Code Review

Seminar Effiziente Programmierung

Jan Ole Wellnitz

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2017-12-07

Gliederung (Agenda)

Einleitung und Motivation

- Code Review
  - Manuelle Prüfung von Code
  - Mehr oder weniger formal
  - Finden von Fehlern und Unklarheiten
  - Sicherstellung von Softwarequalität

Exkurs: Was ist Softwarequalität?

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

Motivation - Softwarequalität

- Höhere Softwarequalität
  - Weniger Stress
  - Software leichter erweiterbar
  - Geringere Projektlaufzeit oder mehr Features
  - Höhere Kundenzufriedenheit

Motivation - Wirtschaftlich

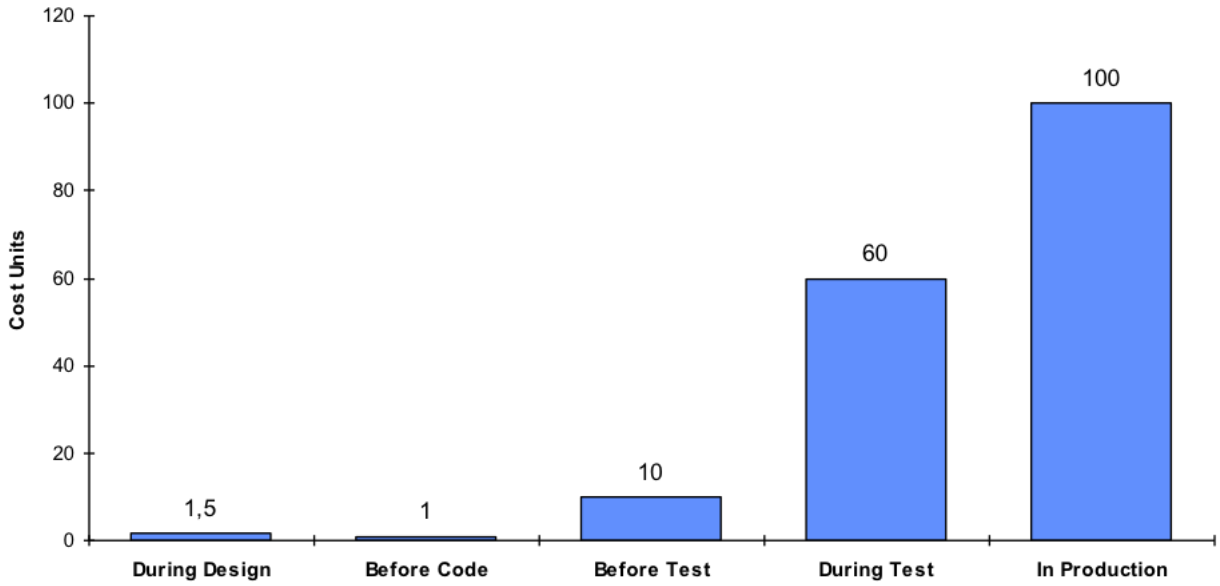

Abbildung: Kosten von Fehlern in Phasen eines Softwareprojektes [02]

Motivation - Wissensverbreitung

- Unerfahrene lernen von Erfahrenen
  - Autor ist ein unerfahrener Entwickler
  - Review durch erfahrenen Entwickler
  - Gutachter gibt Erfahrungen weiter
  - Team-interne Standards werden vermittelt

Motivation - Wissensverbreitung

- Erfahrene lernen von Unerfahrenen
  - Autor ist ein erfahrener Entwickler
  - Review durch unerfahrenen Entwickler
  - Gutachter kommt frisch von der Universität
  - Erfahrene Entwickler lernen ‚State of the Art‘
  - Neue Ansichten gelangen in das Team

Motivation - Wissensverbreitung

- Nur ein Entwickler kennt den Code
  - Entwickler erkrankt oder hat Urlaub
  - Entwickler verlässt das Unternehmen
- Mehrere Entwickler kennen den Code
  - Anderer Entwickler kann einspringen
  - Anderer Entwickler kann helfen und unterstützen

Motivation - Sind Testfälle nicht ausreichend?

- Testfälle
  - Finden 70% der Fehler
- Code Reviews
  - Finden 65% - 85% der Fehler
- Reviews und Testfälle kombiniert
  - Finden 96% - 99% der Fehler

[16]

Code Review-Arten

- Inspektion
- Pair Programming
- Walkthrough
- Automatisiert
- Task-basiert

Inspektion

- Mehrere vorbereitete Gutachter
  - Kein Autor
  - Kein Vorgesetzter
- Moderierte Sitzung
  - Neutraler Moderator
  - Diskussion über Produkt
- Gut geeignet für Entwurfsdokumente (frühe Phasen)

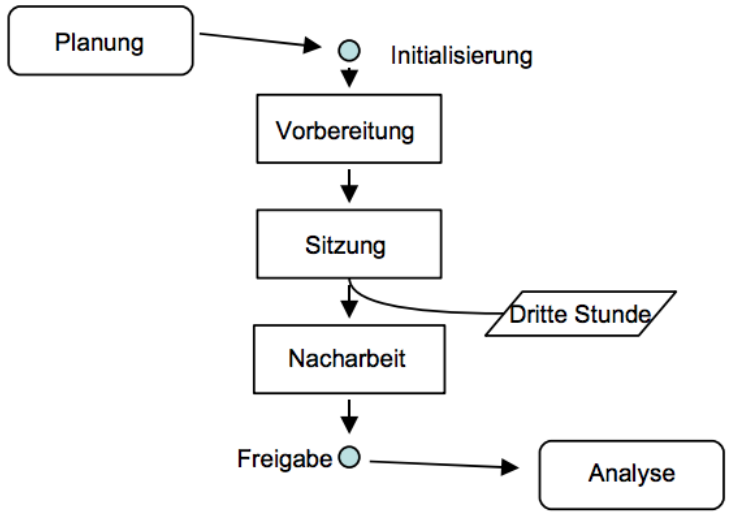

Abbildung: Vorgehensweise bei Inspektionen [07]

Pair Programming

- Ursprung aus dem Xtreme Programming
- Programmieren gemeinsam an einem Computer
- Pilot und Navigator
  - Pilot programmiert
  - Navigator überwacht
  - Rollentausch

Pair Programming

- Ständige Kommunikation
  - Navigator hinterfragt den Piloten
  - Pilot erklärt sein Vorgehen
- Regelmäßige Pausen
- Kompromissbereitschaft
- Kritikfähigkeit

Pair Programming - Anwendung

- Besonders geeignet für komplizierte Probleme
  - Qualität wird verbessert
  - Fehler werden früh erkannt
  - Mehr Spaß bei der Umsetzung (Kommunikation)
  - Wissensverbreitung / Anlernen
  - Kein nachträglicher Review notwendig

Walkthrough

- Schritt für Schritt Erklärung
- Autor
  - Legt Termin fest
  - Erklärt sein Produkt
- Teammitglieder
  - Stellen Fragen
  - Kommentieren und verbessern

Walkthrough - Anwendung

- Geeignet bei Unsicherheiten des Autors
  - Problem korrekt verstanden?
  - Problem geeignet umgesetzt/gelöst?
  - Gibt es Verbesserungspotential?
  - Frühe und richtige Wegweisung für den Autor

Automatisiert

- Leicht zu übersehene Fehler
- Vor manuellem Review
- Findet
  - Falsche Formatierungen
  - Bekannte Fehlermuster
- Beispiel: FindBugs
  - Geeignet für Java-Code
  - Erkennt gängige Fehlermuster

Automatisiert - Beispiel

- Fehlermuster: ==-Vergleich statt !=-Vergleich [04]

```
1  {
2      Object einObjekt = new Object();
3      //...
4
5      if(einObjekt == null){
6          System.out.println("Hash-Code= " +
               ↪ einObjekt.hashCode());
7      }
8
9      //...
10 }
```

Task-basiert

- Pro Task ein Review
- Fokus auf Codequalität
- Code-Standards vorausgesetzt
- Jedes Teammitglied macht Reviews
- Verteiltes Wissen über denselben Code
- Hohe Eigenverantwortung für Mitarbeiter

Exkurs: Task

- Abgeschlossene Teilaufgabe
- Mehrere Tasks ergeben eine Story
- Beispiel
  - Story: Warenkorb-Seite erstellen
  - Task: Kaufen-Button implementieren

Task-basiert - Tools

- Task-Management-Tool (z. B. Jira)
  - Verwaltung von Stories und Tasks
- Versionsverwaltung (z. B. Git)
  - Verwaltung des Quellcodes
- Review-Tool
  - Verwaltung der Reviews
  - Verbindung zwischen Task und Code

Task-basiert

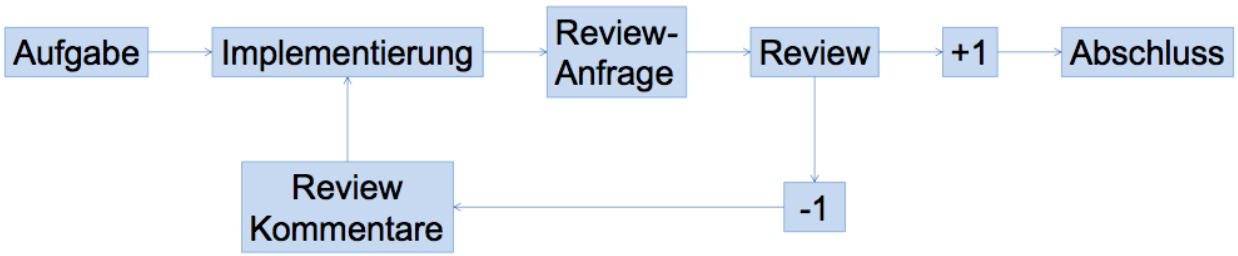

Abbildung: Ablauf eines Task basierten Reviews [06]

Task-basiert - Anwendung

- Tools einrichten
- Code-Standards festlegen
- Mitarbeiter schulen
  - Richtige Verwendung der Tools
  - Review-Standards
- Teammitglieder bekommen Zeit für Reviews

Tools

- GitLab
  - All-in-one-Lösung
  - Ermöglicht Task-Erstellung
  - Versionskontrolle vorhanden
  - Review-Workflow möglich
  - Geeignet für kleine Teams / Projekte

Tools

- Review-Worklfow
  1. Autor erstellt ‚Feature Branch‘
  2. Autor setzt Task um
  3. Autor erstellt ‚Merge Request‘ (Review)
  4. Gutachter führt Review durch und gibt Feedback
  5. Autor überarbeitet Code (ggf. mehrmals)
  6. Zweiter Gutachter führt Review durch
  7. Zweiter Gutachter führt Merge durch

Tools

- Upsource
  - Entwickelt von JetBrains
  - Software für Review-Verwaltung
  - Features
    - Automatisches Code Review
    - Analytische Informationen über Projekt
    - IDE-Plugin für IntelliJ
    - Repository-Übersicht (kein wechseln zu Git notwendig)
    - Kompatibel mit den größten Task-Tools und Versionsverwaltungen

Tools

- Review-Workflow
  1. Autor erstellt ‚Feature Branch‘
  2. Autor setzt Task um
  3. Autor committet Code und eröffnet Review in Upsource
  4. Gutachter übernimmt offenes Review und kommentiert
  5. Autor überarbeitet Code (ggf. mehrmals)
  6. Gutachter schließt Review
  7. Code wird gemerged

Live Demo

Live Demo

Best Practices

- Kritik nicht persönlich nehmen
  - Autor soll aus Fehlern lernen
- Nicht mehr als 200-400 Lines pro Review
  - Neuer Code erfordert hohen kognitiven Aufwand
- Genug Zeit (60-90 Minuten) für Review nehmen
  - Kognitiven Aufwand zeitlich aufteilen
  - Pausen

Best Practices

- Autor soll Code vorbereiten
  - Eigener Review vor dem gemeinsamen Review
- Messbare Ziele
  - Z. B. Anzahl Bug-Meldungen pro Woche
- ‚Big Brother'-Effekt vermeiden
  - Vorgesetzter beurteilt Code $\rightarrow$ Bewertung des Autors
  - Angst vor Review vermeiden

Zusammenfassung

- Code Review...
  - ...hilft früh Fehler aufzudecken
  - ...steigert die Softwarequalität
  - ...mindert die Kosten des Projekts
  - ...steigert die Kundenzufriedenheit
  - ...verteilt Wissen in dem Team
- Jeder Art hat sein Anwendungsgebiet

Zusammenfassung

- Review-Kultur entwickeln
  - Jede Review-Art in Betracht ziehen
  - Art für Problem und Zeitpunkt wählen
    - Entwurfsdokumente – Inspektion

Zusammenfassung

- Review-Kultur entwickeln
  - Jede Review-Art in Betracht ziehen
  - Art für Problem und Zeitpunkt wählen
    - Entwurfsdokumente – Inspektion
    - Entwurfsentscheidungen – Walkthrough

Zusammenfassung

- Review-Kultur entwickeln
  - Jede Review-Art in Betracht ziehen
  - Art für Problem und Zeitpunkt wählen
    - Entwurfsdokumente – Inspektion
    - Entwurfsentscheidungen – Walkthrough
    - Komplexes Problem – Pair Programming

Zusammenfassung

- Review-Kultur entwickeln
  - Jede Review-Art in Betracht ziehen
  - Art für Problem und Zeitpunkt wählen
    - Entwurfsdokumente – Inspektion
    - Entwurfsentscheidungen – Walkthrough
    - Komplexes Problem – Pair Programming
    - Kontinuierliches Review – Task basiert und automatisiert

Literatur I

[01] IT-Agile: Pair Programming. https://www.it-agile.de/wissen/agiles-engineering/pair-programming. Accessed: 2017-11-20.

[02] HU-Berlin: Code review. https://www2.informatik.hu-berlin.de/~hs/Lehre/2004-WS_SWQS/20050126_Reviews.pdf. Accessed: 2017-11-20.

[03] Smartbear: Secrets of peer code review. http://smartbear.com/SmartBear/media/pdfs/best-kept-secrets-of-peer-code-review.pdf. Accessed: 2017-11-20.

Literatur II

[04] Wikipedia: Find bugs. https://de.wikipedia.org/wiki/FindBugs. Accessed: 2017-11-20.

[05] Wikipedia: Review. https://de.wikipedia.org/wiki/Review_(Softwaretest),. Accessed: 2017-11-20.

[06] Javaland: Code review. https://www.doag.org/formes/pubfiles/6774263/2015-null-Rabea_Gransberger-Code_Reviews__Techniken_und_Tipps-Praesentation.pdf. Accessed: 2017-11-22.

Literatur III

[07] Uni Hannover: Reviews. http://www.se.uni-hannover.de/pub/File/kurz-und-gut/ss2006-seminar-werkSoft/Kurz_Gut-Nils_von_Delft_Reviews.pdf. Accessed: 2017-11-22.

[08] Smartbear: Best practices. https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/. Accessed: 2017-11-23.

[09] Uni Stuttgart: Code review. http://www.iste.uni-stuttgart.de/fileadmin/user_upload/iste/se/research/publications/download/sqmb08.pdf. Accessed: 2017-11-23.

Literatur IV

[10] Robert Bullinger: Software reviews. http://robert.bullinger.over-blog.com/article-ieee-1028-software-reviews-109244088.html. Accessed: 2017-11-23.

[11] KIT: Code reviews. https://sdqweb.ipd.kit.edu/wiki/Codereview. Accessed: 2017-11-23.

[12] Thorsten Maier: Code review best practices. https://blog.oio.de/2011/06/15/code-review-best-practices/. Accessed: 2017-11-27.

[13] Agile Modelling: User stories. http://www.agilemodeling.com/artifacts/userStory.htm. Accessed: 2017-11-27.

Literatur V

[14] GitLab: Code review worfklow. https://about.gitlab.com/2017/03/17/demo-mastering-code-review-with-gitlab/. Accessed: 2017-11-27.

[15] Upsource: Features. https://www.jetbrains.com/upsource/features/. Accessed: 2017-11-29.

[16] Software Productivity Research: Software quality. https://cs.nyu.edu/artg/Producing_Production_Quality_Software/Fall2005/lectures/SOFTWARE_QUALITY_IN_2002_CAPERS_JONES.pdf. Accessed: 2017-11-22.