Gojko Adzic

Specification by Example

How successful teams deliver the *right* software

MANNING

Table of Contents

Chapter 17. Songkick..... 1
Changing the process..... 2
Current process..... 5
Key lessons..... 6

Chapter 17. Songkick. Specification by Example: How successful teams deliver the right software, ISBN: 9781617290084

17
Songkick

Songkick is a UK-based startup that operates Songkick.com, a consumer website about live music. They are an interesting case study for two reasons. Unlike all the other companies featured in this book, they implemented Specification by Example while still at a startup phase, without having to deal with a large legacy system in the background. Also unlike most other projects covered in this book, user interaction is one of the most critical aspects of their product, and new features are developed with a strong emphasis on the user experience, based on observation of how users interact with the site.

The complexity of the system mostly comes from the number of user experience subtleties and the number of features they build to give users a rich experience. Songkick implemented Specification by Example to focus on delivering software that matters and to be able to grow their development team.

"As a startup you can't afford to not be delivering value all the time," says their CTO Phil Cowans. Here's what he pointed to as the biggest benefits of SBE:

> "Getting to what we actually intended to build quicker because we use the same language in the tests as we do when we decide what to build and go through the process of understanding our customers. That helps reduce communication issues. We aspire to never having a situation where developers turn around and say: What we built works; you just didn't ask for the right thing."

For a startup, delivering software that adds real value efficiently is much more important than it is at more mature companies. The practices of Specification by Example help Songkick get more value for their investment in software development.

Changing the process

Their project started two and a half years ago. After the first year, the team started growing "beyond the size where everyone can sit around a table and develop the code together," according to Cowans. To deal with a code base that's getting more complex and a growing team, they decided to implement test-driven development. He says:

> Before moving to TDD we relied on trusting that everything we'd previously built still worked when we released new code. But very soon it became clear we'd need more confidence that when we'd finished something, it did what we thought it would and didn't cause regressions. It seemed pretty obvious that we were going to start slowing down if we didn't find some way to avoid constantly tripping over each other's toes and find a better way of communicating requirements and avoiding regression.

Within three months, they felt that TDD was a natural way to do things. At the same time, they started looking into Kanban and the ideas of user stories, which led them to start applying TDD principles to business functionality and effectively start implementing SBE. As a team, they like to experiment with different ways of working, so they just tried it out without much fuss. Cowans explains:

> We were in touch with the guy—who's now a full-time employee but who was an adviser at that point—who had used Kanban on earlier projects. Through him we had some evidence that working in terms of user stories could be successful, and it seemed to make sense. The decision to make that part of our process was really just a matter of "there are some people using this technology to do this; let's just try it and see how it works." It became something that we found very natural to do.

The team started to derive scope from goals to drive the user stories from the business value. They also used Cucumber to create executable specifications. Cowans says that the focus of the process shifted from unit tests to business specifications as they got better with Cucumber:

> Initially we set out to use a mix of Rails test framework and Cucumber. We used Cucumber to drive out the high-level user stories and unit tests to specify the detailed behavior. Over time, we were using Cucumber more and more and found ways to specify things more in Cucumber.

The new way of specifying helped the team focus on building software that really matters. Cowans says:

> It helps people stay focused on why we're doing something and see the value of what we're doing. It helps people avoid wasting time building things we don't need. Everyone approaches the problem from the same direction, which ensures that the development team and the rest of the company think about the problem in the same way.

Similar to the team at ePlan Services, Songkick first implemented (unit) test-driven development and then extended that to business features. They didn't actually hit quality problems that made them change the process, but they did that proactively to be more efficient.

Cowans says that the key challenges for his team, when implementing Specification by Example, were understanding what to test, how to avoid making the executable specifications brittle, and how to make continuous validation faster.

Once the team got comfortable with how to use the tools, they went too far with focusing on the user interface functionality, because that was easy to think about. Cowans says:

> We spent too long testing trivial bits of the user interface, because that was easy to do. We didn't spend enough time digging into edge cases and alternative paths through the application.

They also automated executable specifications in a way that was closely tied to the user interface, so the test results were brittle. Cowans says that in some cases this pushed them to testing after development instead of using tests as specifications:

> Someone changing some punctuation on the web page and breaking the test is not good. It's frustrating because it makes it difficult to predict the effects of a change. So it's difficult to update the entire test suite in advance. As a result, in some cases people would code first and test later.

To address these problems, the team started to make tests more semantic and pushed the translation between the domain language and user interface concepts into the automation layer, Cowans explained.

> You develop a familiarity with the process and start understanding where dependence on the UI is likely to cause a problem in the long term. Developing more domain-focused step definitions [in the automation layer] helped to get around that, giving us higher-level ways to work with the markup.

The changes they implemented were effectively the start of refining the specification and looking for ways to express the specifications in the business language, not in the language of user interface terms.

According to Cowans, it took about six months for the team to get comfortable with the process and the tools they use for Specification by Example:

> It's probably in the last nine to six months that it has felt like this is a part of what we do. Within the last nine months no one has really questioned how we specify work; it's just there in the background.

The team realized exactly how important their executable specifications are when they had to rewrite a part of the system dealing with activity feeds. An existing set of business-focused specifications that were automated as tests gave them the confidence that they didn't introduce bugs or reduce functionality while rewriting the feeds. Cowans says:

> Everyone on the team became aware at that point that this can save us a lot of time. My gut feeling is that we saved 50% of the time doing the refactoring because of the tests.

For a startup, saving 50% of the time on a task means a lot. Their set of executable specifications effectively protects the system from regression issues. According to Cowans, they have so few issues in production that they don't need a bug-tracking system. This allows them to focus on delivering new features instead of maintaining the system.

No living documentation yet

At the time I interviewed Cowans, the number of executable specifications in their system had grown enough for them to start thinking about reorganizing the specifications, essentially starting a living documentation system. Cowans says:

> When setting this up we didn't think enough about the high-level structure of the tests. As the application evolved, we ended up just adding new tests as needed in an ad hoc way. As a result, when modify-

> ing existing code it's hard to find which tests cover which functionality. Deciding on a high-level description of the feature set of the site and organizing the test suite along those lines, rather simply adding new tests for each new feature we built, would have helped. I think that's also useful when it comes to developing a product and maintaining a code base that's relatively easy to understand. You end up with a shared language to describe how things fit in with what's already there.

Current process

Songkick's development process is based on Kanban flow. They have a product team responsible for the roadmap and a development team responsible for implementation. The product team consists of the head of product development, a creative director, and an interaction designer. The development team has nine developers and two testers. In the development team, two people focus more on the client side and user interfaces, and the rest are more focused on middleware and backend. Cowans, who is the CTO, is also part of the development team. According to him, the company tries to build in as much collaboration between product and development as possible, so the boundaries between the teams are fairly blurred.

Once a feature is of sufficiently high priority that it's likely to be built, the product team meets to investigate the user experience and the technology required to implement it. The outputs of this meeting are wire frames, notes about specific cases, and a first guess at the list of user stories for the feature.

When the development team has enough capacity to start implementing the feature, they organize an initial meeting with the product team and any developers or testers likely to work on the feature. At this meeting they break down the feature into user stories and jointly brainstorm the acceptance criteria for each story. The acceptance criteria are defined as a set of things to check, with detailed examples to be filled in later.

The testers own the requirements, including the user stories and the associated list of acceptance criteria. They are responsible for maintaining that as additional information comes through during development. Because of the importance of usability and user interaction, they manually test the core functionality of every feature after development, in addition to running all the executable specifications. So testers start thinking about a test plan after the initial meeting.

The developers write specifications with examples, and the testers review them, advising on what else should be covered. The developers then automate them, implement the required functionality with TDD and make that branch of code available to the testers.

The testers then run their manual tests, start doing exploratory testing, and provide feedback to the developers. Once the testers and the developers agree that a feature is ready, it goes into a queue for integration.

Features from the queue are integrated into the master branch. The entire continuous validation suite is then executed, the code is deployed to a staging environment, and the testers run final core functionality manual tests. After that, the code goes live to a production environment.

Ah-ha moments

I asked Cowans about the key ah-ha moments for him from related to their implementation of Specification by Example. He says:

- It's quite easy to test what you can see, but ultimately you need to have a deep understanding of what the software does rather than what the user interface looks like. Thinking in terms of user stories and paths through the application really helps.
- Treat your test suite as a first-class citizen. It needs careful maintenance as much as the application code itself.
- The tests are the canonical description of what the application does. Ultimately, success is as much about building the right thing as building it well. If the tests are your description of what the code does, they're not just an important part of your development process but an important part of the wider process of building the product. They can help you understand what you've built and keep complexity under control.
- It's important to have everyone in the process involved; it's not just something that developers do. Ultimately it [Specification by Example] gives you tests written by the developers that the product owner can read. You should make good use of that.

Key lessons

The key lesson from Songkick for me was that if you don't have a massive legacy system to slow you down, it's possible to go from TDD to Specification by Example quickly. At Songkick they just approached it an extension of the TDD process to cover business functionality.

The team builds and maintains a web system, so they initially automated tests in a way that was too closely tied to the user interface. This caused lots of maintenance problems and led them to start refining the specifications and automating user interface checks at a higher level of abstraction.

It took them about one year to start thinking about a living documentation and seeing how important that can be when parts of the system are rewritten.

As a startup, Songkick benefits greatly from focusing on delivering the things that really matter. A shared understanding from collaborating on specifications ensures that they all focus on delivering the right product. The second most important benefit for them comes from executable specifications, because they discover problems early and can focus on rolling out new functionality rather than wasting time troubleshooting and fixing bugs.

This page intentionally left blank