Agile Qualitätssicherung

Holger Schmeisky, 17.10.2013

Agile Qualitätssicherung

- Studie
- SoundCloud
- Team Payment
- Entwicklungsprozess
- Qualitätssicherung
- Analyse / Diskussion

Studie

- Wie und unter welchen Bedingungen funktioniert agile Qualitätssicherung?
- Wenn sie gut funktioniert, warum funktioniert sie?
- Fallstudie in Teams bei 2 Firmen
- stelle hier ein Team vor

SoundCloud

- Eigene Musik hochladen und Musik hören
- Viele Feedbackmöglichkeiten, gut für Creators
- ca. 200 Mitarbeiter, >10 Mio. Nutzer
- 2h umsonst, 3€ / Monat 4h, 9€ / Monat unbegrenzt

SoundCloud

- 6 Jahre alt, angefangen als Start-Up
- Früher RoR-App, jetzt Refactoring in Verteilte Dienste
- Kultur immer noch Startupmäßig
- Viel Autonomie und Vertrauen
- Eat you own Dogfood
- Mitarbeiter identifizieren sich mit Firma

Team Payment - Produkt

- Bezahlungsabwicklung
- Bezahlmethoden, Währungen, Eingabemasken
- User Lifecycle

Team Payment - Software

- Eigener Service mit API für Bezahlen
- Kommuniziert mit HTTP und JSON
- Altsystemmigration fast abgeschlossen

Team Payment – Team

- Tobias, Entwickler (seit 2010 bei SC)
- Deepak, Entwickler (seit Frühjahr 2013)
- Moritz, Product Owner (seit Anfang 2012)
- 1 Großraumbüro, Team sitzt nebeneinander

Methodik

- Entwicklerbegleitung: Erklärungen und Beobachtungen
- Interviews mit Tobias und Moritz
- Dokumente
- Aus Daten Bericht geschrieben zum Verständnis

Entwicklungsprozess

- Planung
- Entwicklung
- Veröffentlichung
- Überwachung
- Tests

Prozess - Planung

- PO holt Anforderungen, Team diskutiert sie
- Epics und Sub-Epics, Team teilt in Stories auf
- Entwickler teilen erst auf, PO schaut drüber
- Stories wandern durch Onlinewand
- Alle Arbeit als Story (auch kleine Änderungen)
- Keine expliziten Akzeptanzkriterien

Prozess - Entwicklung

- Zuerst HTTP IN – HTTP Out Integrationstests
- Dagegen Entwickeln
- Codestandards:
  - "Avoid object inheritance in models. An inheritance can always be written as a composition, which is easier to test and is more flexible"
  - "A model should not call external entities like Postman. This makes testing very difficult"
- Github Pull Request, Merge nach Review

Prozess - Tests

- Tests sehr sehr wichtig. Testpyramide:
- Unittests: Nur Buckster, Rest gemockt
- Integrationstests: HTTP IN-JSON OUT, Mock auf HTTP
- Webtests: Selenium, laufen auf Staging vor Deployment
- Guter Test: Kurz in jeder Hinsicht
- Klare, konsistente Teststrukturierung: subject, let, before, should

Prozess - Veröffentlichung

- Entwickler können selber deployen
- SC hat Tool entwickelt: Bazooka, wie Heroku, private Cloud
- Makefile, Konventionen und Env-Variablen
- „rake deploy staging“
- Anschließend manuelle, explorative Abnahme durch PO
- „rake deploy production“
- Danach Livesystem überwachen, manueller Test

Prozess - Überwachung

- Sammeln Statistiken: Art und Anzahl HTTP Responses, Deployments, Anzahl Verkäufe, Anzahl (ruby) Fehler …
- Dazu Alarmprüfungen in New Relic
- Wenn Alarme fehlschlagen wird Entwickler benachrichtigt
- Wenn außerhalb der Bürozeiten Alarm, ist Entwickler auf Bereitschaft und kriegt SMS
- Bereitschaft hat sich entwickelt

Qualitätssicherung

- Qualitätsbegriff
- Automatisierte Tests
- Inkrementeller Rollout
- Manuelle Tests
- Produktvision
- Pull Requests
- Faulteindämming
- Verantwortung

QS - Qualitätsbegriff

- Im Vergleich zum Rest der Firma hohe Anforderung an Korrektheit
- Kultur ist aber weniger Fehler vermeiden, als Fehler schnell beheben
- Falls User Fehler sieht, Entschädigung mit freier Mitgliedschaft

QS – Tests & Testkultur

- Tests sind das, was Vertrauen ins Deployment schafft
- Testbarkeit von vornherein Designziel
- Viel Zeit darin investiert
- ¼ – 1/3 der Zeit für Test

QS – Inkrementeller Rollout

- Alle 2-3 Tage ein kleines Deployment
- Schnelles Feedback
- Leichter Fehler zu lokalisieren
- Wird extra so geplant: Must have, nicht must have

QS – Manuelle Abnahmen

- Stories, die der Nutzer sieht macht Moritz manuelle, explorative Abnahme auf Staging
- Hat dabei bisher nie schwerwiegende Fehler gefunden, nur mal kritische, die aber immer schnell geändert werden konnten
- Entwickler wollen sich explizit nicht darauf verlassen um Fehler zu finden

QS - Produktvision

- Frühe und starke Involvierung in Planung
- Brauchen nicht mal explizite Akzeptanzkriterien
- Produktvision also sehr deutlich
- Also wahrscheinlich nahe echten Anforderungen

QS – Pull Requests

- Review
- Wissenstransfer

QS - Faulteindämmung

- Monitoring, Fehleraggregation
- 5-Uhr-Regel: Letztes Deploy 2h vor Feierabend
- Community Support
- Bereitschaftsdienst

QS - Verantwortung

- Keine richtige Praktik, aber wiederkehrendes Thema:

  ``[...], echt das wichtigste ist, [...], ich möchte das aber betonen: Der Entwickler ist verantwortlich für das was er macht. Und zu keiner Zeit denkt er, er kann den Code schreiben und sich darauf verlassen, dass jemand anders den fixt." (Tobias)

Effektivität

- PO verantwortlich für Qualität
- Zufrieden vorallem, weil Fehler immer schnell behoben werden
- „Frage: Wie zufrieden bist du mit der Qualität, die die produzieren? --

  […] Ich glaube aber, da wir relativ wenig schlimme Bugs haben, dass wir einen ganz guten Job hier machen.``
- Entwickler ebenfalls sehr zufrieden „Habens im Griff“

Diskussion

- Was fällt auf?
- Was haltet ihr davon?

Gründe

- Warum glaubt ihr ist das so?
- Was mir auffällt:
  - Vertrauen
  - Identifikation & Bedeutung
  - Schnelles Feedback
  - Aufgaben von Anfang bis Ende
  - Tests technisch effektiv

JCM

- Arbeit hat 5 Kerndimensionen, wenn diese ausgeprägt sind, dann intrinsiche Motivation hoch
  - Aufgabenvielfalt/Anforderungswechsel:
  - Identität der Aufgabe
  - Aufgabensignifikanz/Wichtigkeit
  - Autonomie
  - Feedback/Rückmeldung
- Gilt nur für Leute mit hohem Wachstumsbedürfnis, Entwickler haben das meistens (Studie)