Software-Qualitätssicherung und -Prüfung; Reviews

Jochen Ludewig, Universität Stuttgart

Vorrede

Meinen Vorlesungen über Software Engineering an der Universität Stuttgart liegt das Buch „Software Engineering“ von Ludewig, Lichter (2007) zu Grunde.
Thema heute ist das Kapitel 13 („Software-Qualitätssicherung und Prüfung“). Folien und Abschnitte sind entsprechend nummeriert. In meiner Vorlesung wären das **zwei bis drei Doppelstunden**.

Übrigens steht genau heute in Stuttgart in der Vorlesung „GSE“ die **Review-Übung** auf dem Programm.

Was ist Qualitätssicherung?

Allgemeine Erfahrung in der fertigenden Industrie, insbesondere in der **Automobilindustrie, des frühen 20. Jahrhunderts**:

Gute Qualität entsteht nicht von selbst;
mit schlechter Qualität muss man vor allem dann rechnen, wenn

- die Arbeiter **schlecht ausgebildet** sind,
- keine **soziale Kontrolle** des Verhaltens stattfindet,
- die Produkte den einzelnen Arbeitern **nicht zugeordnet** werden können oder
- die erreichte Qualität für die Verursacher ohne **Wirkung** bleibt.

So wurden um 1920 in den USA Güter (Autos) produziert, die **billig, aber von schlechter Qualität** waren.

Dagegen setzte die Industrie die **Qualitätssicherung**.

Prinzip: Auf die **Fertigung** folgt eine **Prüfung**.
Nur Produkte, die die Prüfung bestehen, werden ausgeliefert.

Die Prüfung setzt die Existenz eines (realen oder virtuellen) **Ideals** voraus, die Prüfung ist also der **Vergleich mit dem Ideal** (s.u.).

Bei großen Stückzahlen: **statistische Qualitätssicherung** (Prüfung einer **Stichprobe,** Anwendung statistischer Modelle).

Problem in der **Softwaretechnik**: Es gibt **keine Produktion, nur Entwicklung** (d.h. die *Herstellung* eines Ideals).

Es steht kein Ideal für die Prüfung zur Verfügung!

Wir brauchen also eine **QS der Entwicklung**, nicht der Fertigung.

Die Methoden und Mittel der traditionellen Qualitätssicherung sind **für die Softwaretechnik unbrauchbar!**

13.1 Software-Qualitätssicherung (SQS)

> **software quality assurance** — See: quality assurance.
>
> **quality assurance** — **(1)** A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.
>
> **(2)** A set of activities designed to evaluate the process by which products are developed or manufactured.

IEEE Std 610.12 (1990)

Bedeutung (2) ist obsolet, denn die **Prozessbewertung** ist zu einem eigenen Thema geworden (⇒ Kapitel 11).

Die Bedeutung (1) entspricht dem üblichen Verständnis von Software-Qualitätssicherung: **Alles, was zu tun ist, damit man der Qualität eines Produkts trauen kann**, gehört dazu.

Die Doppeldeutigkeit dieser Definition ist sinnvoll:

Damit man dem Produkt traut, kann man es

- **besser machen** oder
- **nachweisen, dass es gut ist**.

Beides ist Qualitätssicherung.

Ein naives, aber keineswegs falsches Verständnis des Begriffs ist:

SQS = Gesamtheit der Aktivitäten,
die dem Ziel dienen, gute Software-Qualität zu erreichen

Maßnahmen zur Hebung der Qualität

Viele Maßnahmen tragen dazu bei, die Qualität zu heben (oder genauer: Qualitätsmängeln vorzubeugen). Beispiele:

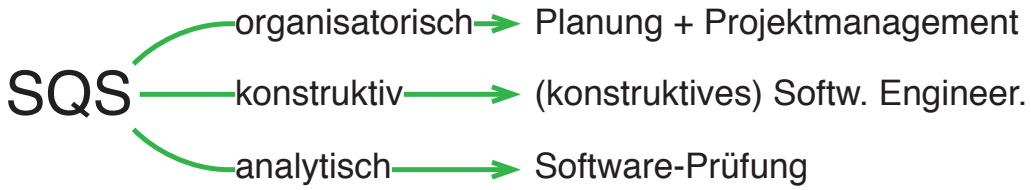
- Einen **Projektplan** erstellen
- Nicht nur die Konstruktion, sondern auch die **Prüfungen planen**
- Die **Dokumentation planen**
- Identifikation und Verwaltung der Ergebnisse regeln (**CM**)
- Aufgaben **erst stellen, dann lösen**
- Resultate **erst prüfen, dann verwenden**
- Für alle wiederkehrenden Arbeiten **Checklisten** verwenden
- **Hohe Programmiersprachen** einsetzen
- Muster (**Templates**) und **Codierrichtlinien** vorgeben
- Fehler und Änderungswünsche **kontrolliert bearbeiten**
- **Fehlerstatistiken** führen
- Jedes abgeschlossene Projekt **kritisch analysieren**

Strukturell steht die Software-Qualitätssicherung auf drei Säulen:

(a) organisatorische Qualitätssicherung
Projektplanung und -management, QS-Organisation

(b) konstruktive Qualitätssicherung
das ganze Spektrum der Möglichkeiten im Software Engineering

(c) analytische Qualitätssicherung
alle Prüfungen (Reviews, Tests usw.).



Achtung: Qualität kann man nicht *ins Produkt hineinprüfen*!
Darum sind (a) und (b) wichtiger als (c)!

13.2 Prüfungen

Prüfungen haben den Zweck, **die Qualität eines Prüflings festzustellen**, insbesondere, ob er im Sinne der Anforderungen fehlerfrei ist.

- Prüfungen zeigen kollektive und individuelle **Schwächen der Prüflinge** (also der Dokumente, Programme usw.). Sie geben damit **Hinweise für das Software Engineering** (allgemein und speziell).
- Direkt erhöhen Prüfungen die **Qualität der Prüflinge** nicht.
- Prüfungen liefern implizit eine Definition der **Qualitätskriterien**.
- Die **Erwartung einer Prüfung** beeinflusst das Verhalten der Entwickler und damit (indirekt) die **Qualität des Produkts**.

13.3 Mängel und Fehler (——)

13.4 Software Engineering und Software-Prüfung

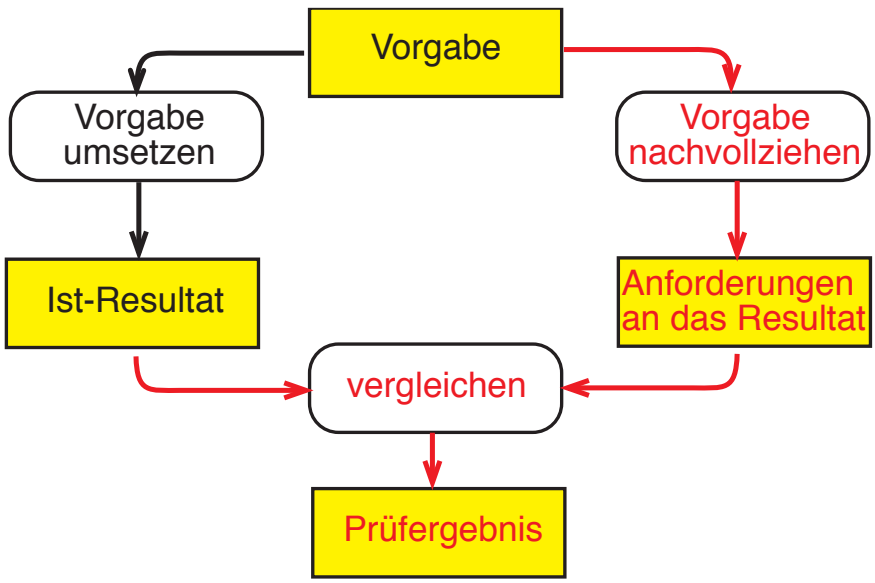
Prüfung ist keine moralische, sondern eine praktische Forderung:

Die Prüfung der Software ist also – wie das ganze Software Engineering – nicht Selbstzweck, sondern wirtschaftlich geboten.

13.5 Prüfungen im Überblick

Alle Prüfungen folgen grundsätzlich dem gleichen **Schema**:

Neben dem eigentlichen Weg von einer Vorgabe zum Resultat (der **Produktion**) wird ein zweiter Weg durchlaufen, auf dem Anforderungen an das Resultat ausgewählt oder festgestellt werden.



Auf diese Weise ist es möglich, das **Resultat** mit den **Anforderungen an das Resultat** zu **vergleichen**.

Eine dabei festgestellte **Diskrepanz** weist auf einen Fehler hin.

Positive und negative Prüfresultate

Prüfung führt zu einem **Befund**: Prüfergebnis ist **positiv** (!)

Vgl. die ähnliche Terminologie in der Medizin, wo man von einem positiven oder negativen Befund spricht.

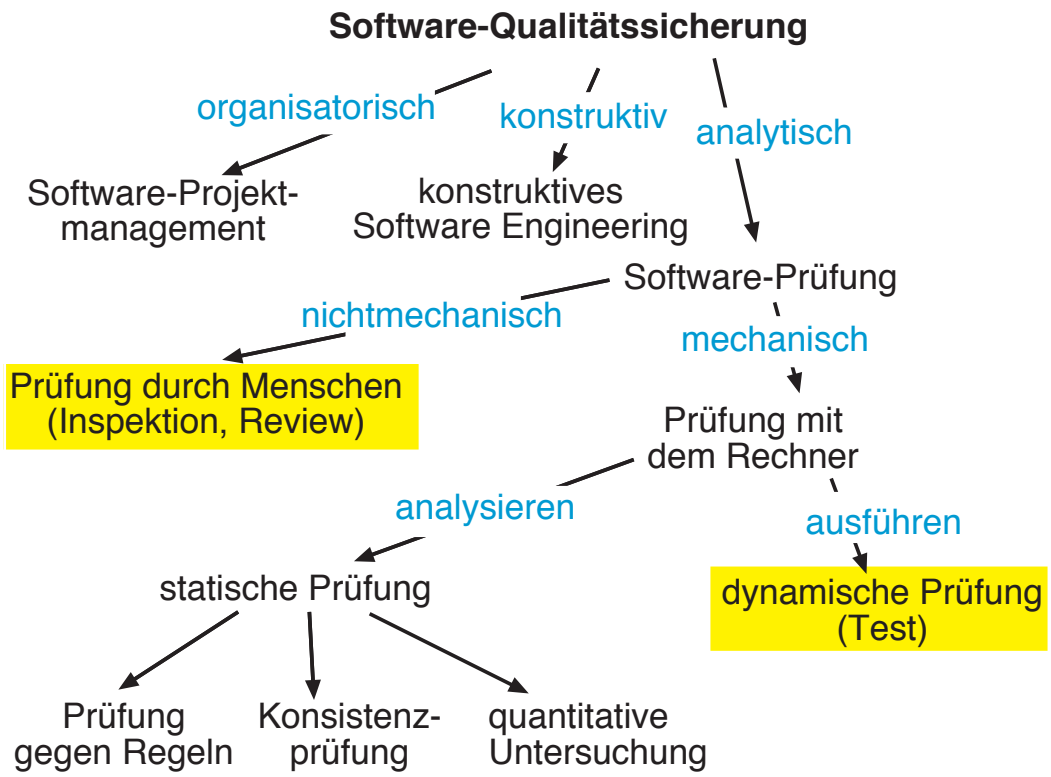
Wenn der Grund für den Befund *nicht* in einem Fehler des Prüflings liegt, ist das Prüfergebnis **falsch positiv**.

(Beispiele möglicher Ursachen:
Falsche Sollresultate, falscher Prüfling, Fehler beim Vergleich)

Falsch positive Resultate machen unnötig Arbeit, sind aber sonst **harmlos**. Viel schlimmer: **falsch negative** Resultate, bei denen ein tatsächlich vorhandener Fehler *nicht* entdeckt wird.

Leider sind falsch negative Prüfresultate oft, z.B. beim Test, der Normalfall.

Die Trennung der Korrektur von der Prüfung (—–)



Vergleich Inspektion versus Test

Eine **mechanische Prüfung** (Test) setzt voraus, dass der Prüfling einer **mechanischen Analyse** zugänglich sind. Darum kommt ganz überwiegend (nur) die **Inspektion** in Frage.

das **Dokument** ... ... ist **prüfbar durch**:	**Inspektion**	**Test**
Lastenheft	X	
Pflichtenheft	X	
Systementwurf	X	
Benutzerhandbuch	X	
Testdaten	X	
Definition der Daten und Algorithmen	X	
Code	X	X
Anleitungen etc.	X	

Inspektionsverfahren

Es gibt viele Verfahren, Software zu inspizieren.
Hier werden behandelt:

1. die ***Durchsicht*** (der „Schreibtisch-Test“)
2. die ***Stellungnahme***
3. das ***technische Review***
4. der ***Walkthrough***
5. die ***Design- and Code Inspections***

Schwerpunkt hier: Reviews

Gesprächsrunden – vor allem zur Klärung der Anforderungen – werden ebenfalls als Reviews bezeichnet werden.

Wir sprechen in diesem Fall von **Workshops**. Sie sind sehr sinnvoll, haben aber eine andere Zielsetzung als die Reviews.

13.6 Reviews

Nachfolgend wird das **technische Review** beschrieben. In der Praxis gibt es zahlreiche Varianten. Zwischen diesen Formen gibt es keine simple Qualitätsordnung!

Eine Software-Einheit wird (dezentral) **von mehreren Gutachtern inspiziert**; in einer gemeinsamen Sitzung werden die **Mängel zusammengetragen und dokumentiert**.

Ziel des Reviews ist es, Fehler zu finden, nicht, die Fehler auch zu beheben.

Prüfling kann jeder in sich abgeschlossene, für Menschen lesbare Teil von Software sein — ein einzelnes Dokument, ein Codemodul, ein Datenmodul.

Zur Prüfung werden **Referenzunterlagen** benötigt (**Vorgabe** oder **Spezifikation**, **Richtlinien**, evtl. auch **Fragenkataloge**).

Rollen im technischen Review

Zuständigkeiten und Verantwortlichkeiten sind im Review klar durch folgende Rollen definiert:

Der **Moderator** leitet das Review, ist also für den ordnungsgemäßen Ablauf verantwortlich.

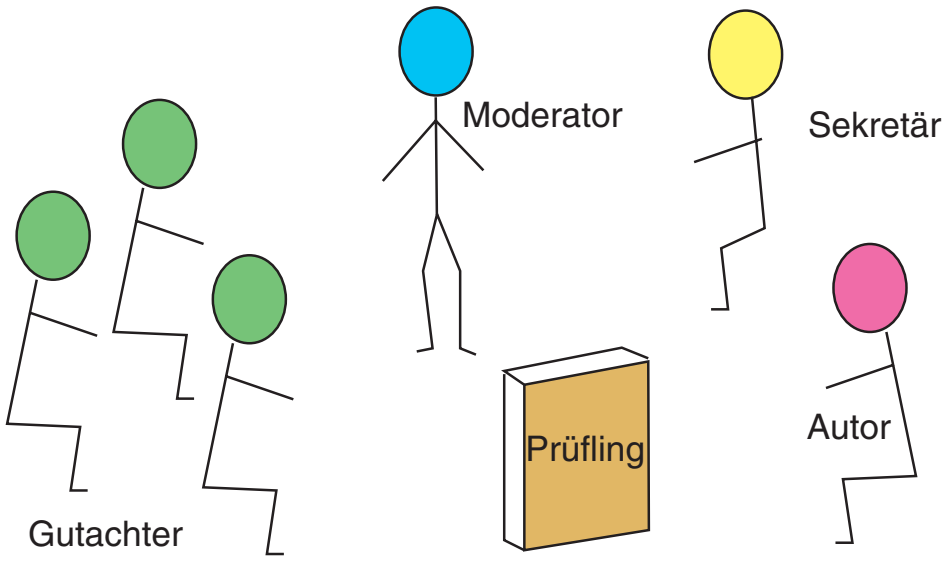
Der **Notar** führt das Protokoll.

Die **Gutachter** sind Kollegen, die den Prüfling beurteilen können.

Der **Autor** ist der Urheber des Prüflings oder ein Repräsentant des Teams, das den Prüfling erstellt hat.

Der **Manager** (Linienvorgesetzte oder Projektleiter) hat den Auftrag zur Erstellung des Prüflings gegeben und somit auch die Verantwortung für die Freigabe des Prüflings. **Er sollte** (vor allem in den ersten Versuchen) **nicht am Review teilnehmen!**

Das **Review-Team** bilden alle Teilnehmer des Reviews außer dem Autor.



Die Gutachter erhalten im technischen Review immer **konkrete Aufträge**. Diese haben einen doppelten Sinn:

- Fehler, die immer wieder vorkommen, werden sehr wahrscheinlich entdeckt.
- Ein Gutachter, der nach bestimmten Mängeln sucht, ist insgesamt aufmerksamer als ohne speziellen Prüfauftrag.

Fragenkatalog für Gutachter **Art des Dokuments: Spezifikation**

Bereiten Sie die Review-Sitzung vor, indem Sie den Prüfling speziell unter den Ihnen in der Einladung zugeordneten Aspekten aus der folgenden Liste prüfen:

Aspekt "Form":

Ist die Darstellung im Dokument sinnvoll?

1. Sind alle Anforderungen erkennbar, d. h. von Erklärungen unterscheidbar?

2. Sind alle Anforderungen eindeutig referenzierbar?
3. Ist die Spezifikation jeder Anforderung eindeutig?
4. Sind alle Anforderungen überprüfbar formuliert?

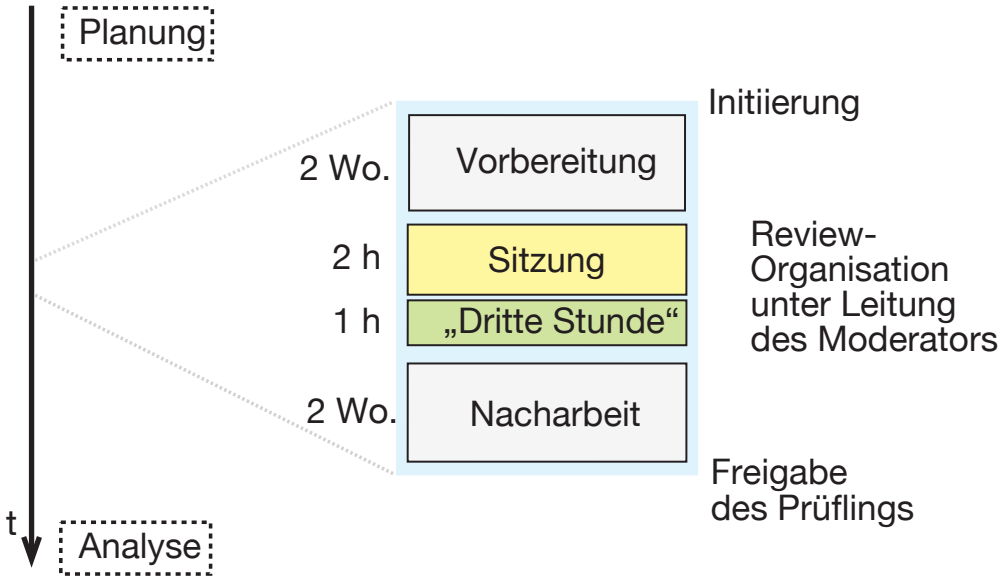
Aspekt "Schnittstellen":

Sind alle Schnittstellen eindeutig spezifiziert?

5. Sind alle Objekte der Umgebung (Benutzer, andere Systeme, Basis-Software etc.) sowie alle Informationsflüsse von und nach diesen Objekten spezifiziert?
6. Sind alle Benutzerklassen des Systems (Dauerbenutzer, gelegentliche Benutzer, System-Administrator etc.) identifiziert?
7. Ist die Bedienschnittstelle für jede der Benutzerklassen festgelegt?
8. Ist die Bedienphilosophie einheitlich?
9. Ist das beschriebene Bedienkonzept den Vorkenntnissen der Benutzer angemessen?

...

Die eigentliche Review-Sitzung ist eingerahmt von je etwa zwei Wochen **Vorbereitung** und **Nacharbeit**;
unmittelbar nach der Sitzung kann die „**Dritte Stunde**“ folgen.



Organisation und Ablauf eines Reviews

- **Anstoß**: von der Konfigurationsverwaltung. Der Moderator wird informiert, er verteilt die Einladungen an die Gutachter; Prüfling und Prüfaufträge sind Teile der Einladung.
- **Vorbereitung**: Die **Gutachter lesen** das zu prüfende Dokument und prüfen es nach den ihnen zugeteilten Gesichtspunkten.
- **Review-Sitzung**: Die Gutachter tragen die in der Vorbereitung entdeckten Mängel vor. Gemeinsam erheben, gewichten und protokollieren sie die Befunde.
- **Resultat**: Liste der Schwächen mit Empfehlung, welche Arbeiten vor der Freigabe des Prüflings durchgeführt werden sollten.
- **Dritte Stunde**: Die Gutachter und der Autor reden ohne Regeln und ohne Protokoll.
- **Nacharbeit**: Sache des Autors oder der Autoren; **Umfang** legt der Manager fest.

Planung und Analyse finden lange vor/nach dem Review statt.

Review-Regeln

1. Der **Moderator organisiert das Review**, lädt dazu ein und führt es durch. (Zweckmäßig: fester Review-Termin)
2. Die Review-Sitzung ist auf **2h** beschränkt. Falls nötig wird eine **weitere Sitzung**, frühestens am nächsten Tag, einberufen.
3. Der Moderator **sagt oder bricht die Sitzung ab**, wenn die Sitzung nicht erfolgreich durchgeführt werden kann. Der **Grund des Abbruchs** ist zu protokollieren.
4. Der **Prüfling**, nicht der Autor **steht zur Diskussion**. Das heißt:
   - Gutachter müssen auf **Sprache** und Ausdrucksweise achten.
   - Der Autor darf weder sich noch den Prüfling **verteidigen**.
5. Die **Rollen** werden nicht vermischt. Insbesondere darf der **Moderator** nicht gleichzeitig als **Gutachter** fungieren.
6. **Stilfragen** (außerhalb der Richtlinien) werden nicht diskutiert.

7. **Problemlösen** ist keine Aufgabe des Review-Teams. **Befunde** werden nicht als Anweisungen an den Autor protokolliert.
8. **Jeder Gutachter** bekommt Gelegenheit, seine Befunde angemessen zu präsentieren.
9. Der **Konsens** der Gutachter zu jedem Befund wird laufend **protokolliert**.
10. Die einzelnen **Befunde** werden **gewichtet** als:
    - *Kritischer Fehler*
    - *Hauptfehler*
    - *Nebenfehler*
    - *Gut* (kein Fehler festgestellt)
11. Das Review-Team gibt eine der folgenden **Empfehlungen** über die Annahme des Prüflings ab:
    - *Akzeptieren ohne Änderungen*
    - *Akzeptieren mit Änderungen*
    - *Nicht akzeptieren*
12. Am Schluss **unterschreiben** alle Teilnehmer das Protokoll.

Das Review als soziales Experiment

Wo Reviews eingeführt werden, bedeuten sie für die Beteiligten eine **große Veränderung**, viel schwieriger als die Einführung eines Werkzeugs oder einer Programmiersprache.

Zuerst werden die negativen Folgen sichtbar: Die Aussicht, als Autor im Review zu sitzen, macht Angst. Viele Entwickler sind Autodidakten!

Regeln, deren Anwendung die Anfangsprobleme entschärfen sollen:

- der Ausschluss des Vorgesetzten
- das Verbot, über Stilfragen zu diskutieren
- die Leitung des Reviews durch den erfahrenen Moderator
- die Dritte Stunde, die den Emotionen Auslauf gibt
- der regelmäßige Rollentausch
- ein objektives, technisches Kriterium, welche Resultate einem Review unterzogen werden.

Lohnt sich denn die ganze Mühe?

Klares Ja!

- Viele Fehler werden zu insgesamt akzeptablen Kosten entdeckt. Es wäre weit teurer, die Fehler *nicht* (oder *nicht so*) zu suchen.
- Die Zusammenarbeit der Entwickler wird besser, das Vertrauen zwischen den Software-Leuten deutlich höher; niemand fürchtet sich davor, die anderen um Rat zu fragen. Richtlinien werden nicht belächelt, sondern gelebt.
- Die Beteiligten entwickeln ein sachlich fundiertes Selbstbewusstsein und einen begründeten Stolz auf ihre Arbeit.

Wer einmal an Reviews gewöhnt ist, möchte auf keinen Fall mehr darauf verzichten.

Ein Erste-Hilfe-Kasten für die häufigsten Probleme mit Reviews

Reviews sind weltweit seit langem als wichtigste Technik der Software-Prüfung anerkannt, ihr Nutzen ist klar nachgewiesen und von allen, die regelmäßig und systematisch Reviews durchführen, bestätigt. Trotzdem gibt es viele Fälle, in denen die Einführung von Reviews gescheitert oder zu einer ohne Überzeugung betriebenen Formalübung degeneriert ist. In diesem Abschnitt werden die wichtigsten Schwierigkeiten diskutiert.

1. Für die Vorbereitung oder sogar für die Review-Sitzung selbst fehlt die Zeit ⇒ Reviews in der Planung berücksichtigen
2. Die Reviews finden in einer bis zur Unkenntlichkeit reduzierten Form statt ⇒ Standard (Minimalreview) vorgeben
3. Gute Moderatoren fehlen ⇒ Leute aussuchen und ausbilden

4. Bezugsdokumente fehlen ⇒ suchen, anpassen, bereitstellen
5. Entwickler haben Angst ⇒ Erste Reviews gründlich vorbereiten (und auf die Ängste eingehen, selbst wenn sie geleugnet werden)
6. Reviews beißen sich an Äußerlichkeiten fest ⇒ ihre Bedeutung klarstellen, dann weiter. Beim zweiten Review aber diesen Punkt erneut betonen, nicht schleifen lassen.
7. Bezugsdokumente sind ungeeignet ⇒ diskutieren u. verbessern
8. Zeitdruck sabotiert Prüfungen ⇒ Klare Entscheid. der Prioritäten
9. Geprüfte Dokumente werden verändert ⇒ CM verbessern
10. Interesse an Reviews sinkt ⇒ mit Statistiken Erfolg nachweisen

Gründe, die Einführung von Reviews zu verschieben

Nur in speziellen Situationen ist von der Review-Einführung abzuraten:

A wenn ein Projekt **praktisch bereits gescheitert** ist und die Verantwortlichen nach einem Wunder Ausschau halten,

B wenn **keinerlei Bezugsdokumente** vorliegen,

C wenn die Entwickler in **feindliche Lager** gespalten sind.

Was tun?

A **Retten**, was zu retten ist, z.B. zentrale Dokumente (Spezifikation, Planung für das Rest-Projekt) in akzeptablen Zustand bringen.

B **Versuchsreview** durchführen, Defizite erkennen, Arbeit an Bezugsdokumenten anstoßen.

C Reviews zunächst **separat weiterlaufen lassen**, Gutachter zwischen den Lagern austauschen

13.7 Varianten der Software-Inspektion

Durchsicht

... führt der **Entwickler allein** durch. Es ist zweckmäßig, dazu den Bildschirm zu verlassen und das (Teil-)Resultat in Ruhe, aus einer gewissen Distanz, zu überprüfen.
Die Durchsicht sollte selbstverständlich und obligatorisch sein!

In Zeiten der (reinen) Stapelverarbeitung war die Durchsicht einfach **notwendig**. Heute wissen wir (u.a. durch Humphreys **PSP**), dass die Durchsicht **effizienter ist als jeder** (spontane) **Test**.

Stellungnahme

ist ein **„off-line"-Review** unter der Regie des Autors.

Den **Vorteilen** (geringer Organisationsaufwand) stehen **erhebliche Nachteile** gegenüber (Prüfling nicht geschützt, Qualität der Prüfung und Umsetzung der Resultate nicht kontrolliert).

Structured Walkthrough

ist die **Billig-Variante** des Reviews: Der Autor ist Moderator; er kompensiert durch seine Präsentation die Einsparung (oder Reduktion) der Vorbereitung.
Während seiner Vorbereitung entdeckt er selbst viele Fehler.

Varianten **mit oder ohne Vorbereitung der Gutachter**

Typische Anwendung: **Programmcode**.

Die **Effizienz** ist niedriger als beim Review!

Design and Code Inspection

... (nach Michael Fagan, IBM) ist die **Edel-Variante** der Reviews: Mit Einführungssitzung, Gutachter-Notizen, die abgegeben werden, Vorleser, Entscheidungskompetenz, Metriken-Erhebung.

Fazit

Allgemein und für die Inspektionen im besonderen gilt:

„You get what you pay for!“