Dieser Text erscheint in leicht geänderter Form als Gastbeitrag in Dr. Ernest Wallmüllers Buch
„Software Quality Engineering - Ein Leitfaden für bessere Software-Qualität“ (Hanser, 1. Quartal 2010)

Populäre Irrtümer und Fehleinschätzungen in der Reviewtechnik

Peter Rösler

Die Grundidee von Reviews ist leicht verständlich: Ein Team von Reviewern prüft ein Dokument, z.B. ein Fachkonzept oder ein Programm, und findet damit Fehler, die ansonsten erst im Test oder beim Kunden entdeckt worden wären. In der Reviewtechnik gibt es allerdings einige Sachverhalte, die sich ganz anders darstellen, als es Projektleiter oder Software-Entwickler intuitiv vermuten würden. Ein paar dieser populären Irrtümer und Fehleinschätzungen sollen hier behandelt werden.

Definitionen

Major Defect:	Fehler, der später im Projekt erheblich höhere Kosten verursachen würde, wenn er nicht jetzt im Review entdeckt worden wäre (z.B. Fehler, durch die ein Testfall scheitern würde)
Minor Defect:	alles andere (z.B. Rechtschreibfehler)
Effektivität:	Anzahl der gefundenen Major Defects / Anzahl der im Dokument vorhandenen Major Defects
Effizienz:	Anzahl der gefundenen Major Defects pro Stunde

Einführung

Die folgenden Bemerkungen gelten nicht für Walkthroughs, in denen der Autor dem Reviewteam das Dokument vorstellt (unterrichtendes Review), und gelten auch nicht für Management- oder Projektstatus-Reviews. Die Bemerkungen gelten nur für Reviews, deren Hauptzweck es ist, Fehler zu finden. Diese Art von Review, oft auch "Software-Inspektion" genannt, läuft laut Gilb/Graham [Gilb93] nach folgenden Phasen ab:

Planning & Entry	In der Planungsphase wird u.a. ein Entry-Check durchgeführt, bei dem z.B. geprüft wird, ob sich das Dokument überhaupt in einem reviewbarem Zustand befindet.
Kickoff	Im (optionalen) Kickoff-Meeting gibt der Autor den Reviewern so viel Hintergrundinformation zum Dokument und zum Projektumfeld, dass jeder Reviewer auch sinnvoll Fehler finden kann.
Checking	In der Checking-Phase, oft auch "Vorbereitung" genannt, sucht jeder Reviewer nach Fehlern im Dokument.
Logging	Im Logging-Meeting, oft auch "Review-Sitzung" genannt, werden die Fehler zu Protokoll gegeben und Fehlalarme aussortiert.
Process Brainstorming	Im Process-Brainstorming-Meeting wird nachgedacht, warum die Fehler gemacht wurden und durch welche Verbesserungen im Softwareentwicklungsprozess diese Art von Fehlern zukünftig vermieden werden kann.
Edit	In der Edit-Phase korrigiert der Autor die gefundenen Fehler.
Followup	In der Followup-Phase prüft der Moderator, ob der Autor alle Fehler behoben hat.
Exit	In der Exit-Phase wird entschieden, ob das Dokument freigegeben werden kann oder ob ein wiederholtes Review nötig ist.

Software-Reviews wurden schon 1976 von Michael Fagan vorgestellt [Faga76]. Seitdem haben zahlreiche Untersuchungen den Nutzen von Reviews belegt. Dieser Nutzen wird aber nicht oder nur teilweise erzielt, wenn man folgende Irrtümer und Fehleinschätzungen nicht kennt.

Irrtum 1 (Review-Sitzung)

Ein verbreiteter Irrtum ist, dass die Fehler in der Review-Sitzung entdeckt werden. Tatsache ist jedoch, dass meist über 95% der Fehler, die vom Reviewteam insgesamt entdeckt werden, schon in der Checking-Phase gefunden werden. Die Effektivität eines Reviews hängt also entscheidend von der Gründlichkeit der Checking-Phase ab. Eine der wichtigsten Aufgaben des Moderators ist es folglich, die Reviewer zu motivieren, gründlich zu prüfen. Die Review-Sitzung zu leiten ist im Vergleich dazu nur eine nachrangige Aufgabe des Moderators.

Irrtum 2 (Kosten)

Dieser Irrtum besteht in der Ansicht, dass Reviews Kosten verursachen und man als Ausgleich dafür bessere Qualität bekommt. Man gibt also Zeit (= Kosten) und bekommt Qualität. Richtig dagegen ist, dass man Qualität bekommt *und* zusätzlich Zeit einspart. Denn nicht nur der Kunde profitiert von der besseren Qualität der Software, sondern auch schon das Projektteam selbst spürt die bessere Qualität der Software, nämlich in den Testphasen. Die Testphasen sind viel schneller fertig als üblich, weil weniger Major Defects zu korrigieren sind. Die Erfahrung zeigt, dass man mit einer Stunde Reviewaufwand ca. 4-8 Stunden Testaufwand einsparen kann. Wie viel Projektzeit man insgesamt durch Reviews wegoptimieren könnte, zeigt Abbildung 1:

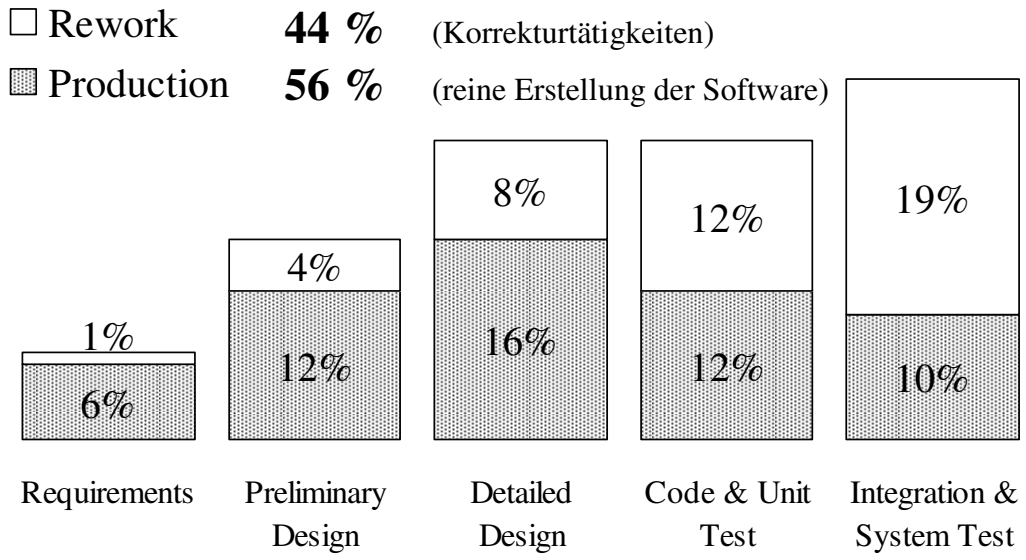

Abb. 1: Anteil von Korrekturtätigkeiten am Gesamtaufwand (nach [Whee96, p9]).

Ca. 44% des Aufwands eines typischen Software-Projekts sind Korrekturtätigkeiten, also Tätigkeiten, die nur nötig sind, weil die Dokumente, die in den vorangegangenen Projektphasen erstellt wurden, Major Defects enthielten. Diese 44% könnte man also im Prinzip einsparen, wenn die Effektivität der Reviews 100% wäre. Leider sind Reviewteams im wirklichen Leben nicht ganz so perfekt. Mit einer realistischen Annahme von 50%

Effektivität können hier aber immerhin noch 22% Projektaufwand eingespart werden. Das Kostensenkungspotential durch Reviews ist viel höher als allgemein vermutet, und daher bezeichnen viele Fachleute die Reviewtechnik als "eine der am meisten unterschätzten Schlüsseltechnologien in der System- und Software-Entwicklung".

Irrtum 3 (Effizienz)

Diese Fehlannahme besteht darin, die gute alte 80/20-Regel auf die Checking-Phase anzuwenden. Nach dem Motto: "Es mag zwar 100 Minuten dauern, dieses Dokument gründlich zu prüfen, und ca. 100 Fehler sind dadurch auffindbar. Aber lieber Reviewer, prüfe bitte nur 20 Minuten lang, denn dadurch kannst du schon 80 dieser 100 Fehler finden." Diese Annahme ist vermutlich falsch. In der Checking-Phase scheint es eine konstante Fehlerentdeckungsrate zu geben, wie die Arbeiten von Dunsmore/Roper/Wood [Duns03], Górski/Jarzębowicz [Gors05] und Rösler [Rösl05] nahe legen. Im obigen Beispiel wird man nach 20 Minuten erst 20 Fehler, nach 40 Minuten 40 Fehler, nach 60 Minuten 60 Fehler finden, etc. Während der vollen Prüfzeit bleibt die Effizienz konstant. Es ist einem Reviewer in der Checking-Phase nach heutigem Erkenntnisstand nicht möglich, eine bessere Effizienz durch Zugeständnisse bei der Effektivität zu erreichen. Schlampig prüfen lohnt sich also auch hier nicht.

Irrtum 4 (Prüfdauer)

Eine der folgenreichsten Fehleinschätzungen betrifft die Zeitdauer, die in der Checking-Phase nötig ist, um ein Textdokument gründlich zu prüfen. (Für Code-Reviews gilt diese Fehleinschätzung nicht.) Bei Textdokumenten jedoch versagt die menschliche Intuition komplett. Die meisten Software-Entwickler würden vermuten, dass sie in einer Stunde durchaus 5, 10 oder 20 Seiten gründlich prüfen können. Eine Umfrage unter Software-Entwicklern ergab ein dazu passendes Bild, nämlich dass in ihren Firmen durchschnittlich 10 Seiten pro Stunde geprüft werden [Rösl05]. Das ist aber 10 Mal schneller, als die von Fachleuten empfohlene "optimale Inspektionsrate" für Textdokumente. Diese beträgt nach Gilb/Graham [Gilb93] nur ca. eine (!) Seite pro Stunde (Bandbreite: 1 ± 0,8 Seiten pro Stunde [Gilb05]). Da in den Firmen im Durchschnitt 10 mal schneller geprüft wird, hat das zur Folge, dass die Reviewteams nur ein Zehntel so viele Major Defects finden, wie bei optimaler Prüfzeit zu erwarten wäre (wegen der konstanten Fehlerentdeckungsrate, siehe Irrtum 3, darf man abgesehen von einem gewissen Doubletteneffekt linear herunterrechnen). Statt einer Effektivität von ca. 50%, die erfahrungsgemäß ein Reviewteam mit optimaler Prüfzeit erreicht, wird also in den meisten realen Projekten nur eine Effektivität von ca. 5% erreicht. Mit anderen Worten, 95% der Fehler stecken in den Dokumenten noch unentdeckt drin. Die Chancen, dass das Projekt erfolgreich zu Ende geführt werden kann, haben sich durch diese Art von Review nur marginal verbessert. Die Empfehlung lautet daher: Der firmeneigene Review-Prozess muss gewährleisten, dass die Moderatoren die optimale Inspektionsrate für Textdokumente kennen, und die Moderatoren sollten es zu einer ihrer Hauptaufgaben machen, dafür zu sorgen, dass die Reviewer die optimale Prüfzeit auch tatsächlich einsetzen.

Schlussbemerkung

Reviews sind ein besonders nützliches Werkzeug aus dem großen Werkzeugkasten der Software-Qualitätssicherung. Den vollen Nutzen kann man aus ihnen aber nur ziehen, wenn man sie nicht rein intuitiv einsetzt, sondern zu einem ingenieurmäßigen, von Zahlen und Fakten gesteuerten Einsatz übergeht.

Dipl.-Inform. Peter Rösler (www.reviewtechnik.de) arbeitet als freiberuflicher Reviewtechnik-Trainer und -Berater. Er ist trainierter Reviewmoderator für Gilb-Inspektionen und bei mehreren Schulungsanbietern Dozent für Review-Seminare.

Literatur

[Duns03] *Dunsmore, A., Roper, M., Wood, M.:* The Development and Evaluation of Three Diverse Techniques for Object-Oriented Code Inspection. IEEE Trans. Software Eng. 29(8), 2003, pp. 677-686

[Faga76] *Fagan, M. E.:* Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal 15(3), 1976, pp. 182-211

[Gilb93] *Gilb, T., Graham, D.:* Software Inspection, Addison-Wesley, 1993

[Gilb05] *Gilb, T.:* www.gilb.com (Download Center, Stand 23.09.2005), "Optimizing Inspection"

[Gors05] *Górski, J., Jarzębowicz A.:* Development and validation of a HAZOP-based inspection of UML models, in: Proceedings of the 3rd World Congress for Software Quality (Vol. I), Erlangen, 2005, pp. I-345 - I-354

[Rösl05] *Rösler, P.:* Warum Prüfen oft 50 mal länger dauert als Lesen und andere Überraschungen aus der Welt der Software-Reviews. Softwaretechnik-Trends 25(4), November 2005, pp. 41-44

[Whee96] *Wheeler, D.A., Brykczynski, B., Meeson R.N.:* Software Inspection: an industry best practice, IEEE Computer Society Press, 1996