

Design- und Code-Reviews für Embedded Software

Übersicht

1. Motivation, Reviewarten, Begriffe
2. Wasserfallmodell und klassische Reviews
3. Agile Inspektionen
(auch für Wasserfallprojekte)
4. Folgerungen für Tester



Dipl.-Inform. Peter Rösler

Stephan Grünfelder:

(Embedded Testing Konferenz 2015, Keynote-Vortrag „Test von Embedded Software“)

- Code-Reviews bei Embedded Software wichtiger als für andere Software
- Auch Design ist bezüglich Wartbarkeit und Sicherheit zu inspizieren

Im Folgenden geht es darum, wie man Design- und Code-Reviews so durchführt, dass möglichst viele Fehler gefunden werden.

Reviewarten (1)

3

Reviewart	Reviewobjekt	Hauptzweck
Informelles Review Desk-Check, 4 Eye Check, Passaround	Ein einzelnes Dokument typischerweise	Fehler finden
Walkthrough	Ein einzelnes Dokument typischerweise	Fehler finden
Inspektion Review* , Peer-Review, Fagan-/Gilb-Inspektion	Ein einzelnes Dokument typischerweise	Fehler finden

* Die „Inspektion“ wird oft „Review“ genannt. Daher muss beim Wort „Review“ immer aus Zusammenhang erschlossen werden, ob Überbegriff „Review“ oder Reviewart „Review/Inspektion“ gemeint ist.

Reviewarten (2)

4

Reviewart	Reviewobjekt	Hauptzweck
Agile Inspektion Extreme Inspection, Agile Specification Quality Control	Stichprobe aus einem Dokument typischerweise 1 - 3 Seiten	Lernkurve des Autors fördern Um zukünftige Fehlerrate zu senken
Technisches Review Expertenreview	Ein einzelnes Dokument oder ein Softwareprodukt	Eignung bewerten, Fehler finden
Management-Review Projekt-Status-Review, Meilensteinreview	Projekt als Ganzes	Projektfortschritt bewerten
Audit	Prozesse des Projekts oder der Organisations- einheit	Prozesseinhaltung bewerten

“major defect” (im Gegensatz zu “minor defect”)

- Fehler, der möglicherweise erheblich höhere Kosten verursacht, wenn er später gefunden wird als jetzt

“Inspektionsrate“

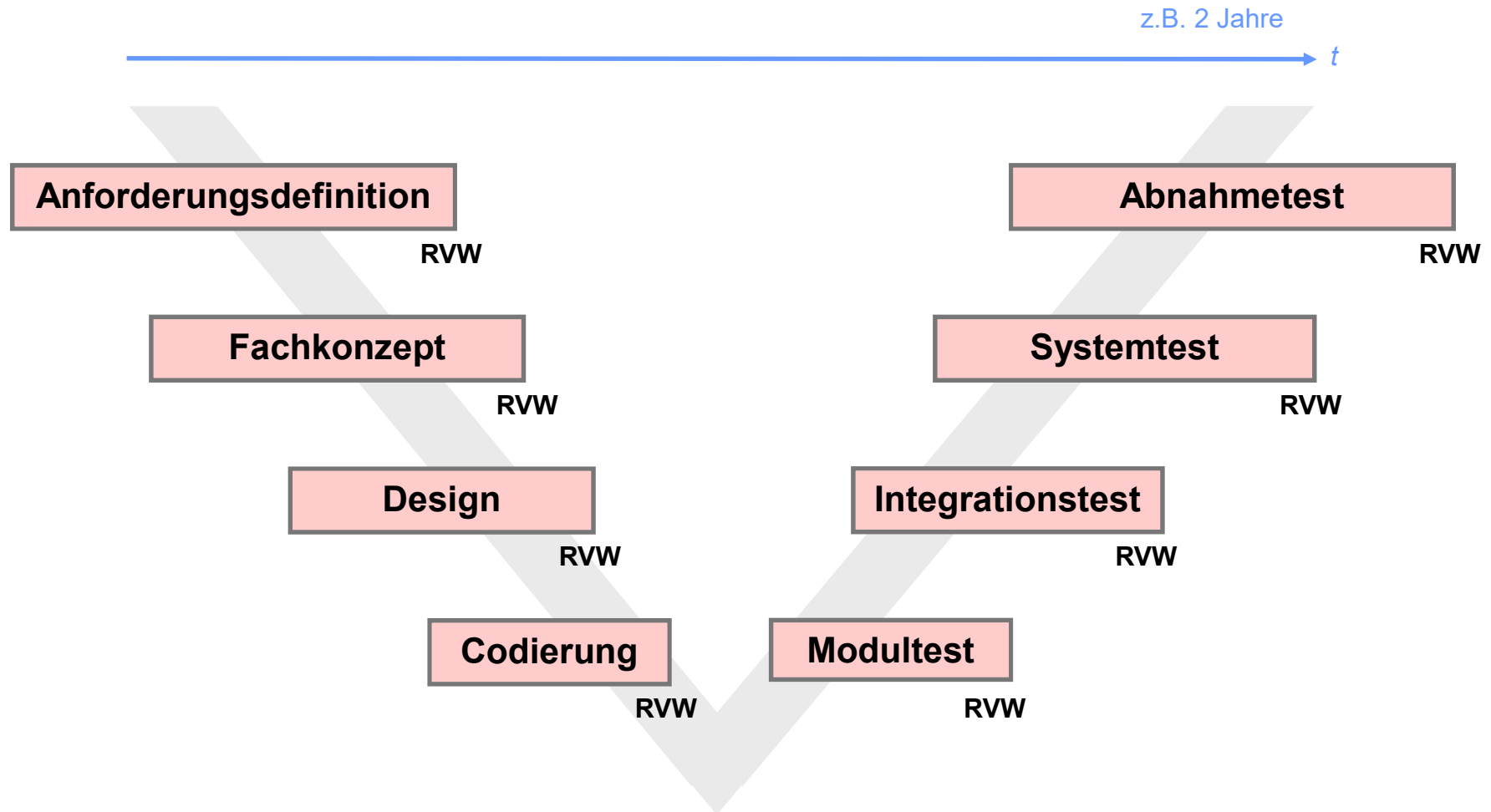
- Prüfgeschwindigkeit, wird bei Programmen in NLOC / h und bei Textdokumenten in Seiten / h angegeben.

$$\text{Effektivität} = \frac{\text{gefundene Mj}}{\text{alle vorhandenen Mj}}$$

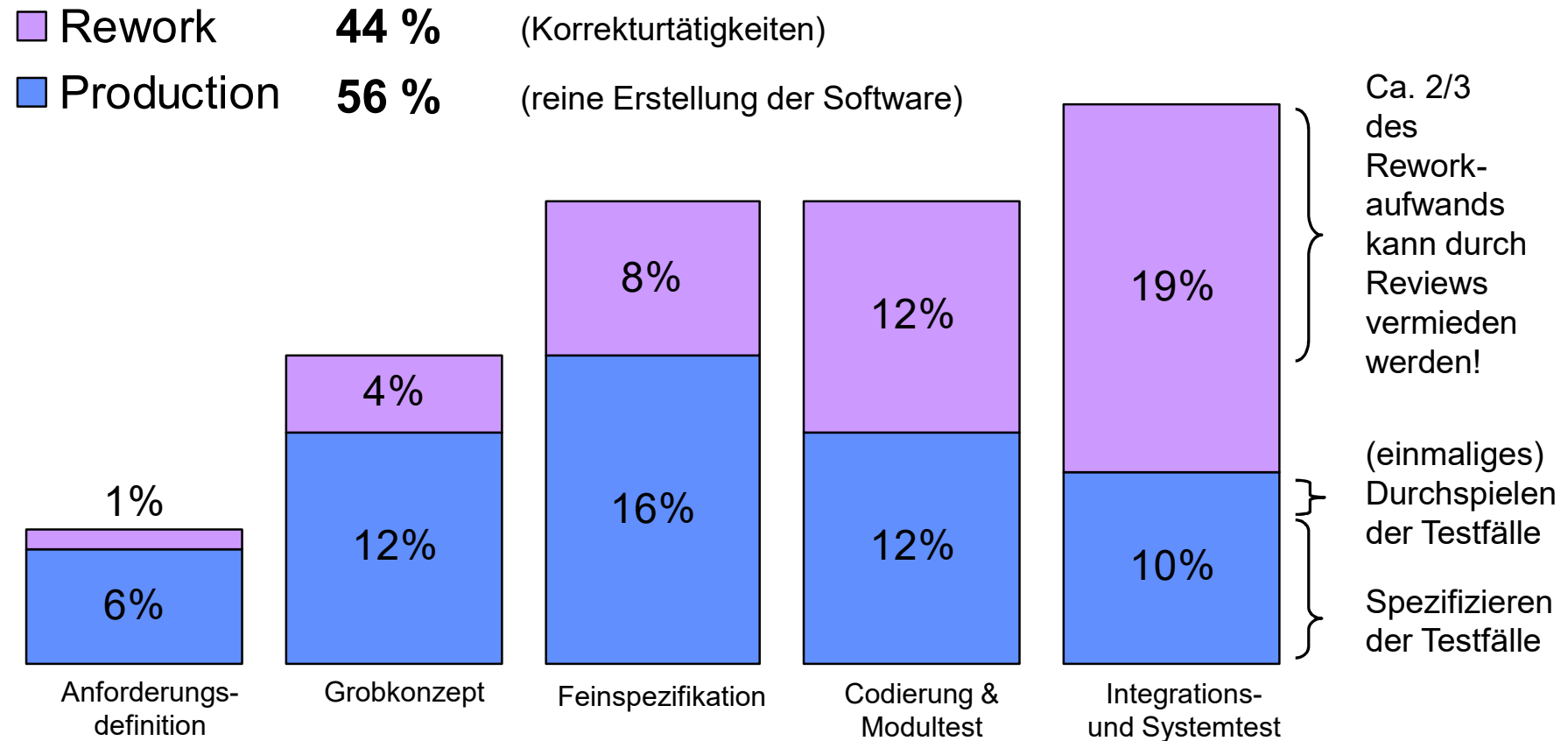
Effektivität eines Review-Teams liegt irgendwo zwischen 3% und >90% (!)

Typisches V-Modell

7



Anteil von Korrekturtätigkeiten am Gesamtaufwand

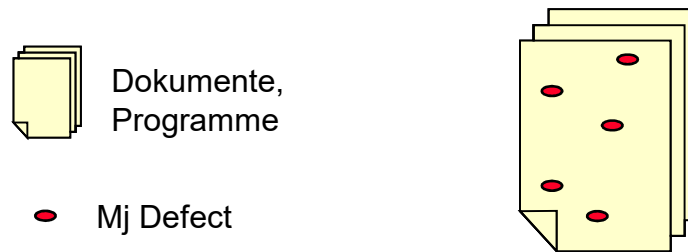


- Rework sind alle Arbeiten, die aus Major Defects resultieren
 - Lokalisieren von gefundenen Fehlern
 - Korrigieren der Dokumente
 - Ändern von Testspezifikationen
 - Wiederholen von Testläufen, etc.
- **Reworkaufwand** per Definition (zumindest annähernd) proportional zu **Anzahl der Major Defects** in den Dokumenten, also zu **Fehlerdichte** der Dokumente.

Die Fehlerdichte steuert den Reworkaufwand

10

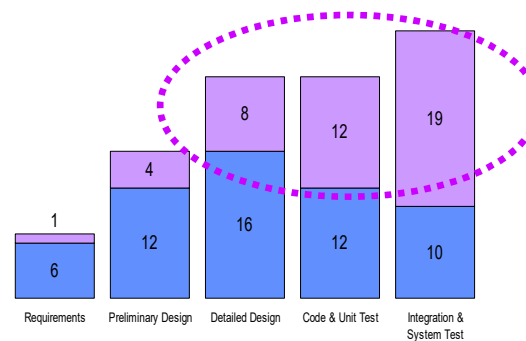
typisches Projekt



Die Anzahl der Major Defects entscheidet ...

Rework

Production

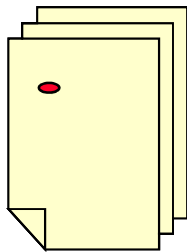


... wie hoch der Reworkaufwand ist.

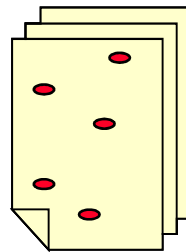
Die Fehlerdichte ist stark variabel ...

11

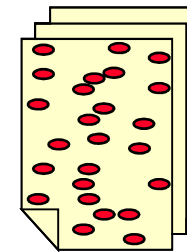
Projekt A,
5 mal weniger Mj Defects



typisches Projekt



Projekt C,
5 mal mehr Mj defects



← 3 Projekte mit um Faktor 25 unterschiedlichen Fehlerdichten →

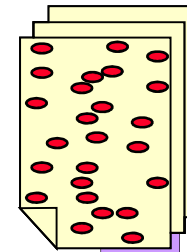
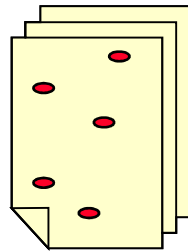
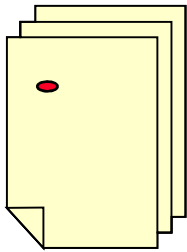
... die Folgen für den Aufwand sind enorm

12

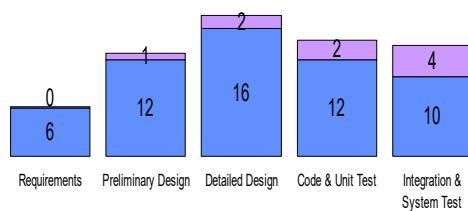
5 mal weniger Mj Defects

typisches Projekt

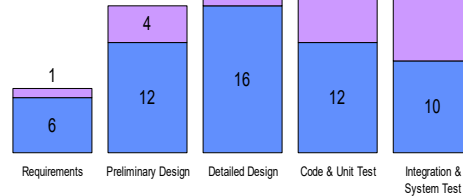
5 mal mehr Mj Defects



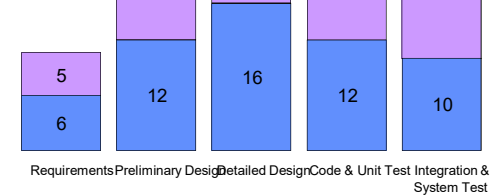
■ Rework
■ Production



65 Wo



100 Wo



276 Wo

Gesamtaufwand in Wochen

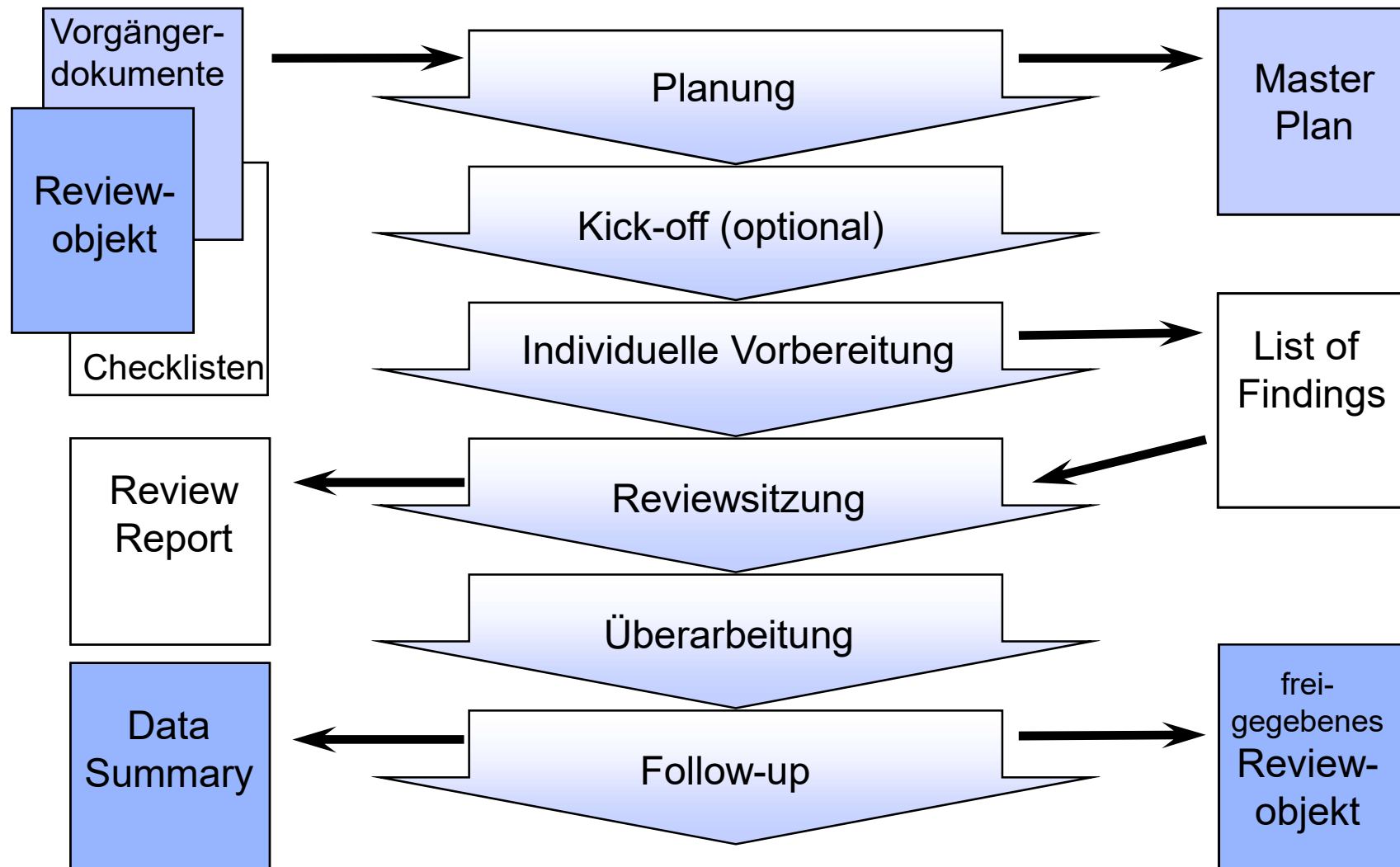


- Moderator
- Autor
- Protokollführer
- Reviewer

Ein Teilnehmer kann mehrere Rollen übernehmen.
Einzige Einschränkung: der Autor darf zusätzlich höchstens die Rolle eines Reviewers übernehmen.

Phasen eines Reviews

14



- potentielle “major defects” finden
- und notieren
- optimale Checking Rate einhalten
- Lesetechniken einsetzen (z.B. Checklisten)



Fast alle Fehler (ca. 80-99%),
die das Reviewteam entdecken kann,
werden schon in dieser Phase gefunden!

Programme

- 100 – 150 NLOC / h 100 - 150 LOC/h

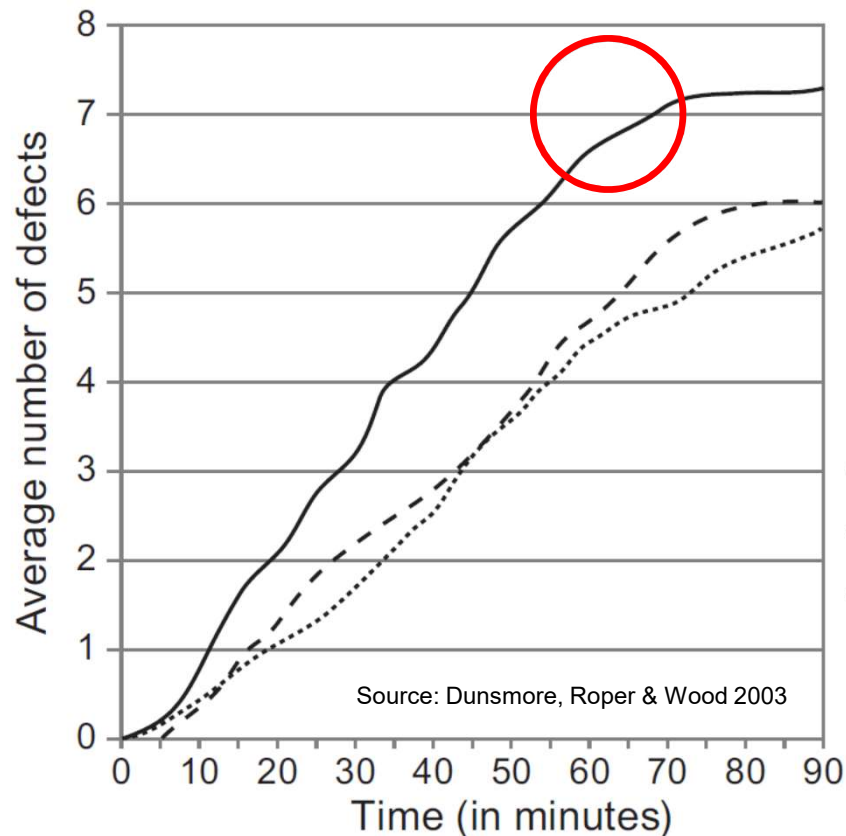


Textdokumente

- Gilb/Graham: ca. 1 Seite / h 1 ± 0.8 , “typically 0.3 - 1.0“, “avoid rates above 2 p/h“
- Strauss/Ebenau: 3 – 5 Seiten / h
- IEEE Std 1028-2008: 2 – 3 Seiten / h für Architektur- und Requirements-Dokumente

Fehlerfinderate flacht ab, sobald Reviewer alle seine Prüfstrategien ausgeschöpft hat

17



Die Effizienz geht rasch auf fast Null zurück.

Effizienz = gefundene Mj / h

Die optimale Inspektionsrate von Gilb/Graham sagt die Stelle des „Knicks“ voraus.

(Behauptung, P.Rösler 2011)

Effektivität eines Reviews

18

„Daumenwert“ : **ca. 50%** der im Dokument vorhandenen Fehler werden durch das Reviewteam entdeckt, sofern die optimale Inspektionsrate eingehalten wird.

Angaben von
Michael Fagan:
("Erfinder" der
SW-Inspektionen)

Maturity	Effectiveness
inspections just implemented	50-60%
refined processes	75-85%
refined processes, exceptional cases	>90%

Source: Fagan 2001

Effektivität eines Reviews (2)

19

Wenn die optimale Inspektionsrate ignoriert wird:

immature inspection process	Effectiveness
the normal for inspections not optimized with optimal checking rates etc.	about 3%
Source: Tom Gilb / Kai Gilb 2004	

**Klingt viel zu pessimistisch.
Stimmt aber!
(Zumindest für Textdokumente)**

- geringere Entwicklungskosten (25-35%)
- kürzere Entwicklungszeiten (25-35%)
- geringere Wartungskosten (Faktor 10-30)
- höhere Zuverlässigkeit (10-100 mal weniger Fehler)



- Agile Inspektionen:
vorgestellt 2005 von Tom Gilb



Tom Gilb
www.gilb.com

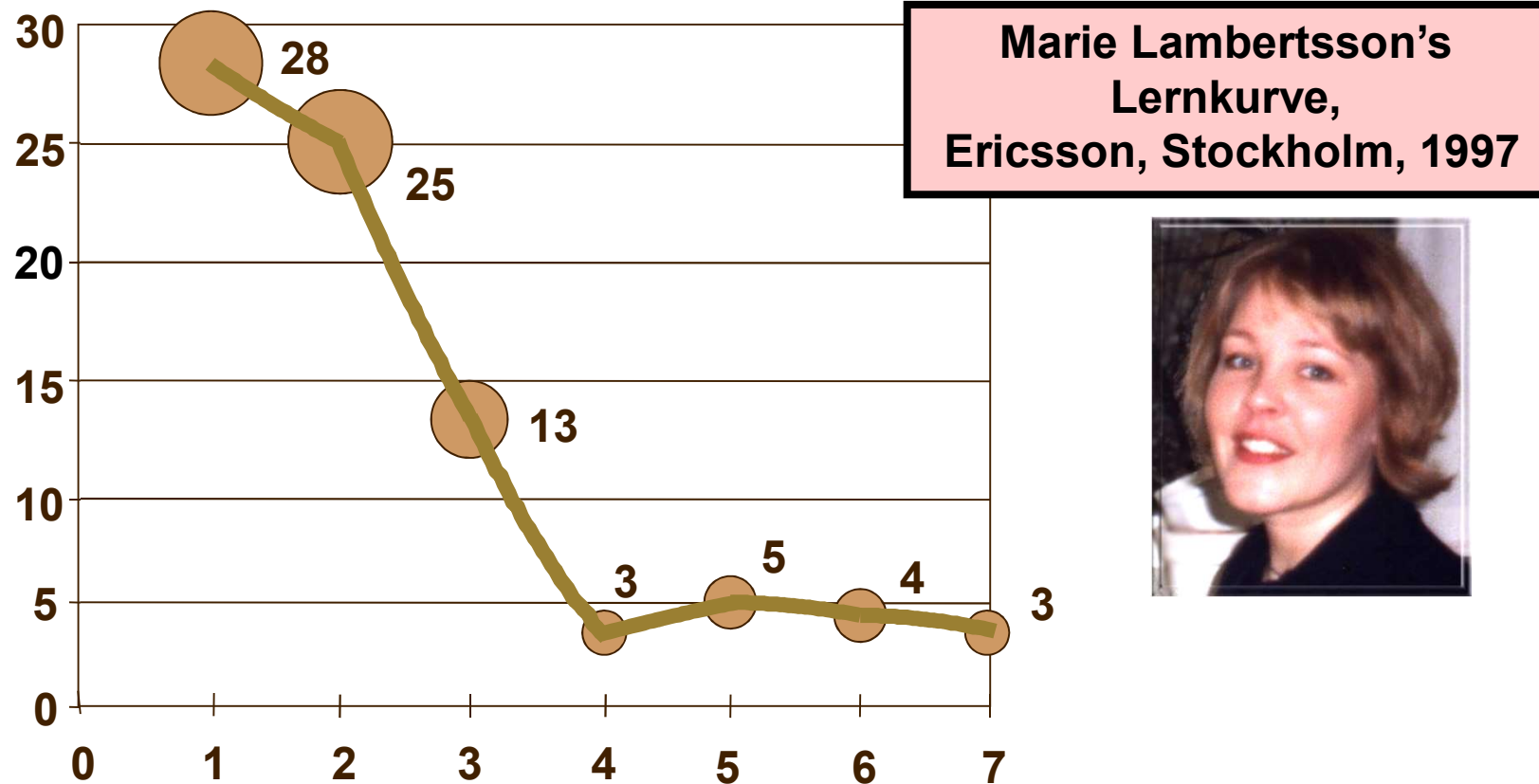
- Agile Inspektionen **besonders in V-Modell-Projekten nützlich**, weil dort keine kurzen Feedback-Schleifen (z.B. in Form von Iterationen)

Bei *klassischen* Inspektionen:

- Autoren beginnen **manchmal** aufgrund Review-Erfahrungen, deutlich fehlerfreier zu arbeiten.
- Signifikante **Lernkurve** kann erreicht werden.

Marie's persönliche Lernkurve

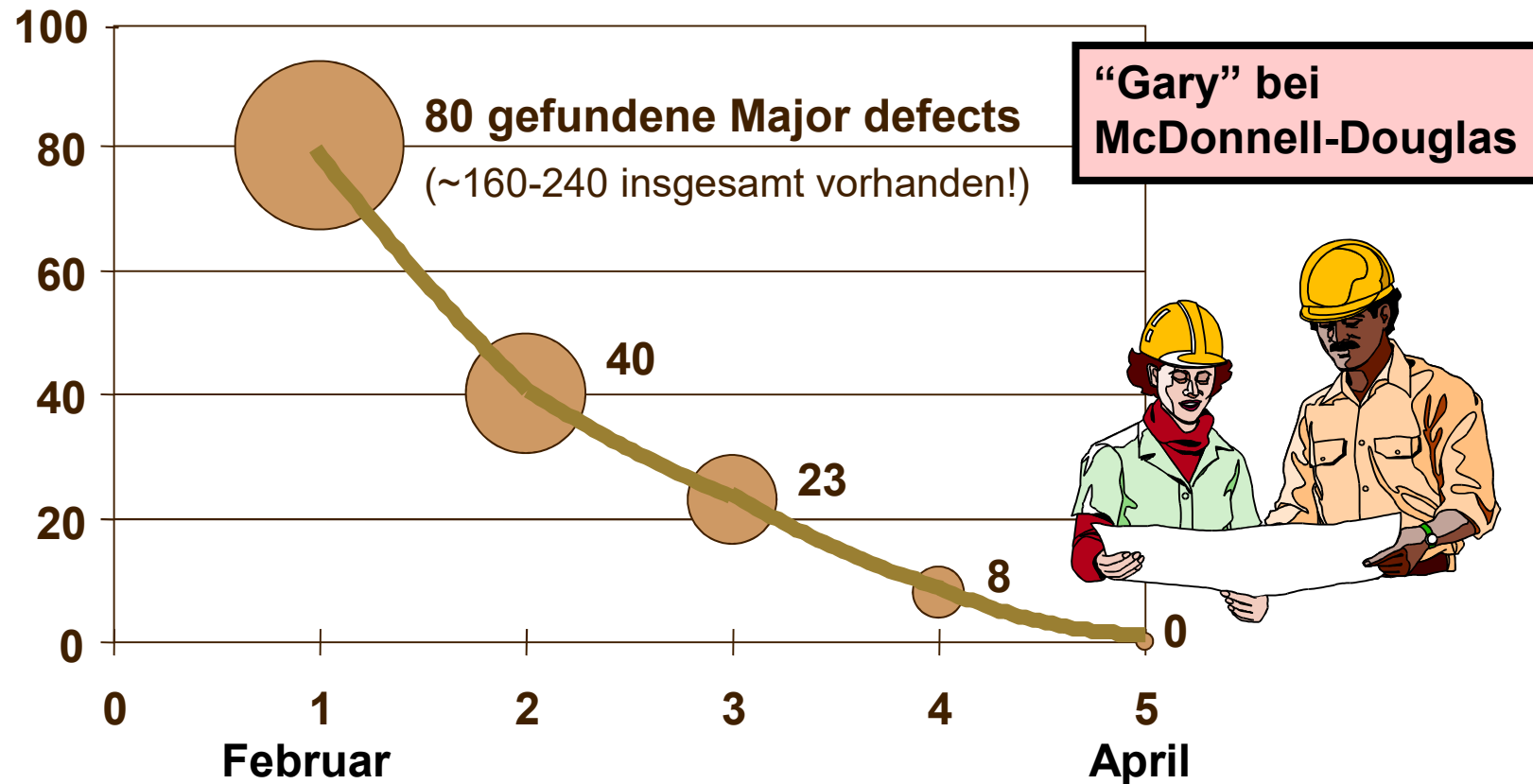
Geschätzte verbleibende Major defects / Seite



Reihenfolge der Dokumente, die einem Review unterzogen wurden

Gary's persönliche Lernkurve

Gefundene Major defects / Seite



Reviews von Gary's Design-Dokumenten

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.

Kosten für Inspektionen sinken enorm:

- Stichproben reichen aus (statt 100% der Dokumente zu prüfen), denn Zweck ist jetzt „Messen“ statt „cleanup“-Modus.

Allgemeine Prinzipien von Agilen Inspektionen

- **Wenige** Seiten auf einmal (z.B. 1 - 3 Seiten)
- Evtl. **frühzeitig** (erste 5% eines großen Dokuments)
- **Kontinuierlich** (z.B. jede Woche), bis Arbeit fertig ist
- Für jeden Entwickler (**jeder** einzelne Autor muss persönlich motiviert und trainiert werden)

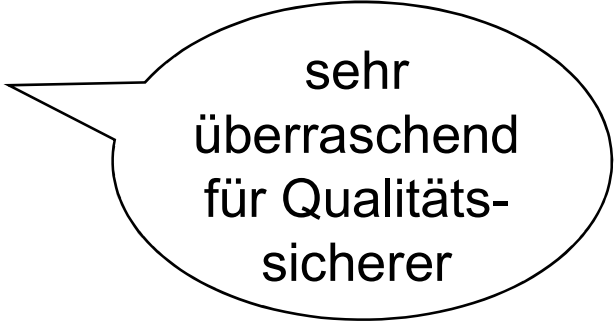
- **Stichprobe** wird ausgewählt (z.B. 1 Seite) und gegen ca. 3 - 7 Regeln geprüft, z.B.
 - Clarity („clear enough to test“)
 - Unambiguous („to the intended readership“)
 - Consistent („with other statements in the same or related documents“)
 - Completeness („compared to sources“)
 - Für Anforderungsdokumente: „no design“
- Reviewer sollen alle Regelabweichungen identifizieren und klassifizieren. Die Mj Defects werden an Moderator berichtet.

- An Prüfsitzung nehmen beispielsweise zwei Reviewer teil, Dauer ca. **30-60 Minuten**. Geprüft wird mit optimaler Inspektionsrate. Trainierter Moderator ist anwesend und leitet den Prozess.
- Danach wird geschätzte **Anzahl** der tatsächlich **vorhandenen Fehler** aus Gesamtzahl der gefundenen Fehler berechnet.
(Berechnungsgrundlage: typischerweise findet Team in diesem Setting ein Drittel der vorhandenen Fehler.)

- **Maßnahmen** werden festgelegt
 - Bei sehr hoher Fehlerdichte (z.B. 10 Mj / p oder mehr):
Unökonomisch, die anderen Seiten zu prüfen, um „alle Fehler“ zu finden, oder die bisher gefundenen Fehler zu korrigieren. Zu viele Mj Defects bleiben trotzdem unentdeckt.
 - Beste Alternative: Autor oder jemand anderes schreibt Dokument **neu**. (Hier sieht man, dass eine frühzeitige Agile Inspektion sinnvoll ist, z.B. schon wenn 5% des Dokuments fertig sind.)

Unmittelbare Lösung gegen hohe Fehlerdichten

- ist **nicht**, die gefundenen Fehler aus dem Dokument zu entfernen,
- und ist **nicht**, den SW-Entwicklungs-Prozess zu ändern!



sehr
überraschend
für Qualitäts-
sicherer

Die effektivste praktikable Lösung:

- Sicherstellen, dass jeder Autor das **Ausgangskriterium der maximalen Fehlerdichte** ernst nimmt.

- Hypothese: Für Tester wird es immer wichtiger, Kompetenzen zu Reviews und Inspektionen aufzubauen.
(Zeigt sich auch in den Lehrplänen von ISTQB Certified Tester.)
- Zwei Reviewverfahren stehen zur Verfügung:
 - klassische Reviews (etabliert in SW-Industrie)
 - agile Inspektionen (kaum Industrieerfahrung vorhanden)
- Tester können beitragen als
 - Reviewmoderatoren
 - Reviewteilnehmer (s. nä. Folie)

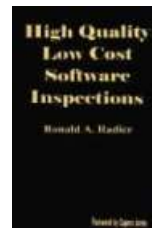
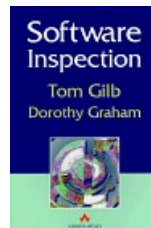
Tester als Reviewteilnehmer

<i>Employee Type</i>	<i>Contract</i>	<i>Reqs.</i>	<i>Archit. Design</i>	<i>Detail Design</i>	<i>Test Plan</i>	<i>Test Design</i>	<i>Source Code</i>	<i>Tech. Doc.</i>	<i>User Manual</i>
Task author	x	x	x	x	x	x	x	x	x
Req. analyst	x	x	x	x	x	x			x
Arch. designer	x	x	x	x	x	x		x	
Detail. designer			x	x	x	x	x	x	
Programmer				x	x	x	x	x	
Tester	x	x	x	x	x	x	x	x	x
Maintainer		x	x	x	x	x	x	x	x
User	x	x	x		x	x			x
Manager	x	x	x		x				x
Marketing	x	x	x						x
Legal department	x	x						x	x

- Bekannt: Kaum möglich, Qualität in ein Softwaresystem hineinzutesten
- Bekannt: Hilfe oft zu spät, wenn Tester ins Projekt erst während Testphasen einsteigen
- Es tut dem Projekt gut, wenn Tester als „Dokumenten-Tester“ früh im Projekt aktiv sind!*
- Am besten schon zum Review der Anforderungen („Sind die Anforderungen testbar formuliert?“)

Literatur zu Reviews und Inspektionen

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. Rösler, P.; Schlich, M.; Kneuper, R.: Reviews in der System- und Softwareentwicklung: Grundlagen, Praxis, kontinuierliche Verbesserung. dpunkt.verlag, Heidelberg, 2013





Das Gummibärchenbild enthält fünf absichtlich eingebaute "Fehler" bzw. Auffälligkeiten. Können Sie mindestens vier dieser Fehler finden?

Peter Rösler · Maud Schlich · Ralf Kneuper

Reviews

in der System- und