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 Bier-Idee.

1.2 Gültigkeitsbereich

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

1.3 Referenzen

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

2 Definition of Done

Als primäre Qualitätssicherungsmassnahme wurde eine Definition of Done[1] erstellt. Diese enthält die im folgenden erwähnten Qualitätsrelevanten Punkte.

2.1 Code Reviews

Um die Qualität der 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 `master` Branch gemerged werden, ohne dass dieser reviewed wurde.

2.1.1 Pull Requests

Um die Code Reviews durchzuführen, wurde ein Feature von Github[2] eingesetzt, welches sich „Pull Requests„ nennt. Der Workflow funktioniert folgendermassen:

1. Ein Teammitglied will ein neues Feature entwickeln. Er erstellt mit Git einen neuen Branch und arbeitet darin.
2. Wenn die Entwicklung des Features fertig ist oder er Feedback braucht, wird der Branch auf Github pushed.
3. Auf Github erstellt das Teammitglied eine Anfrage, um den Branch in den `master` Branch zu megen (der sogenannte „Pull Request“). Der Pull Request ist ein Diskussionsthread, in welchem man den ganzen Branch, einzelne Commits oder einzelne Stellen im Code kommentieren kann.
4. Wenn der Pull Request noch nicht merge-bereit ist, werden von einem Teammitglied im Branch weitere Commits erstellt. Sobald sie auf Github pushed werden, erscheinen sie auch im Pull Request.
5. Wenn alle mit dem Code zufrieden sind, wird er mit dem `master` Branch gemerged. Falls der Merge konfliktfrei möglich ist, kann dies direkt über das Webinterface von Github gemacht werden.

[1] Siehe *Definition.of.Done.pdf*
[2] `http://github.com/`

Mit diesem Workflow konnten wir Frontend, Backend wie auch die LaTeX-Dokumente reviewen, überarbeiten und laufend verbessern. Ein grosser Vorteil daran ist vor allem, dass man direkt im Code-Diff Diskussionen erstellen kann.

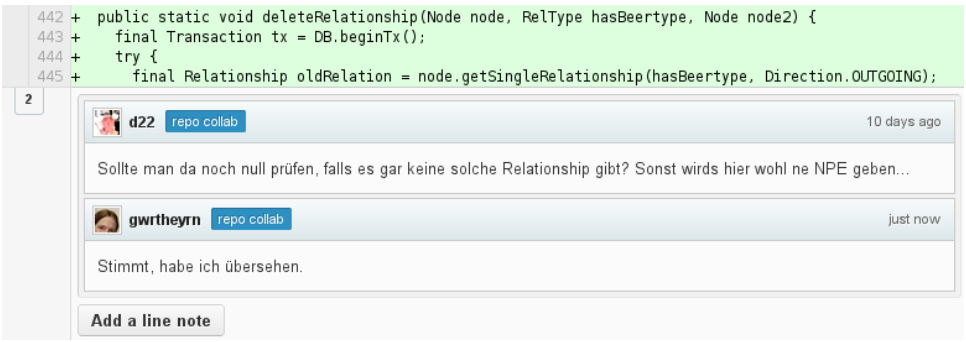

Abbildung 1: Diskussion im Diff

Die Github-Gruppe für unser Projekt kann online[3] eingesehen werden.

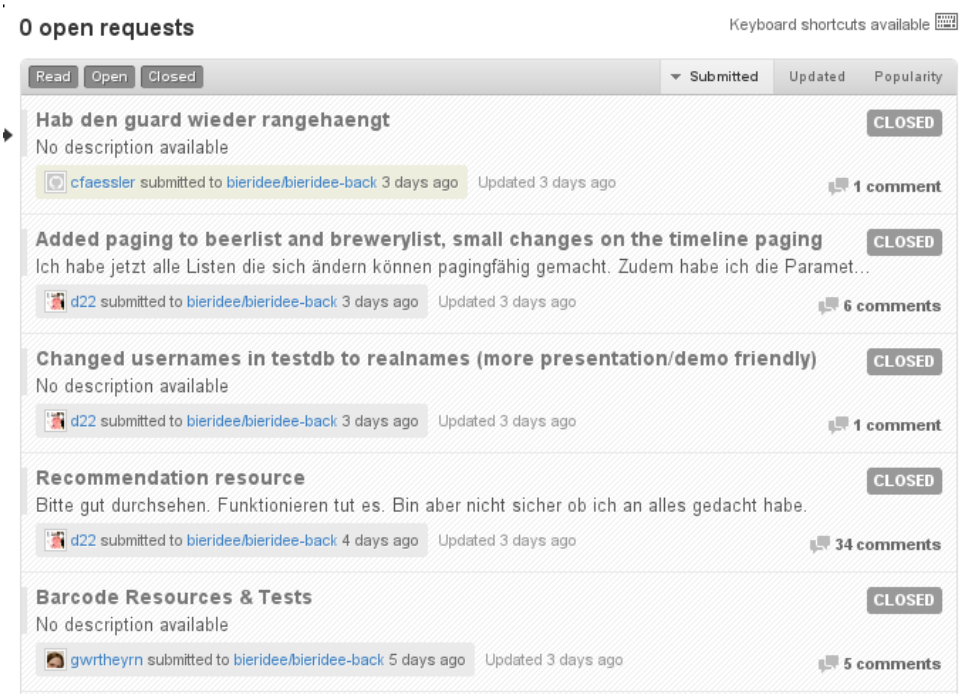

Abbildung 2: Backend Pull Requests (Auszug)

Ein erweitertes Beispiel einer Pull Request Diskussion findet sich in der Datei `PullRequestExample.jpg`.

[3] `https://github.com/bieridee`

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

Während des Projektes wurde Checkstyle[4] mit einer eigens für das Projekt erstellen Konfigureation eingesetzt. Um die Definition of Done zu erfüllen, dürfen im Code mithilfe dieser 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[5].

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. Um die Android-Integrationstests auszuführen wurde im Hintergrund jeweils eine VNC-Server-Instanz gestartet, die den Emulator gestartet und die Tests darin ausgeführt hat.

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

[4] `http://checkstyle.sourceforge.net/`
[5] `http://code.google.com/p/robotium/`

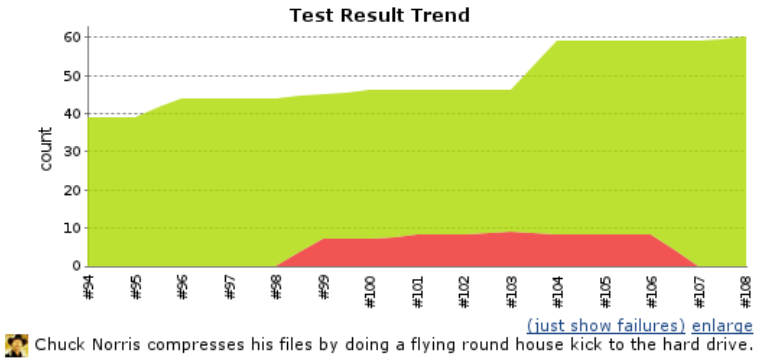

Abbildung 3: Test Result-Trend Backend

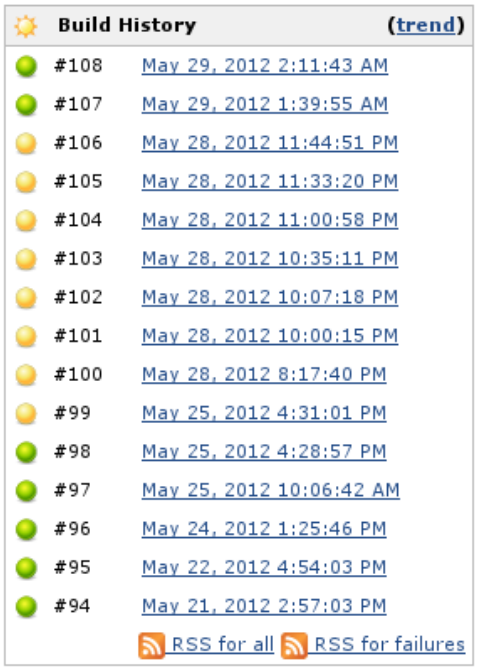

Abbildung 4: Build-History Backend

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 (vor Allem im Frontend-Projekt) 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 Redmine gearbeitet. Alle Aufgaben wurden in Tasks unterteilt und den jeweiligen Personen zugeteilt. Zeitschätzungen wurden eingetragen und Kategorien gesetzt.

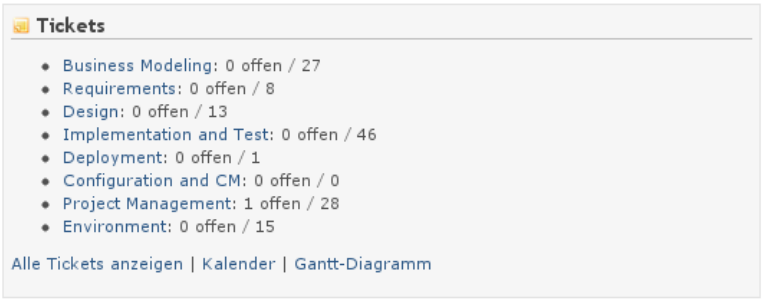

Abbildung 5: Redmine Ticket-Statistik

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 wurde wegen Bugs auf deren Website auf Textdateien umgestellt.

Literatur

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