September 2014

testing experience

The Magazine for Professional Testers

NEW: New Columns & Book Corner

printed in Germany
print version 8,00 €
free digital version
www.testingexperience.com
ISSN 1866-5705

By Felix Krüger

Field Report:
Test Automation and Quality Assurance in the Context of Multi-Platform Mobile Development

The word "app" still suggests that we are dealing with "little applications". While that may be true in some cases, this field report is about a pretty big app that is used to remote control and monitor the statuses of different parts of a machine, such as light, air flow, and position. The machine uses a mobile communications network to be accessible by a backend server, which our app accesses via the internet. In all, the complexity is comparable to a desktop application.

One major aspect of this app is variant management. Different customer groups receive different feature sets and different machine types require specific data presentation. This results in a very dynamic app – both in terms of its composition during the build and also at runtime, depending on which machine type we want to use.

So this is anything but a "small" project. It is not a mobile app that accompanies an existing business application – it is the only solution for this business process. There will be a longer maintenance phase with enhancements, new features, and far more variants to support. The app is an intrinsic part of the machine and has to fulfill the same high standards concerning quality, usability, and user experience.

The report provides an overview of the project, as well as our decisions and experiences regarding quality assurance and test automation, continuous integration, and project management.

Project setup: one concept, two apps, two teams

The project uses the SCRUM agile software development framework. A sprint takes two weeks, including a total of about one day for sprint review, retrospective, and planning. Sprint reviews are held with the whole development team, one or two customer representatives, and sometimes additional stakeholders from the Quality Assurance or Infrastructure departments. The review often results in specification refinement by the customer. During the first part of sprint planning – the user story selection – a customer representative also participates. This allows the developers to ask questions about specification details and sometimes to propose specification changes to allow more "native" behavior of the app.

The main development phase is planned to take about nine months. During this time, the team size will vary between 8 and 13 persons including the Product Owner and the Scrum Master. The fluctuation is partly due to students who are with the project for a set amount of time and partly due to experts for special tasks joining the team temporarily.

Our target devices are iPhones and Android phones, specifically 3GS and above – (iOS 6+) for the iPhone, and Version 4 and above for Android. The machine to be controlled and the server backend already exist, so our sole task is to develop the app, including the user interface, backend communication, variant management, and integration of platform-specific services (e.g. for push notifications, maps, or social networks).

To assure the best possible user experience, we are not using a cross-platform toolkit; instead, two independent native apps are being developed. The developers are split into sub-teams with experts for their respective platform. To promote communication, both teams work together on a single site.

Because of the two teams, the product backlog contains most user stories twice – one version for each supported platform. For most user stories, both versions are planned in the same sprint, depending on the development progress, which does differ for iOS and Android. When a story is implemented, the result is compared with the other platform's app. During the sprint review, we prefer to present a feature in parallel for Android and iOS. By doing this, we can ensure that we achieve feature-identical apps and a very similar user experience for both target platforms.

Beyond the user experience, the Android and iOS apps have very similar software architecture, despite being implemented independently. The data model, layered design, screen flow management, variant management, and domain-specific algorithms are specified in a common software architecture document. So, when implementing a function for the second platform there is a template which is easy to understand because it is implemented on the same basis. This does not work for view implementation due to different widgets and user interaction concepts – here the development is completely platform-specific.

Figure 1. Communication setup from mobile device to machine

Automated testing

The challenge for our quality assurance in terms of testing is to have tests for each app, at multiple levels (unit, integration, and acceptance (UI tests). Since we want to automate as much as possible, we have a QA consultant who is a part of the team and who drives our test automation. He is responsible for test specification and review of test implementations. The actual implementation of automated tests is done by the developers, triggered by a test task that is generated by default for each user story.

Depending on the implemented feature, there are acceptance tests (automated UI tests), unit tests, and integration tests. Tests are always implemented platform-specifically. In the case of the UI tests, they are synchronized by using the same acceptance criteria. For each acceptance criterion defined in a (UI-related) user story there is at least one UI test.

To implement all these different automated tests, we use platform-specific frameworks. Our lower-level tests are implemented using SenTest or JUnit respectively. On iOS, additional libraries like nocilla and JRSwizzle are used for mocking. For UI tests we use KIF for iOS and Robotium for Android. In order to get more stable Android tests and eliminate "false negative" results, the Robotium Recorder (commercial product) has proven useful. Even though we place great importance on the apps being feature-identical, the differences in navigation and user experience between iOS and Android mean that the steps required to access and use each feature are different. Unlike in the desktop world, it is only theoretically possible to use one UI test to cover cross-platform apps that have gone beyond simple concepts. This has the disadvantage of increasing the technology stack and effort, but has the advantage of being able to use specific tools for specific problems.

In terms of ratio, it is often said that UI tests should be the smallest "chunk" of the testing pie. This is partly due to their execution time, but also because they are still often seen as the hardest to write and maintain. Our experience is that it can be worth re-evaluating the ratios of test levels specific to each project. With an increased focus on customer acceptance (both in agile projects and in the mobile domain) and the improved suitability of UI testing tools, testing through the UI and of the GUI logic itself should not be neglected; indeed, some projects may find that the pie is more heavily weighted towards UI tests than unit tests.

For this project, we have around 10 % unit tests, 40 % integration tests (mocked and against the real backend), and 50 % UI tests. The amount of integration tests is only that high because of quality issues (poor interface specification) in the product we received from the independent backend supplier.

Continuous integration

We use Git with a basic branching model as our version control system. It defines a master branch, release branches, and branches for each feature and bug fix. The developer merges a feature as a whole into the master branch when the user story is "done". To ensure that no incomplete features are merged into the master branch, default tasks are generated for each user story. There are default tasks for acceptance (reviews by product owner and QA consultant) and code quality (code review, static code analysis, and automated tests).

The basis for continuous integration is the master branch, because it should always contain a project "ready to release". Each commit (merged feature) triggers a full build cycle consisting of the following jobs:

- Updating/building dependencies
- Building the app (currently three different build variants)
- Static analysis
- Unit tests
- UI tests (one job for each build time variant)
- App distribution via intranet

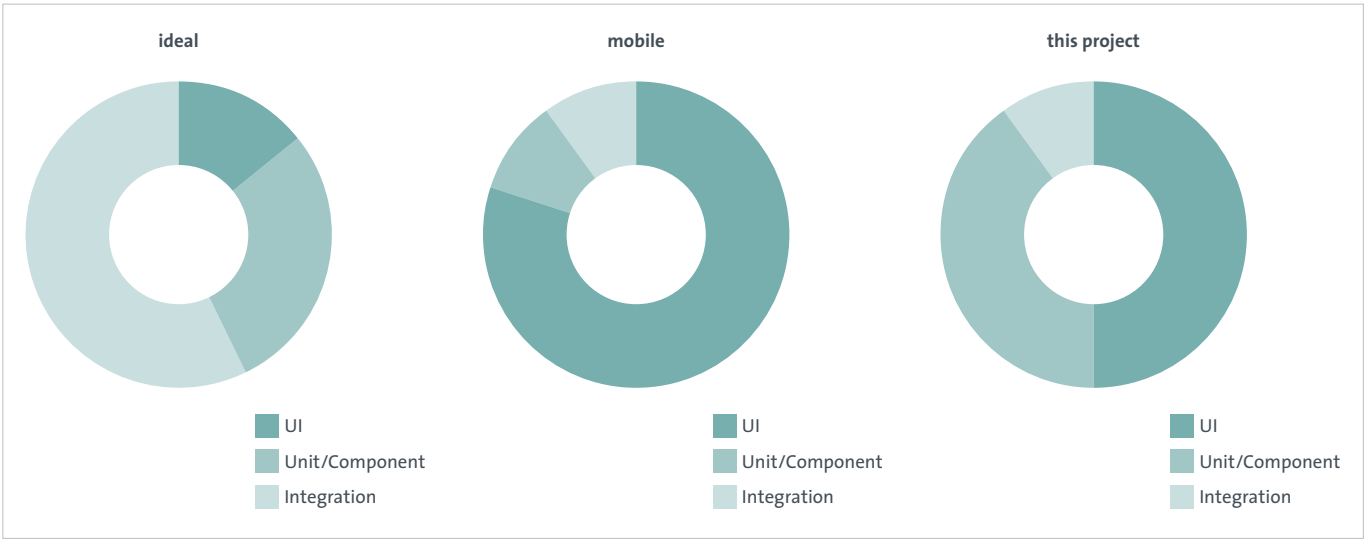

Figure 2. Test level ratio (ideal, typical mobile development project, this project)

We use Jenkins for iOS and Android, as it is the company default and well supported by the IT department. Especially in the iOS development, we had teething problems with Jenkins that could have been avoided had we been able to choose the Xcode server. However, additional plugins in Jenkins did eventually make it possible to integrate our iOS systems into the CI, for example the Clang Analyzer, and plugins to manage environment variables or share workspaces between jobs.

Final thoughts: where automation cannot help

As described, our process contains various factors to help us fulfill our customer's high quality expectations. The whole team is involved in the quality process, and quality accompanies the project in each sprint. This is considerably helped by our automated tests.

However, we have identified areas where quality assurance must be done manually. Since these are not areas that teams used to desktop development would instantly think of, they are listed below:

- **Usability and user experience:** This is an area where it is widely accepted that manual testing is required, but customers of mobile apps weight this quality attribute much more heavily. With the added complexity of gestures and orientation changes, we find that we have to place more focus on this area of quality – and the tests are performed manually by the team as well as by customer representatives. In this setup, it is the customer who ensures that different devices are tested as part of their manual acceptance test. Our automated tests are restricted to one version of each platform.
- **Internationalization:** The normal process in multi-lingual desktop applications is to have strings checked by native speakers, outside of the context of the application. Due to the decreased size of the screen, our planned internationalization into over 15 languages is more time-consuming than it would be for a desktop application. Our translation is performed by external employees, and each translation must be manually checked to ensure that it uses the correct amount of space on the screen. We use our UI tests to support this by creating automated screenshots that can be reviewed by the translation team.

Quality counts – even (especially) for apps

This field report shows that app development requires a test strategy and quality assurance activities to at least the same degree as a desktop business application. In some respects, the requirement for good quality practices is even higher due to the challenges of developing one app for two platforms.

In many ways, we can see that mobile development does not change the quality activities required. What can change, though, is the relevance or importance of specific activities. For us, this was clear in the distribution of our unit, integration, and acceptance tests, as well as in the areas of manual testing that would have been less important or time-consuming in a non-mobile project. Our conclusion? Quality counts – and it is important to know what you want to test, why, and how. Even for mobile applications. ■

> about the author

Felix Krüger works as a software engineer at BREDEX GmbH (*www.bredex.de*) in Brunswick, Germany. He studied computer science at the Brandenburg University of Technology in Cottbus. He is involved in the development of various mobile and desktop clients for enterprise software systems.

Over the past eight years he has worked in different fields of automotive software development. He is experienced in handling (Java and Eclipse-based) client-server systems with desktop clients or mobile clients, as well as test automation systems and embedded software.