Qualitätsmanagement
Projekt BierIdee

Danilo Bargen, Christian Fässler, Jonas Furrer

31. Mai 2012

Inhaltsverzeichnis

Änderungshistorie

Version	Datum	Änderung	Person
v1.0	30.05.2012	Dokument erstellt	dbargen

1 Einführung

1.1 Zweck

Dieses Dokument beschreibt die Qualitätssicherungsmassnahmen des Projektes BierIdee.

1.2 Gültigkeitsbereich

Die Gültigkeit des Dokumentes beschränkt sich auf die Dauer des SE2-Projekte Modules FS2012.

1.3 Referenzen

- Definition.of.Done.pdf
- usertest.baumann.pdf
- usertest.tanner.pdf

2 Definition of Done

Als primäre Qualitätssicherungsmassnahme wurde eine Definition of Done[1] erstellt. Diese enthält folgende Qualitätsrelevanten Punkte:

2.1 Code Reviews

Um die Qualität unserer Codebasis zu gewährleisten, wurden regelmässig Code Reviews durchgeführt. Gemäss Steve McConnell [McC05] ist dies eine äusserst effektive Qualitätsmassnahme:

> ... software testing alone has limited effectiveness – the average defect detection rate is only 25 percent for unit testing, 35 percent for function testing, and 45 percent for integration testing. In contrast, the average effectiveness of design and code inspections are 55 and 60 percent. Case studies of review results have been impressive ...

Gemäss der Definition of Done darf kein nichttrivialer Code in den `main` Branch gemerged werden, ohne dass er reviewed wurde.

[1] Siehe *Definition.of.Done.pdf*

2.2 Code-Kommentare

Um die Codebasis übersichtlich und gut verständlich zu halten, sollten überall wo sinnvoll Code-Kommentare eingefügt werden. Klassen und Methoden müssen mit Javadoc kommentiert werden. Daraus wird nach Abschluss des Projektes eine Javadoc-HTML-Hilfe generiert.

2.3 Checkstyle

Wir haben während dem Projekt Checkstyle[2] eingesetzt und dafür unsere eigene Konfiguration erstellt. Um die Definition of Done zu erfüllen, dürfen im Code mithilfe unserer Checkstyle-Konfiguration keine Warnungen mehr vorhanden sein.

2.4 Unit Tests

Unit Tests müssen vorhanden sein, um die neu erstellte Funktionalität zu testen. Im Backend wurde dafür JUnit 4 eingesetzt, im Frontend JUnit 3 mit Robotium[3].

2.5 Continuous Integration

Damit die Codebasis stets kompilierbar bleibt, wurde ein Jenkins Buildserver eingesetzt. Dieser wurde mit diversen Plugins erweitert (GitHub Plugin, Maven Plugin, Android Plugin, etc. . .). Neben dem Buildvorgang wurden auch die Unit- und Integrationtests ausgeführt.

Zu Beginn des Projektes wurde vereinbart, dass ein durch Unachtsamkeit verschuldeter Build Fail bestraft wird – die entsprechende Person muss am nächsten Tag Gipfeli für das Team mitbringen.

Die Continuous Integration verlief leider nicht so reibungslos wie erwartet. Durch Probleme mit dem Caching der Codebasis (das Repository wurde nicht jedesmal neu ausgecheckt sondern nur updated) war der Build-Status auf dem Integrationsserver häufig auf „Fehlerhaft“, obwohl der Build wie auch alle Tests eigentlich erfolgreich durchführbar waren. Diese Probleme wurden jeweils durch ein manuelles „cleaning“ des Workspace gelöst.

3 Issue Tracking

Um den Überblick über die geplanten Tasks zu behalten und um sicherzustellen, dass keine Features vergessen gehen, wurde während dem Projekt konsequent mit

[2] `http://checkstyle.sourceforge.net/`
[3] `http://code.google.com/p/robotium/`

Redmine gearbeitet. Alle Aufgaben wurden in Tasks unterteilt und den jeweiligen Personen zugeteilt. Zeitschätzungen wurden eingetragen und Kategorien gesetzt.

4 Usertests

Um auch Feedback von Benutzern zu erhalten und so auf Probleme aufmerksam zu werden die man als Entwickler häufig übersieht, haben wir ein Testprotokoll erstellt und die App von Freiwilligen testen lassen. Die Erkenntnisse flossen jeweils direkt in den Entwicklungszyklus ein.

Zwei Beispiele sind in in den PDF-Dateien `usertest.baumann.pdf` und `usertest.-tanner.pdf` beigefügt.

5 Protokolle

Für jedes Meeting wurde jeweils ein Protokoll erstellt. Zu Beginn haben wir dafür die Website http://minutes.io/ verwendet, später sind wir wegen Bugs in ihrem System davon weggekommen und haben die Protokolle in Textdateien erstellt.

Literatur

[McC05] S. McConnell. *Code Complete*. Microsoft Academic Program : Softwareentwicklung. Microsoft Press, 2005.