan, operate, and sell their products. They are an early adopter of Extreme Programming and Specification by Example and an interesting case study because they applied SBE on a massive project, with a relatively large distributed team.

The project was Sabre AirCentre Movement Manager, a software system that monitors airline operations and alerts the relevant teams when it finds issues, allowing them to adjust schedules to minimize the impact to customers and the airline. According to Wes Williams, an agile coach at Sabre, two previous projects to build similar systems failed because of the domain complexity and quality issues. Specification by Example enabled them to complete this project successfully.

Changing the process

Because of the complexity of the domain, the teams at Sabre were looking for a collaborative way to specify and automate acceptance testing soon after implementing Extreme Programming. Williams said that they initially tried to do it with a technical unit-testing tool. That approach didn't help with collaboration and it wasn't reusable, so they abandoned it.

They started looking for a tool to drive collaboration. In 2003, Williams found FIT, the first widely available tool for automating executable specifications. His team started implementing acceptance testing with FIT, but they focused on the tool, not the practices. Williams says that it didn't give them the improvement in collaboration they expected:

> We liked the idea that a customer could define the test and drive the value you deliver in an application. In reality, we never got a client to write FIT tests in HTML. The tests got written most of the time by a developer. We had a hard time getting the customers to do it. Testers were using QTP. That never drove collaboration, and developers never ran QTP tests or got involved in writing them.

The developers were the only ones who wrote executable specifications, and they understood that didn't give them the benefits they expected. To improve the communication and collaboration, everyone had to be involved. A single group of people was unable to do that on their own.

A senior vice president of product development, influenced by the development team, brought in consultants from ObjectMentor to train everyone. They made the wider group aware of the goals of Specification by Example and the benefits they could get out of it. Although they didn't get everyone on board immediately, the training helped them get more people enthusiastic about the practices. Williams says:

> Not everyone adopted it. Still, there was a core group of people who believed in it, and they learned a lot. Those who didn't continued to fight against it.

That core group of people started with a relatively simple web project—an internal system for aggregating software build information. They wanted to try out the practices and get their heads around the tools, which were a lot worse in 2004 than they are now. The team selected FitNesse to manage executable specifications collaboratively. They wrote executable specifications either just before the development or roughly at the same time. The business stakeholder for the project was an internal manager, who got involved in reviewing the tests. The team initially looked at the automation layer as second-grade test code and cared little about making it clean, which caused numerous maintenance problems. They also ended up with a lot of duplication in test specifications. Williams says:

> We learned that we should try to keep fixtures as simple as possible and that duplication is bad. It [automation layer] is code, like any other code.

Developers didn't care much about making the automation layer or executable specifications maintainable because they just associated them with testing. By the end of the project, they realized that this approach led to huge maintenance problems. Similar to

the team at Iowa Student Loan, the first project allowed the Sabre Airline team to learn how to use a tool and see the effects and limitations of the way they automated executable specifications. This gave them ideas about how to improve the next project.

After the smaller team better understood the limitations of tools and realized why they should invest more in writing maintainable specifications with examples, they started to roll out the process to a large and risky project. This was a rewrite of a C++ legacy system to Java, with lots of deliveries. The project was data driven and had to support global distribution. At the end, it took 30 people two years to deliver the whole thing. They were split in three teams on two continents.

Because of the risk, they wanted to significantly improve the coverage and frequency of testing. This led them to start using the practices implemented on the smaller project. Williams says:

> Proper manual testing of a large application like this would take months. We wanted to prevent defects and not have to spend months testing. We did continuous testing. You can't even do manual sanity testing daily on applications this big.

Because they now had in-house experience with FitNesse, the people working on the previous project started to automate functional tests. They involved the business users in specifying the tests, expecting that this would ensure that their targets were met.

Improving collaboration

The group was split into three teams. The first team was working on the core features, the second on the user interface, and the third on integrations with external systems. It took about four months for the first version of the user interface to be delivered. Once the business users started to look at it, the core features team noticed that their software missed many customer expectations. Williams explained:

> The customer thought completely differently about the application when they saw the user interface. When we started writing acceptance tests for the UI, they had much more in them than the ones written for the domain. So the domain code had to be changed. But the customer assumed that that part was done. They had their FitNesse test there, they drove it, and it was passing. People assumed that the back end would handle everything that the UI mockup screens had on them. Sometimes the back end didn't support queries or data retrieval in a form that was usable to the front end.

They realized the problem was in the division of work between the teams. The customers naturally thought about the system at a more detailed level once they could see something visually, so they couldn't engage properly in defining the specifications for the work of the teams that didn't deliver any user interfaces.

About six months after the project started, the group decided to reorganize the work so that teams deliver end-to-end features. This allowed the business users to engage with all the teams. Williams added:

> Once we divided in the feature groups, we were in such a mature state on our user stories and the core of application that we didn't have story explosions. The surprises that came up were much lower.

When each team worked to deliver a whole feature end to end, it was much easier for business users to collaborate with the team to specify the conditions of satisfaction and engage in illustrating them with examples.

After the group reorganized the work, the teams realized that they need faster feedback on implemented stories, so they halved the length of an iteration to one week. Although they were writing acceptance tests before implementation, they still considered them tests, not specifications. Testers were charged with writing acceptance tests, but they couldn't keep up with such short iterations. To help remove this bottleneck, the group who implemented FitNesse on a previous project suggested that developers should help write acceptance tests. Williams says that the testers were initially reluctant to allow that:

> It was a struggle at the beginning to say that it's OK for a developer to write a test, because testers thought that they did such a better job of testing. I think they come from a completely different perspective. Actually, I've found since then that when a developer and a tester talk about the test together, it comes out significantly better than if one of them does it on their own.

Williams realized that this required a change of culture. As a coach, he tried to bring people together and let them expose the problems. When a tester got behind on testing, he would bring in a developer to help. When testers complained that developers didn't know how to write tests, he suggested pairing and writing tests in a group.

> They both went away and came back surprised with, "Wow—what I would have written on my own was nothing like what came out of this!" You need to get them through this experience.

Williams was surprised by how much trust was built between the testers and the developers as a result of that:

> The trust was amazing. They realized that they do a better job together, that they are on the same page, and that the other person is not trying to make things bad for them. At the end, you have a much more collaborative environment.

Getting people to work together not only helped them address bottlenecks in the process but also resulted in better specifications, because different people were approaching the same problem from different aspects. Collaboration helped both groups share knowledge and build trust in the other group gradually, which made the process much more efficient long term.

The result

Although the previous two attempts to rewrite the legacy system failed because of quality problems, this project went live initially with a very big customer and had very few issues. They discovered only one critical issue, which was related to failover. Williams said that Specification by Example was "one of the key pieces" for the success.

Key practices for data-driven projects

Wes Williams shared his top five tips for writing good specifications in a data-driven environment:

- Hide incidental data.
- Remove the duplication.
- Look for the duplication when you do incremental development—look at the old similar tests and clean up.
- Refactor tests similarly to the code.
- Isolate yourself and don't depend on third parties where you can't control the data. In the airline world, the system is going to talk to some host system at the end. They might have a test system as well, but you can't control the data. You need to have tests that talk to them, but these are completely separate tests. During the automated acceptance testing, this is what you want to mock.

Key lessons

Developers were driving the adoption of SBE as a way to reach out to testers and business users, but they quickly found out that focusing on a tool within a closed group wouldn't succeed. It was crucial to get everyone engaged. Although the training didn't get everyone on board, it gave them a common baseline, and it identified a core group of people who were genuinely interested in trying out the new ideas.

They used a smaller and less risky project to get their heads around the tools and discover good ways to write and maintain the specifications and the automation layer. A small group of people involved in that project acted as a catalyst for the larger group on the big project.

While the teams were delivering components of the system, the business users couldn't engage properly with the teams working on background components, which caused a lot of rework and missed expectations. Once they restructured into feature teams, the problem went away.

Getting testers and developers to collaborate on writing acceptance tests produced much better specifications and helped to build trust between those two groups.

Specification by Example helped them conquer a complex domain by providing a clear target for development and continuous validation.