[Textfassung zu Folien AgileInspektionen_Roesler_ESE2010_v1.3.ppt. Der Vortrag wurde auf dem ESE-Kongress am 09.12.2010 gehalten]

Agile Inspektionen

Peter Rösler, ros@reviewconsult.de

Agile Inspektionen sind eine neue Entwicklung auf dem Gebiet der Reviewtechnik. Mit ihnen werden größere Qualitätsverbesserungen und damit höhere Kosteneinsparungen erreicht als mit klassischen Inspektionen. Im Gegensatz zu den klassischen Inspektionen liegt der Schwerpunkt bei den Agilen Inspektionen vorrangig auf der steilen Lernkurve der Entwickler und nicht mehr auf den gefundenen Fehlern im Dokument.

1. Wasserfallmodell und klassische Inspektionen

In einem typischen Wasserfall- bzw. V-Modell gibt es aufeinanderfolgende Phasen wie Anforderungsdefinition, Fachkonzept, Design, Codierung, Modultest, Integrationstest, Systemtest und Abnahmetest. Die Fehler, die die Entwickler in den Phasen Anforderungsdefinition bis Codierung machen, werden meist erst in den Testphasen entdeckt, also zu einem relativ späten Zeitpunkt. Der Aufwand, die Fehler dann zu korrigieren („Rework-Aufwand“), ist zum Teil erheblich und nimmt in einem typischen Softwareprojekt in der Summe ungefähr die Hälfte des Entwicklungsaufwands ein.

Seit ungefähr 1975 werden daher in vielen Projekten Software-Inspektionen eingesetzt. Diese Inspektionen (oft auch „Software-Reviews“ genannt) dienen dazu, sofort nach Fertigstellung eines Dokuments („Artefakts“) möglichst viele Fehler zu entdecken. Die gravierenden Fehler, also die Fehler, die später Zeit kosten und damit Rework-Aufwand verursachen, bezeichnet man als „Major Defects”. Mit richtig durchgeführten Inspektionen kann man ungefähr die Hälfte bis zu 2/3 der in den Dokumenten vorhandenen Major Defects finden und damit den Rework-Aufwand entsprechend verringern. Der Gesamt-Entwicklungsaufwand des Projekts reduziert sich damit um ungefähr 25 % bis 35 %.

2. Agile Software-Entwicklung

2.1 Prinzipien

Eine Gemeinsamkeit der verschiedenen agilen Entwicklungsmethoden ist, dass das zu entwickelnde System in vielen Iterationen (oder „Sprints“) entsteht. Iterationen dauern üblicherweise ein bis vier Wochen. Am Ende jeder Iteration steht ein getestetes und lauffähiges System („potentially shippable code“) mit neu dazu gekommener Funktionalität. Agile Entwicklungsmethoden bedienen sich in unterschiedlichem Ausmaß weiterer agiler Praktiken wie Test-Driven Development (TDD), Refactoring etc. Im Gegensatz zu Wasserfall-Projekten kann in agilen Projekten auf sich ändernde Anforderungen oder Prioritäten schnell reagiert werden. Es gibt einen sehr frühen Return on Investment, weil der Kunde schon nach der ersten Iteration ein lauffähiges System hat, das er produktiv einsetzen kann. Ein komplettes Scheitern des Projekts (kein lauffähiger Code), wie es bei Wasserfall-Projekten immer wieder vorkommt, ist wenig wahrscheinlich.

2.2 Pair Programming und anderes „Reviewartiges“ in der agilen Software-Entwicklung

Pair Programming

Pair Programming ist Teil der agilen Entwicklungsmethode XP (Extreme Programming), kann aber auch in anderen Entwicklungsmethoden eingesetzt werden. Zwei Entwickler arbeiten gemeinsam am Bildschirm. Der „Pilot“ („Driver“) schreibt den Code. Der „Navigator“ („Observer“) prüft den Code und denkt über Verbesserungen am Design nach. Wenn nötig, wird sofort diskutiert. Die Rollen werden oft gewechselt, z.B. alle paar Minuten. Die Paarzusammensetzungen im Projektteam werden ebenso gewechselt, beispielsweise zweimal am Tag. Der Vorteil von Pair Programming ist die höhere Qualität der Software (je nach Studie 15 % bis 50 % weniger Fehler). Viele Fehler können sofort am Bildschirm entdeckt werden und ein besseres Design kann gefunden werden. In Projekten, die Pair Programming einsetzen, kennen sich mit jedem Teil des Codes mindestens zwei Entwickler sehr gut aus. Damit ist das Projekt weniger stark gefährdet, wenn einzelne Projektmitarbeiter zum Beispiel durch Krankheit oder Kündigung ausfallen. Der Aufwand für Pair Programming ist nicht, wie man vielleicht annehmen könnte, 100 % mehr als bei Einzelprogrammierung, sondern (je nach Studie) 15 % bis 85 % mehr. Diesem Mehraufwand stehen durch die verbesserte Qualität Einsparungen später im Projekt gegenüber.

Design- und Code-Reviews in FDD (Feature Driven Development)

In der agilen Entwicklungsmethode FDD gibt es für jedes zu entwickelnde Feature auch die Meilensteine „Design Inspection“ und „Code Inspection“. In der Design-Inspektion werden das Design und Abnahmetests geprüft. In der Code-Inspektion werden der Code und Abnahmetests geprüft. Der Chef-Programmierer als Leiter des Projektteams bestimmt den Grad der Formalität der jeweiligen Inspektion. Im Gegensatz zu Reviews in Wasserfall-Projekten ist bei den Inspektionen in FDD die Gefahr von psychologischen Problemen geringer, weil der Owner des zu prüfenden Artefakts nicht eine Einzelperson ist, sondern das gesamte Feature-Team. Der Einsatz von Inspektionen in FDD wird mit wissenschaftlichen Untersuchungen begründet, die zeigen, dass Code-Inspektionen Fehler mit weniger Aufwand beseitigen können als Tests. Im Gegensatz zur Entwicklungsmethode XP mit seinem Pair Pogramming vertraut FDD voll auf klassische Inspektionen. Das ist einer der Gründe, warum FDD unter den agilen Entwicklungsmethoden als diejenige angesehen wird, die den klassischen Vorgehensmethoden am nächsten steht.

Sprint Review Meetings in Scrum

In der agilen Entwicklungsmethode Scrum erfolgt nach einem Sprint ein informelles Review durch Team, Product Owner und Stakeholder. Dazu wird das Ergebnis des Sprints durch das Team vorgeführt. Product Owner und Stakeholder geben Feedback, das in die weitere Arbeit mit einfließt. Im Sprint Review Meeting werden keine Folien gezeigt, auch kein Quell-Code, sondern es wird die laufende Software vorgeführt. Das Sprint Review Meeting ist also kein Review im Sinne einer Inspektion, welche Fehler auf Artefakt-Ebene finden will. Das Sprint Review Meeting trägt vielmehr einige Merkmale eines Abnahmetests. Reines Scrum hat erstmal keine „reviewartigen“ Mechanismen im Sinne von Inspektionen. Daher ist es empfehlenswert, in Scrum-Projekten zusätzlich (mindestens) die Praktiken Pair Programming oder Inspektionen einzusetzen.

3. Agile Inspektionen

Agile Inspektionen wurden von Tom Gilb im Jahr 2005 vorgestellt [1] und werden auch „Extreme Inspections“ oder „Agile Specification Quality Control (SQC)“ [5] genannt. Agile Inspektionen sind besonders in Wasserfall-Projekten nützlich, weil es dort keine kurzen Feedback-Schleifen (z.B. in Form von Iterationen) gibt.

Ausgangspunkt für Agile Inspektionen sind Beobachtungen, die beim Einsatz von klassischen Inspektionen gemacht wurden. Bei klassischen Inspektionen kommt es manchmal vor, dass die Autoren aufgrund der Review-Erfahrungen beginnen, deutlich fehlerfreier zu arbeiten. Es gibt mehrere Beispiele solcher Lernkurven, in denen es den Autoren gelungen ist, nach einiger Zeit ungefähr 10 Mal weniger Major Defects pro Seite zu machen als vorher (was den Rework-Anteil im Projekt um Faktor 10 sinken lässt). Der Lernprozess dauerte wenige Monate und die Autoren nahmen während dieser Zeit an ca. fünf bis sieben Inspektionen teil.

Aus diesen Beobachtungen entstand die Grundidee der Agilen Inspektionen: Der Schwerpunkt der Inspektionen wird verschoben weg vom frühzeitigen Fehler finden und korrigieren („cleanup“-Modus) hin zum Schätzen der Fehlerdichte der Dokumente, um die Entwickler zu motivieren, zu lernen wie man von vorne herein fehlerfreier arbeitet. Dadurch sinken die Kosten für Inspektionen enorm: es müssen nicht mehr alle Seiten aller Dokumente geprüft werden, sondern Stichproben reichen aus.

Das Hauptziel der Agilen Inspektionen ist es also, die Entwickler zu motivieren, ihre Fehlerrate zu senken. Zusätzliche Ziele sind, genauso wie bei klassischen Inspektionen: verhindern, dass zu fehlerhafte Dokumente in die folgenden Software-Entwicklungsphasen gelangen (Terminverzögerungen und Qualitätsprobleme wären sonst die Folge). Zudem werden gültige Prozess-Standards durchgesetzt und gelehrt.

Agile Inspektionen laufen nach den folgenden allgemeinen Prinzipien ab:
- Wenige Seiten auf einmal werden geprüft, beispielsweise ein bis drei Seiten.
- Eventuell wird sehr frühzeitig geprüft, zum Beispiel schon wenn die ersten 5 % eines großen Dokuments fertig sind.
- Es wird kontinuierlich geprüft (z.B. jede Woche), bis die Arbeit fertig ist.
- Die Dokumente jedes Entwicklers werden geprüft, denn jeder einzelne Autor muss persönlich motiviert und trainiert werden.

Ablauf einer Agilen Inspektion

Eine Stichprobe wird ausgewählt (z.B. eine Seite) und gegen ca. drei bis sieben Regeln geprüft. Beispiele für Regeln: 1. Clarity („clear enough to test“), 2. Unambiguous („to the intended readership“), 3. Consistent („with other statements in the same or related documents“). Die Reviewer sollen alle Abweichungen von diesen Regeln identifizieren und nach Major oder Minor Defect klassifizieren. Die Major Defects werden an den Moderator berichtet.

An der Prüfsitzung nehmen beispielsweise zwei Reviewer teil, also eher weniger als bei einer klassischen Inspektion. Die Sitzung dauert ca. 30-60 Minuten. Geprüft wird mit optimaler Inspektionsrate, typischerweise eine bis höchstens zwei Seiten pro Stunde. Ein trainierter Moderator ist anwesend und leitet den Prozess.

Nach der Prüfung wird die geschätzte Anzahl der tatsächlich vorhandenen Fehler aus der Gesamtzahl der gefundenen Fehler berechnet. (Berechnungsgrundlage: typischerweise findet das Team in diesem Setting ein Drittel der vorhandenen Fehler.) Als Exit-Kriterium der Inspektion wird anfangs ein Wert wie beispielsweise „maximal 10 Major Defects pro Seite" angesetzt. Nach ein paar Monaten „Kulturänderung" sollte man das Limit eher bei „1 Major Defect pro Seite" ansetzen.

Abhängig von der Fehlerdichte legt das Reviewteam weitere Maßnahmen fest. Wenn die Fehlerdichte sehr hoch ist (z.B. 10 Major Defects pro Seite oder mehr), wäre es unökonomisch, die anderen Seiten des Dokuments zu prüfen, um „alle Fehler" zu finden. Und es würde auch nicht viel nützen, die bisher gefundenen Fehler zu korrigieren. Es würden trotzdem zu viele Major Defects unentdeckt bleiben. Die beste Alternative ist, das Dokument durch den Autor oder jemand anderen neu schreiben zu lassen. (Hier sieht man, dass eine frühzeitige Agile Inspektion Sinn macht, beispielsweise wenn 5 % des Dokuments fertig sind.)

Bemerkungen

Eine Eigenheit der Agilen Inspektionen ist, dass im Gegensatz zu klassischen Inspektionen das Dokument nicht gegen die Vorgängerdokumente geprüft wird, sondern nur gegen das Wissen der Reviewer. Dadurch wird die Prüfung beschleunigt, allerdings ist das Ergebnis ungenauer.

Gilb vollzieht mit den Agilen Inspektionen einen Paradigmenwechsel, der nicht nur für Qualitätssicherer sehr überraschend ist. Die unmittelbare Lösung gegen hohe Fehlerdichten ist nicht, die gefundenen Fehler aus dem Dokument zu entfernen, und ist nicht, den Softwareentwicklungsprozess zu ändern! Die effektivste praktikable Lösung ist: Sicherstellen, dass jeder Autor das Exit-Kriterium der maximalen Fehlerdichte ernst nimmt. Das ist erreichbar, denn im Durchschnitt sollte nach jeder Agilen Inspektion die Fehlerrate eines Autors um ca. 50 % sinken.

4. Wie agil sind Agile Inspektionen?

Da Agile Inspektionen besonders in Wasserfall-Projekten nützlich sind, ist das Wort „agil" in dieser Hinsicht irreführend. Berechtigt ist das Wort „agil" aber aus zwei Gründen. Erstens sind Agile Inspektionen viel leichtgewichtiger als klassische Inspektionen. Der Prozess ist einfacher und Agile Inspektionen erreichen eine größere Qualitätsverbesserung mit deutlich weniger Aufwand als klassische Inspektionen. Zweitens sind mit Agilen Inspektionen viele kurze Feedbackschleifen im Projekt möglich. Das entspricht voll dem Grundgedanken der agilen Vorgehensmodelle.

Literatur

1. Gilb, Tom: Agile Specification Quality Control. Cutter IT Journal, Vol. 18, No. 1, pp. 35-39, January 2005
2. Gilb, Tom / Graham, Dorothy: Software Inspection, Addison-Wesley, 1993
3. Radice, Ronald A.: High Quality Low Cost Software Inspections, Paradoxicon Publishing, 2002
4. www.gilb.com (Download Center, Stand 23.09.2005), “Optimizing Inspection” von Tom Gilb, S. 2
5. Gilb, Tom: Competitive Engineering, Butterworth-Heinemann, 2005
6. Foliensatz zu dieser Textfassung: www.reviewtechnik.de/vortraege.html (Stand 09.12.2010)

Autor:

Dipl.-Inform. Peter Rösler ist trainierter Reviewmoderator für Gilb-Inspektionen und arbeitete als Programmierer, QS-Beauftragter und Projektleiter in verschiedensten Softwareentwicklungsprojekten, vorwiegend im Bereich Airports/Airlines bei der Firma Softlab GmbH in München. Seit 2005 arbeitet er als freiberuflicher Reviewtechnik-Trainer und ist Mitgründer und Partner bei ReviewConsult.