Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

Code Review

vorgelegt von

Jan Ole Wellnitz

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

Studiengang:	Software-System-Entwicklung
Matrikelnummer:	6796226

Betreuer: Anna Fuchs

Hamburg, 2018-03-21

Abstract

In dieser Hausarbeit wird gezeigt, wie Code Reviews dabei helfen Softwareprojekte qualitativ und wirtschaftlich zu verbessern. Dazu werden verschiedene Code Review-Arten vorgestellt. Der Fokus liegt hierbei auf den Task-basierten Reviews, die mit Hilfe von Hilfsprogrammen durchgeführt werden können. Für die Durchführung der Reviews werden Best Practices vorgestellt, die dabei Helfen den reibungslosen Ablauf von Code Reviews zu gewährleisten.

Contents

1 Einleitung und Motivation

Der Prozess des Code Reviews ist eine manuelle Prüfung von Code [01]. Er kann formal mit Hilfe von dokumentierten Abläufen durchgeführt werden. Jedoch ist eine weniger formale Durchführung durch ein nicht dokumentiertes Gespräch über ein Produkt ebenfalls als Code Review zu verstehen. Das Ziel von Code Reviews ist das Finden von Fehlern und Unklarheiten. Auf diese Weise kann die Softwarequalität in Projekten verbessert werden [01].

1.1 Softwarequalität

Die Softwarequalität wird durch die Begriffe Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit definiert [13]. Diese Parameter bestimmen, inwiefern ein Softwareprodukt die Anforderungen erfüllt, für den Benutzer bedienbar ist und wartbar beziehungsweise erweiterbar ist.

Durch den Einsatz von Code Reviews sind mehrere Personen für eine Softwarekomponente verantwortlich. Das heißt, dass mehrere Personen die Softwarequalität für diese Komponente kontrollieren. Durch diese mehrfache Kontrolle steigt die Softwarequalität des Produktes. Die höhere Qualität ist ein Grund, dass die Software in späten Projektphasen weniger Fehler aufweist. Das führt zu weniger Stress bei den Entwicklern und zu einem früheren Projektende [05].

Durch den früheren Projektabschluss erhält der Kunde die Software früher. Zudem ist die Software aufgrund der höheren Softwarequalität fehlerfreier. Das Ergebnis ist eine höhere Kundenzufriedenheit [05].

1.2 Wirtschaftlich

Aus wirtschaftlicher Sicht haben Code Reviews einen hohen Nutzen. In frühen Projektphasen ist eine Fehlerkorrektur günstiger als in späteren Phasen. Werden Fehler in der Planungsphase entdeckt, betragen die Kosten nur 1/100 der Fehlerbehebungskosten während des Produktionsbetriebes. In der Entwicklungsphase kostet eine Korrektur nur 1/10 der Fehlerbehebungskosten im laufenden Betrieb (siehe Abbildung 1.1) [03].

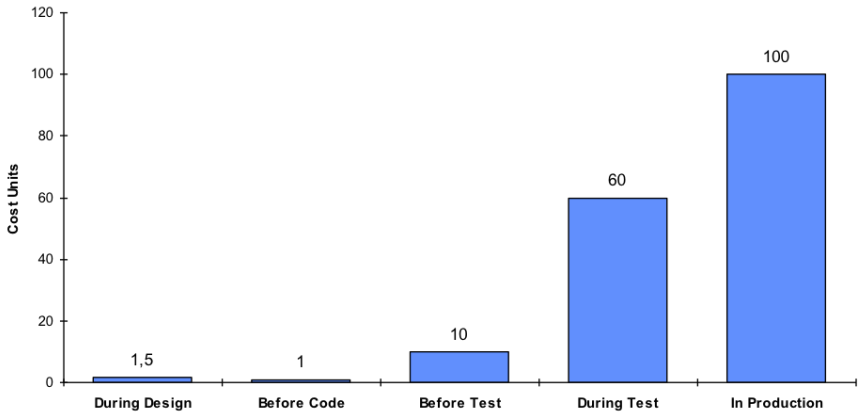

Abbildung 1.1: Kosten von Fehlern in Phasen eines Softwareprojektes [03]

1.3 Wissensverbreitung

Code Reviews fördern die Wissensverbreitung in einem Team [14]. Erfahrene Entwickler geben ihre Erfahrungen an neue junge Mitarbeiter weiter. Sie vermitteln interne Standards und erleichtern den Einstieg in das neue Projekt. Führen junge Mitarbeiter Reviews durch, lernen sie den Programmierstil der erfahrenen Entwickler kennen. Sie müssen versuchen, den Code der erfahrenen Entwickler zu verstehen. Das Gelernte können die jungen Entwickler anwenden, wenn ein ähnliches Problem erneut auftritt [14].

1.4 Sind Testfälle nicht ausreichend?

Die Firma Software Productivity Research fand 2002 in einer Studie heraus, dass 70% der bekannten Fehler in einem Softwareprojekt durch Testfälle entdeckt werden können. Durch Code Reviews können 65% - 85% gefunden werden. Die Kombination von Testfällen und Code Reviews findet 96% - 99% der bekannten Fehler [12]. Daraus folgt, dass weder Testfälle noch Reviews alleine das optimale Ergebnis erbringen. Die Kombination aus beiden Verfahren legt die meisten Fehler offen und bietet das beste Ergebnis.

2 Code Review-Arten

In diesem Kapitel werden fünf Review-Arten vorgestellt. Es wird erläutert, wann und wie diese Techniken einzusetzen sind.

2.1 Inspektion

Die Inspektion wird in einer moderierten Sitzung durchgeführt. Vor der Sitzung schickt der Autor das Produkt an alle zuvor ausgewählten Gutachter. Das Produkt kann in diesen Verfahren auch ein Entwurfsdokument sein. In dieser Sitzung diskutieren die Gutachter, von denen keiner der Autor ist, das zuvor eingereichte Produkt. Der Moderator nimmt eine neutrale Position ein. Alle Ergebnisse der Diskussion werden festgehalten und der Autor setzt die erarbeiteten Verbesserungen um. Anschließend wird das überarbeitete Produkt erneut von den Gutachtern kontrolliert und freigegeben [06].

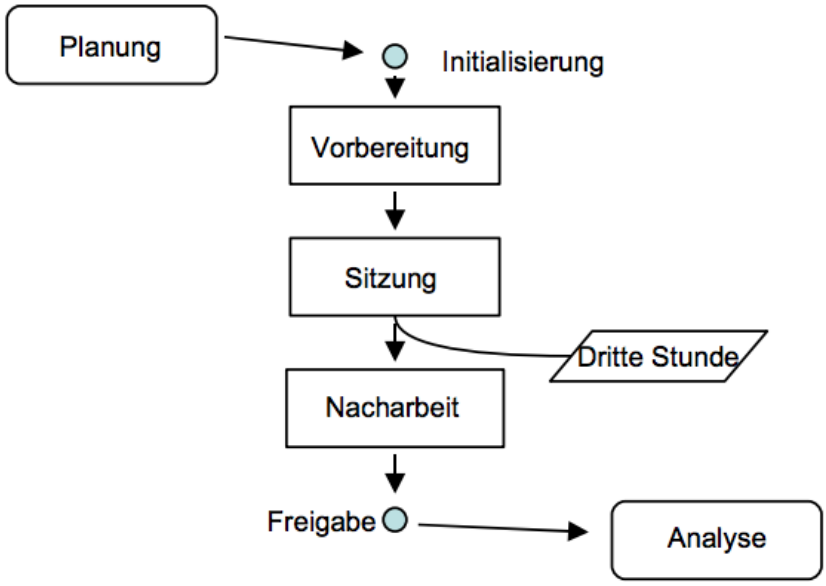

Abbildung 2.1: Vorgehensweise bei Inspektionen [06]

2.2 Pair Programming

Die Ursprünge des Pair Programming stammen aus der agilen Vorgehensweise Extreme Programming [20]. Hierbei ist es vorgesehen, dass im gesamten Entwicklungsprozess durchgehend zwei Entwickler an einem Computer zusammenarbeiten. Dabei gibt es einen Piloten und einen Navigator. Der Pilot sitzt an der Tastatur und Maus. Der Navigator sitzt daneben, überwacht den Entwicklungsprozess und gibt die Richtung vor. Diese Rollen werden in regelmäßigen Abständen getauscht, sodass die Entwickler beide Rollen zu gleichen Zeitanteilen besetzen. Besonders wichtig ist die durchgängige Kommunikation zwischen Pilot und Navigator. Der Pilot erklärt sein Vorgehen und begründet es. Der Navigator hinterfragt das Handeln des Piloten. Dieser Prozess fordert von beiden Entwicklern einen hohen kognitiven Aufwand, sodass regelmäßig Pausen einzulegen sind. Damit der Prozess erfolgreich gestaltet werden kann, müssen beide im höchsten Maße kompromissbereit und kritikfähig sein [02]. Das Ziel ist es, gemeinsam das bestmögliche Produkt zu entwickeln.

Anwendung

Das Pair Programming ist besonders für komplizierte und kritische Bereiche eines Programmes geeignet. Diese Abschnitte werden fehlerfreier, lesbarer und weisen eine höhere Qualität auf [20]. Ebenfalls wird das Wissen über diese Probleme in dem Team verteilt, weil zwei Entwickler aktiv und gemeinsam die Komponente entwickeln [02]. Daraus folgt, dass das Review selbst während des Programmierens durchgeführt wird. Es ist kein nachträgliches Review notwendig.

2.3 Walkthrough

Der Walkthrough ist eine schrittweise Erklärung eines Produktes wie zum Beispiel eines User-Interface oder Programm-Codes. Der Autor stellt das von ihm erstellte Produkt einer ausgewählten Gruppe von Personen vor. Die Gutachter sind vorrangig Teammitglieder, die in demselben Kontext arbeiten. Sie stellen Fragen während der Vorstellung, kommentieren das Produkt und machen Verbesserungsvorschläge [06].

Anwendung

Der Walkthrough kommt zur Anwendung, wenn der Autor des Produktes Unsicherheiten bezüglich der Richtigkeit hat [15]. In diesen Fällen ist der Autor unsicher, ob er das Problem korrekt verstanden oder umgesetzt hat. Aber auch kann Verbesserungspotenzial aufgedeckt werden, damit der Autor früh einen korrekten Lösungsansatz wählt.

2.4 Automatisiert

Automatisierte Reviews werden verwendet, um leicht zu übersehene Fehler aufzudecken. Das sind vor allem Fehler, die durch Unaufmerksamkeit des Autors entstehen. Dazu gehören zum Beispiel falsche Formatierungen oder bekannte Fehlermuster. Ein Beispiel für ein Fehlermuster ist die Wahl eines falschen Vergleiches. Statt auf Ungleichheit wird auf Gleichheit geprüft [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 }
```

Listing 2.1: Beispiel eines Fehlermuster: ==-Vergleich statt !=-Vergleich [04]

Anwendung

Das automatisierte Review soll vor dem manuellen Review durchgeführt werden [05]. Die Fehler, die durch die automatisierten Reviews entdeckt werden, können den Gutachter von den inhaltlichen Fehlern ablenken. So kann sich der Gutachter vollständig auf die inhaltlichen Aspekte des Codes konzentrieren und Fehler in der Logik aufdecken.

2.5 Task-basiert

Bei dem Task-basierten Review-Prozess wird für jeden umgesetzten Task ein Review durchgeführt.

Task

Ein Task ist eine abgeschlossene Teilaufgabe. Mehrere Tasks ergeben eine Story. Die Story umfasst eine größere Aufgabe [09]. Ein Beispiel für eine Story ist die Erstellung einer Warenkorb-Funktion auf einer Website. Die Erstellung der Warenkorb-Seite ist in diesem Beispiel die Story. Ein Task dieser Story kann die Erstellung des Kauf-Buttons sein.

Tools

Für die Durchführung der Task-basierten Reviews ist die Verwendung von Tools zu empfehlen. Es wird ein Task-Management-Tool benötigt. Beispiele sind die Tools Jira, speziell für die Softwareentwicklung, und Trello, ein leichtgewichtiges Management-Tool [16, 21]. Dort werden die Stories und Tasks verwaltet. Jedes Teammitglied kann den Status eines Tasks sowie den Bearbeiter einsehen. Der Quellcode wird in einer Versionsverwaltung wie Git gehalten. Alle Teammitglieder haben darauf Zugriff und können ihre Änderungen hochladen [18]. Die Schnittstelle zwischen den Tasks und dem Code bildet das Review-Tool. Dort werden die Code-Änderungen der Teammitglieder den Tasks zugeordnet. Diese Zuordnung kann automatisch über Schnittstellen erfolgen oder manuell von den Teammitgliedern gepflegt werden. Für jeden Task kann in dem Tool ein Review eröffnet, durchgeführt und dokumentiert werden. Das Tool Upsource von JetBrains sowie das Tool Crucible von Atlassian sind Beispiele für Review-Tools [11, 17].

Die Wahl der Tools sollte mit Bedacht erfolgen. Nicht alle Tools sind kompatibel zueinander und können Daten automatisch untereinander austauschen. Ein weiterer Aspekt ist die Einrichtung beziehungsweise das Aufsetzen der Tools. Ist das Projekt groß und langfristig geplant, sollte mehr Zeit für die Einrichtung eingeplant werden. Bei kleinen kurzfristigen Projekten sind leichtgewichtige Tools von Vorteil. Diese benötigen weniger Zeit für die Einrichtung, bieten jedoch nicht alle Funktionen.

Ablauf

Zu Beginn wird einem Teammitglied ein Task zugewiesen. Anschließend wird der Task bearbeitet, der Code in die Versionsverwaltung geladen und ein Review eröffnet. Ein anderes Teammitglied führt dieses Review durch. Ist der Task erfolgreich bearbeitet, wird das Review geschlossen und der Task ist erfolgreich beendet. Gibt es Verbesserungsvorschläge seitens des Gutachters, werden diese in dem Review vermerkt. Das Review geht zurück an den Autor. Dieser setzt die Verbesserungen um und eröffnet das Review erneut. Dieser Prozess wird so lange wiederholt bis keine Verbesserungsvorschläge mehr gefunden werden.

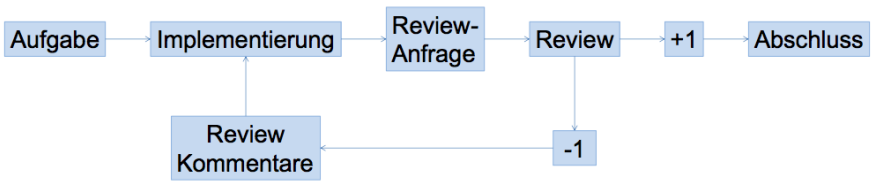

Abbildung 2.2: Ablauf eines Task-basierten Reviews [05]

3 Tools

Im folgenden Kapitel werden zwei Tools vorgestellt, mit denen Task-basierte Reviews umgesetzt werden können.

3.1 GitLab

Die Versionsverwaltung GitLab ist eine All-in-one-Lösung. Sie ist nicht gezielt für diese Funktionalität entwickelt worden. Durch einen Workflow wird ein Review-Verfahren jedoch ermöglicht. Mit dieser Lösung können kleine Teams oder Projekte schnell und unkompliziert Reviews durchführen [10].

Workflow

Der Entwickler erstellt im ersten Schritt einen 'Feature Branch'. Dieser wird von dem aktuellen Entwicklungstand abgezweigt. In diesen Branch wird nur dieser Task umgesetzt. Wenn der Entwickler den Task bearbeitet hat, erstellt er einen 'Merge Request'. Dieser 'Merge Request' ist eine Anfrage, um den eigenen Branch in den 'Develop Branch' zurückzuführen. Bevor die Zusammenführung jedoch erfolgt, muss ein Review von einem weiteren Entwickler durchgeführt werden. Findet der Gutachter Mängel in dem Code, lehnt er den Request ab und gibt ein Feedback an den Entwickler zurück. In diesem Feedback befinden sich Verbesserungsvorschläge, die der Entwickler in dem 'Feature Branch' nachbessern muss. Anschließend stellt der Autor wieder einen 'Merge Request', dieser soll jedoch vorzugsweise von einem anderen Gutachter durchgeführt werden. Dieser Prozess wird so lange wiederholt bis es keine Verbesserungen mehr gibt. Anschließend werden die Branches zusammengeführt und der Task ist erfolgreich bearbeitet [10].

3.2 Upsource

Das von JetBrains entwickelte Tool Upsource ist spezialisiert auf die Durchführung von Code Reviews. In diesem Tool können Reviews eröffnet, durchgeführt und verwaltet werden. Es muss mit einem Versionsverwaltungstool wie GitLab verbunden werden. Upsource synchronisiert sich mit dem Versionsvewaltungstool und zeigt alle getätigten Code-Änderungen an. Für diese sogenannten Commits können Reviews eröffnet werden.

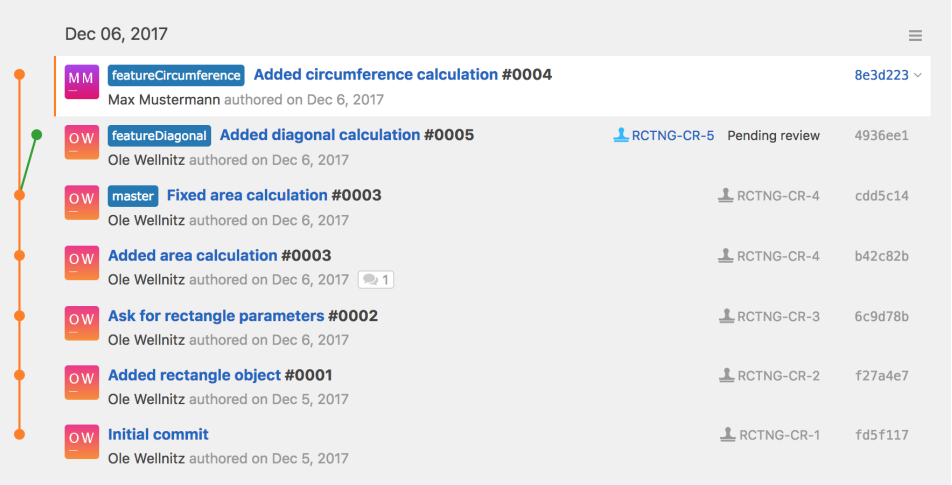

Abbildung 3.1: Ansicht der Commits und Review-Status

Sämtliche Reviews und Tätigkeiten bezüglich eines Reviews werden in Upsource vermerkt und gespeichert. So können alle Teammitglieder den aktuellen Status der Tasks beziehungsweise Reviews beobachten.

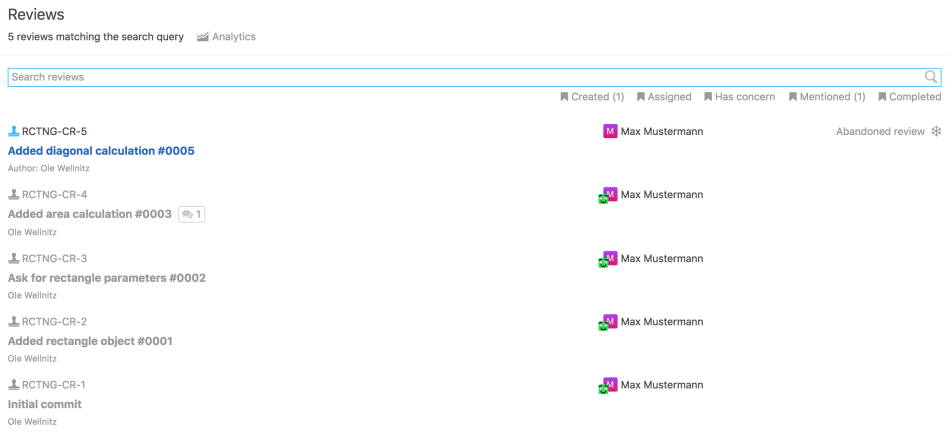

Abbildung 3.2: Ansicht aller offenen und geschlossenen Reviews

Ebenfalls ist in einer Summary-Ansicht der zeitliche Verlauf eines Reviews ersichtlich. In dieser Ansicht wird vermerkt, wer den Review durchgeführt hat und welche Anmerkungen gemacht wurden.

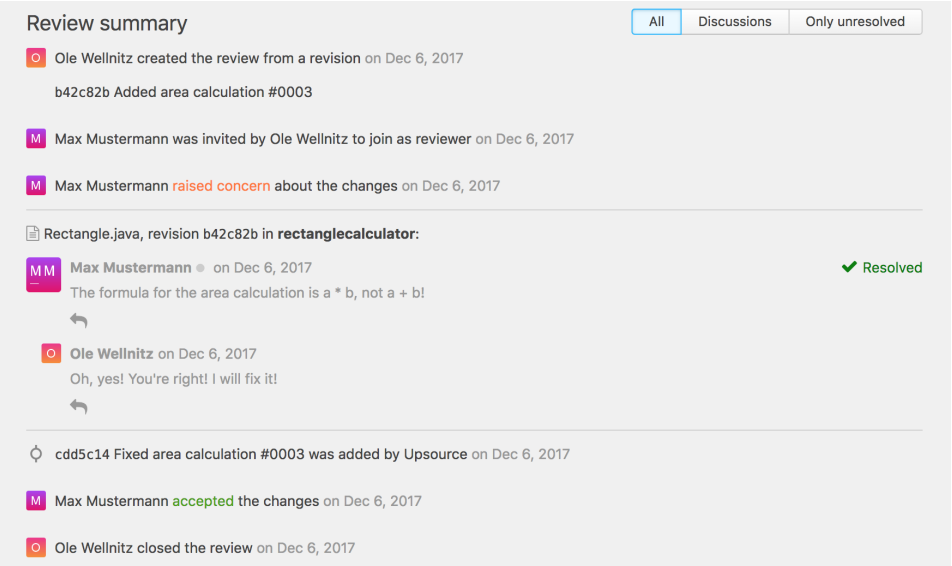

Abbildung 3.3: Ansicht einer Review-Zusammenfassung

Das Tool bietet weitere nützliche Funktionen wie die Integration von Task-Management-Tools, automatische Code Reviews, analytische Auswertungen über Reviews und ein PlugIn für IntelliJ an [11].

Workflow

Wie bei dem GitLab-Workflow muss der Entwickler zuerst einen 'Feature Branch' erstellen und den Task umsetzen. Die Commits des Tasks werden in Upsource angezeigt und der Entwickler eröffnet für diese Commits ein Review. Er kann ein bestimmtes Teammitglied einladen das Review durchzuführen oder einem beliebigen Teammitglied wird das Review aus der Liste der offenen Reviews zugewiesen. Das Teammitglied führt in Upsource das Review durch und vermerkt seine Verbesserungsvorschläge an den dazugehörigen Code-Zeilen. Wenn es nach dem Review weitere Verbesserungsvorschläge gibt, wird das Review abgelehnt. Der Entwickler muss anhand der Kommentare das Produkt verbessern und seinen Code hochladen. Die neuen Commits fügt er dem Review hinzu und eröffnet das Review erneut. Dieser Prozess wird so lange wiederholt bis der Gutachter das Review akzeptiert. Dann werden die Branches wieder zusammengeführt [19].

4 Best Practices

Im folgenden Abschnitt werden sechs Best Practices vorgestellt, auf die während eines Reviews zu achten ist. Sie sollen dabei helfen die Qualität der Reviews zu verbessern.

4.1 Kritik nicht persönlich nehmen

Der Autor soll die Kritik und Verbesserungsvorschläge objektiv aufnehmen. Sie stellen keinen Angriff auf die Person selbst dar. Die Vorschläge sollen dabei helfen, das Endprodukt zu verbessern. Der Autor soll diese Kritik als Chance sehen, um aus dem Fehler zu lernen [07].

4.2 Nicht mehr als 200-400 Lines pro Review

Das Lesen und Verstehen fremden Codes erforderte einen hohen kognitiven Aufwand. Der Umfang des Reviews soll deshalb nicht mehr als 200-400 'Lines of Code' umfassen [07]. Mit der Zeit nimmt die kognitive Leistung des Gutachters ab. Muss dieser zu viele Zeilen lesen und verstehen, erhöht sich die Wahrscheinlichkeit einen Fehler zu übersehen.

4.3 Genug Zeit (60-90 Minuten) für Review nehmen

Aufgrund der in 4.2 beschriebenen hohen kognitiven Belastung ist es wichtig, dass der Gutachter genügend Zeit für das Review hat. Um die vorgeschlagenen 200-400 'Lines of Code' zu prüfen, sollen dem Gutachter 60-90 Minuten zur Verfügung stehen [07]. So kann er Pausen machen und gerät nicht in Stress. Auch diese Maßnahme fördert die Qualität des Reviews. Ist der Gutachter kognitiv überlastet oder hat Zeitdruck, ist die Wahrscheinlichkeit hoch, dass Fehler übersehen werden.

4.4 Autor soll Code vorbereiten

Bevor das Review von einer anderen Person durchgeführt wird, soll der Autor selbst ein Review durchführen. Hierbei können die leicht zu findenden Fehler von dem Autor selbst verbessert werden. Diese Fehler müssen von dem Gutachter nicht behoben werden. Der Gutachter hat dementsprechend mehr Zeit und kognitive Leistung zu Verfügung, um die Fehler zu finden, die der Autor selbst nicht gefunden hat [07].

4.5 Messbare Ziele

Um feststellen zu können, dass der Einsatz von Reviews einen positiven Nutzen hat, müssen messbare Ziele definiert werden. Mit diesen Zielen kann nach einem festgelegten Zeitraum festgestellt werden, ob Verbesserungen in gewünschten Bereichen eingetreten sind [07]. Als Beispiel kann die Anzahl der Bug-Meldungen pro Woche gemessen werden. Nach der Einführung von Code Reviews werden einige Wochen die Bug-Meldungen gemessen. Diese Zahlen werden mit der Anzahl der Bug-Meldungen vor der Umstellung verglichen. Wenn die Anzahl der Bug-Meldungen abgenommen hat, kann die Umstellung als erfolgreich betrachtet werden. Die Mitarbeiter sehen so, dass Code Reviews einen positiven Nutzen haben. Das führt zu einer erhöhten Motivation, Code Reviews weiter oder intensiver zu betreiben.

4.6 Big Brother-Effekt vermeiden

Die Gutachter der Reviews beurteilen den Code des Autors. Wird diese Rolle von einem Vorgesetzten eingenommen, kann dies indirekt auf die Beurteilung des Mitarbeiters Einfluss nehmen. Diese Tatsache kann zu Angst bei den Entwicklern führen, da sich die Fehler negativ auf ihre persönliche Beurteilung ausüben. Das wird als Big Brother-Effekt bezeichnet [08]. Diesen Effekt gilt es zu vermeiden, sodass nur das Produkt bewertet wird und nur die Qualität des Produktes im Fokus steht. Der Autor selbst soll nicht anhand des Produktes bewertet werden. Das Ziel ist es, gemeinsam das bestmögliche Produkt zu schaffen.

5 Zusammenfassung

Code Reviews sind eine Methode, um die Qualität des Softwareproduktes in vielen Bereichen zu verbessern. Mit Code Reviews können Fehler früher entdeckt und behoben werden. Das Beheben von Fehlern ist in frühen Phasen günstiger. Auf diese Weise können die Projektkosten und Projektlaufzeit gesenkt werden. Diese Effekte können die Kundenzufriedenheit erhöhen.

Das Entwicklerteam selbst profitiert ebenfalls von Code Reviews. Zum einen werden Kompetenzen auf mehrere Teammitglieder aufgeteilt, zum anderen können neue Teammitglieder leichter in das Team integriert werden. Zudem wird der Stress gegen Ende des Projektes reduziert, weil weniger Fehler auftreten. Die meisten Fehler wurden bereits in frühen Projektphasen mit wenig Aufwand eliminiert und treten folglich nicht mehr auf.

Wenn Code Reviews angewendet werden, sollte darauf geachtet werden, alle zu Verfügung stehenden Arten in Betracht zu ziehen. Für jede Phase und jedes Problem gibt es eine angepasste Review-Art. Um Entwurfsdokumente zu untersuchen, bietet sich die Inspektion an. Bei Entwurfsentscheidungen und bei Unsicherheiten seitens des Entwicklers können Walkthroughs dazu genutzt werden, Missverständnisse zu beseitigen. Wenn ein komplexes Problem auftritt oder ein kritischer Programmteil entwickelt wird, kann Pair Programming als Möglichkeit betrachtet werden. So findet bereits während der Entwicklung ein Review statt. Allerdings sollten auch kontinuierliche Reviews durchgeführt werden. Auf diese Weise werden während der gesamten Projektlaufzeit Reviews durchgeführt, Mitarbeiter teilen ihr Wissen und die allgemeine Qualität des Produktes steigt.

Wichtig ist zu dem richtigen Zeitpunkt die richtige Review-Art zu wählen.

Bibliography

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

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

[03] Prof. Dr. Holger Schlingloff, HU-Berlin: Qualitätssicherung von Software. https://www2.informatik.hu-berlin.de/~hs/Lehre/2004-WS_SWQS/20050126_Reviews.pdf. Accessed: 2017-11-20.

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

[05] Rabea Gransberger, Javaland: Code Reviews: Techniken und Tipps. https://www.doag.org/formes/pubfiles/6774263/2015-null-Rabea_Gransberger-Code_Reviews__Techniken_und_Tipps-Praesentation.pdf. Accessed: 2017-11-22.

[06] Nils von Delft, Uni Hannover: Reviews, Inspektionen und Walkthroughs. 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.

[07] Smartbear: Best practices for peer code review. http://support.smartbear.com/support/media/resources/cc/11_Best_Practices_for_Peer_Code_Review.pdf. Accessed: 2017-11-23.

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

[09] Agile Modelling: User Stories: An Agile Introduction. http://www.agilemodeling.com/artifacts/userStory.htm. Accessed: 2017-11-27.

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

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

[12] Capers Jones, 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.

[13] ISO/IEC 9126 - Software quality: Software quality. https://www.cse.unsw.edu.au/~cs3710/PMmaterials/Resources/9126-1%20Standard.pdf. Accessed: 2017-11-25.

[14] Philippe Schrettenbrunner: Code reviews – neue Perspektiven auf die Code-Basis. https://www.maibornwolff.de/blog/code-reviews. Accessed: 2018-01-29.

[15] Shweta Kulkarni: Code Review vs Code Walkthrough. http://www.anarsolutions.com/code-review-vs-code-walkthrough/. Accessed: 2018-01-29.

[16] Atlassian: Jira. https://de.atlassian.com/software/jira. Accessed: 2018-02-22.

[17] Atlassian: Crucible. https://de.atlassian.com/software/crucible. Accessed: 2018-03-20.

[18] Atlassian: What is Git. https://www.atlassian.com/git/tutorials/what-is-git. Accessed: 2018-02-22.

[19] JetBrains: Upsource Workflow. https://blog.jetbrains.com/upsource/2017/04/13/automating-your-code-review-workflow-with-upsource/. Accessed: 2017-02-22.

[20] Raymund Sison, College of Computer Studies, De La Salle University: Investigating the Effect of Pair Programming and Software Size on Software Quality and Programmer Productivity. https://pdfs.semanticscholar.org/9ca1/c9a5fb00c58e56152f5daae151ee37744293.pdf. Accessed: 2018-02-24.

[21] Trello. https://trello.com/. Accessed: 2018-03-20.