Willkommen zur Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2012

Prof. Dr. Jan Jürjens

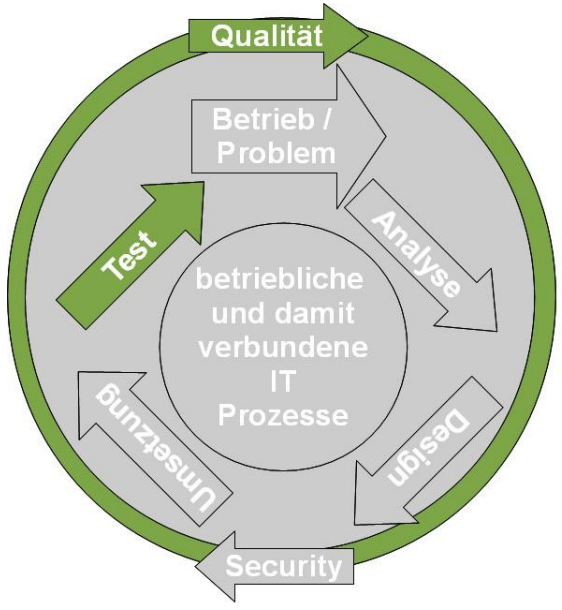
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

4.3 Statischer Test

Basierend auf dem Foliensatz
„Basiswissen Softwaretest - Certified Tester“ des „German Testing Board“
(nach Certified Tester Foundation Level Syllabus, deutschsprachige Ausgabe, Version 2011) (mit freundlicher Genehmigung)

Einordnung Statischer Test

- Business Prozesse
- Qualitätsmanagement
- Testen
  - Einführung
  - Grundlagen Softwaretesten
  - Testen im Softwarelebenszyklus
  - Statischer Test
  - Black-Box-Test
  - White-Box-Test
  - Test-Management
  - Testwerkzeuge
  - Fuzzing
- Sicheres Software Design

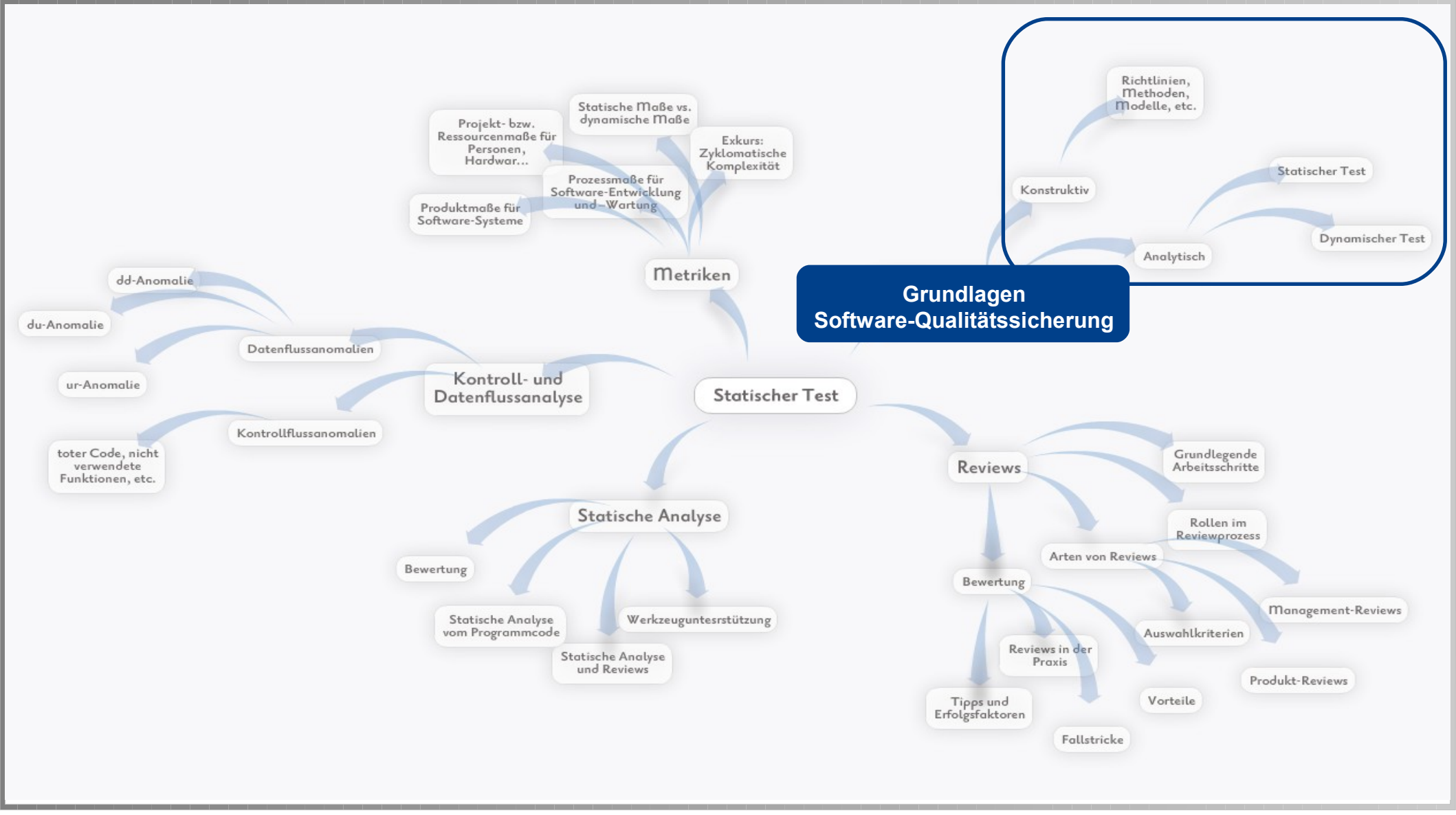


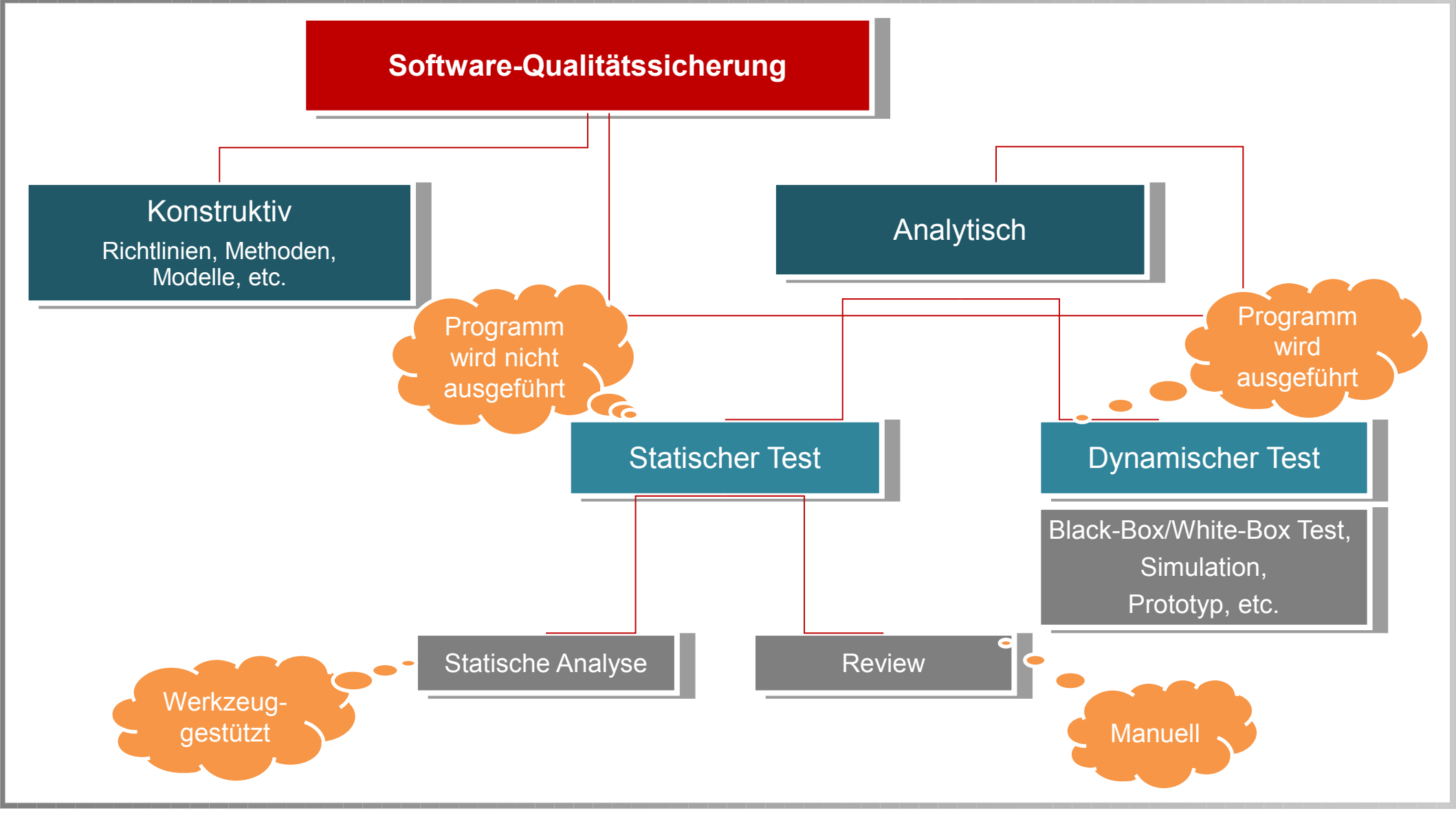
4.3 Statischer Test

4.3 Statischer Test

- Grundlagen
- Reviews
- Statische Analysen
- Kontroll- und Datenflussanalyse
- (Metriken: Abschnitt 3.2)

Statischer Test

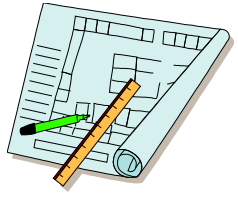



Software-Qualitätssicherung
Konstruktiv
Richtlinien, Methoden, Modelle, etc.
Analytisch
Programm wird nicht ausgeführt
Programm wird ausgeführt
Statischer Test
Dynamischer Test
Black-Box/White-Box Test, Simulation, Prototyp, etc.
Statische Analyse
Review
Werkzeug-gestützt
Manuell

Statischer Test vs. dynamischer Test

Im Entwicklungsprozess werden unterschiedliche Produkte (Artefakte) erstellt:

- Informelle Texte
- Modelle
- Formale Texte
- Programmcode
- ...



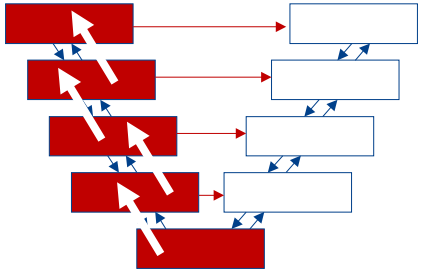
Programme sind statische Beschreibungen von dynamischen Prozessen (Berechnungen).

Dynamische Tests prüfen die durch »Interpretation« einer Beschreibung (Testobjekt) resultierenden Prozesse. Im **statischen Test** wird das Testobjekt nicht »ausgeführt«, sondern nur »als solches« untersucht.

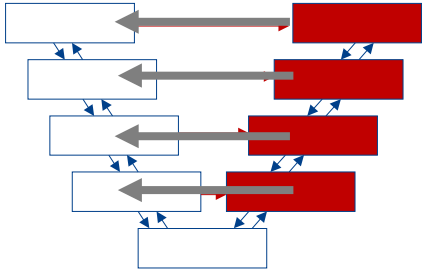
Im Gegensatz zum dynamischen Test finden statische Tests eher Ursachen der Fehlerwirkungen (Fehlerzustände) als Fehlerwirkungen.

Statischer Test vs. dynamischer Test im SW-Lebenszyklus

Im **statischen Test** wird das Testobjekt gegen ein Dokument aus einer vorhergehenden Konstruktionsphase „**verifiziert**“, bei informellen Dokumente durch Review, bei formalen Dokumenten (z.B. Programmcode) durch statische Analyse.



Dynamische Tests prüfen die durch »Interpretation« einer Beschreibung (Testobjekt einer Teststufe) resultierenden Prozesse gegen ein Dokument aus der Konstruktionsphase, was dann „**validieren**“ genannt wird.



Statischer Test

Analyse des Testobjekts (meist eines Dokuments) durch intensive Betrachtung.

Ziel: **Ermittlung von Fehlerzuständen** (Defekten) im Dokument:

- Verstöße gegen Spezifikationen oder einzuhaltende Standards, Fehler in Anforderungen, Fehler im Design, unzureichende Wartbarkeit und falsche Schnittstellenspezifikationen
- Nachweis der Verletzung von Projektplänen
- Ergebnisse der Untersuchungen werden darüber hinaus dazu benutzt, den Entwicklungsprozess zu optimieren.

Grundidee: **Prävention !**

- Fehlerzustände und Abweichungen sollen so früh wie möglich erkannt werden, noch bevor sie im weiteren Verlauf der Software-Entwicklung zum Tragen kommen und aufwändige Nach- oder Verbesserungen nach sich ziehen.

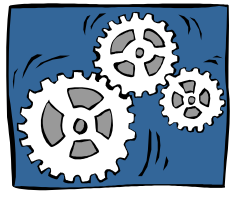
Manuelle vs. automatisierte Prüfung

Betrachtung des Prüfobjekts durch Mensch oder Werkzeug

Reviews

- Manuelle Prüfungen durch eine oder mehrere Personen
- Menschliche Analyse- und Denkfähigkeit wird genutzt, um komplexe Sachverhalte zu prüfen und zu bewerten.
- Kann bei allen Dokumenten durchgeführt werden, die während des Softwareentwicklungsprozesses erstellt oder verwendet werden (z.B. Vertrag, Anforderungsspezifikation, Designspezifikation, Quelltext, Testkonzepte, Testspezifikationen, Testfälle, Testskripte oder Anwenderhandbücher).

Statische Analyse

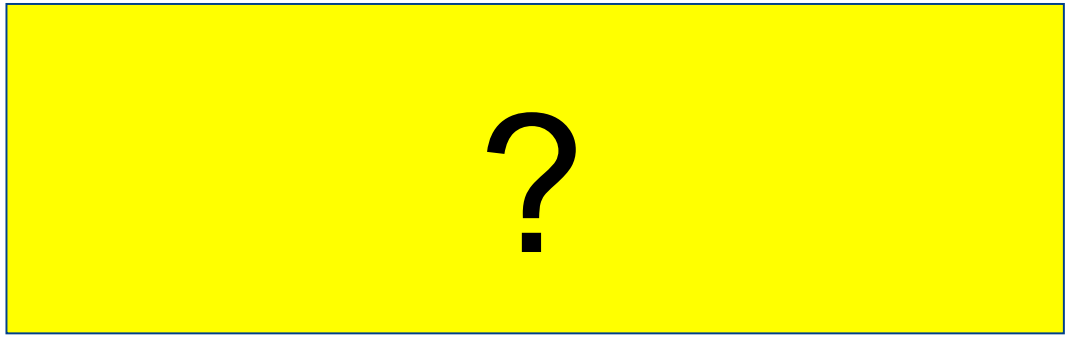


- Automatisierte Prüfungen durch entsprechende Werkzeuge gegen Regeln.
- Nur bei Dokumenten mit formaler Struktur (z.B. Programmtext, UML-Diagramme).

Wie viele "F" kommen im folgenden Text vor?

```
FINISHED FILES ARE THE RE-
SULT OF YEARS OF SCIENTIF-
IC STUDY COMBINED WITH THE
EXPERIENCE OF YEARS
```

Wie viele "F" kommen im folgenden Text vor?



Vergleich manuelle vs. Automatisierte Überprüfung

- Es sind sechs!
- Wie viele hatten Sie gezählt?
- Drei zu finden ist normal - sechs wirklich gut!
- Irren ist menschlich ☺

```
FINISHED FILES ARE THE RE-
SULT OF YEARS OF SCIENTIF-
IC STUDY COMBINED WITH THE
EXPERIENCE OF YEARS
```

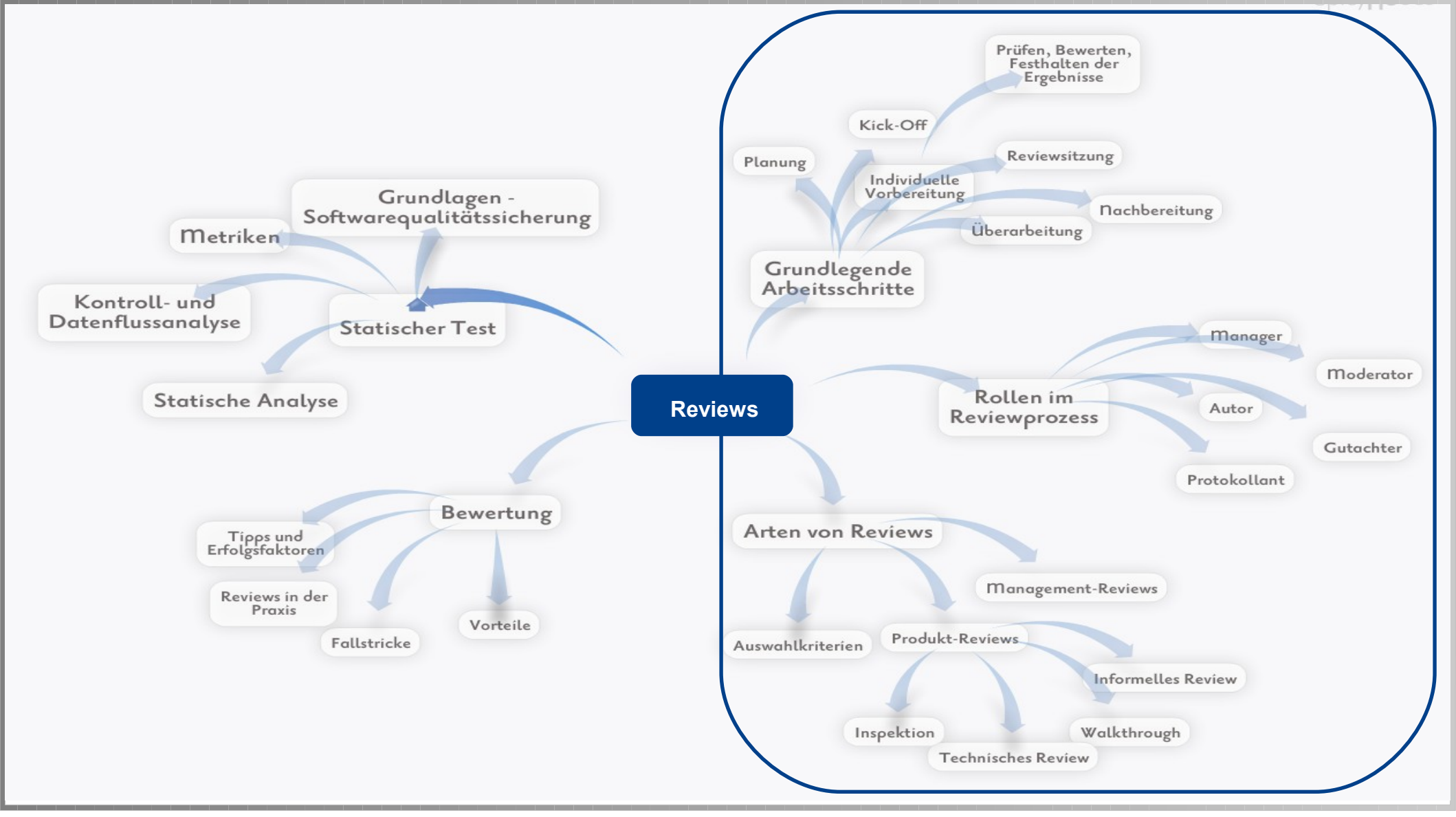
=> Automatisierte Überprüfung erhöht Verlässlichkeit.

4.3 Statischer Test

4.3 Statischer Test

- Grundlagen
- **Reviews**
- Statische Analysen
- Kontroll- und Datenflussanalyse
- (Metriken: Abschnitt 3.2)

Reviews


Reviews
Grundlagen - Softwarequalitätssicherung
Metriken
Kontroll- und Datenflussanalyse
Statischer Test
Statische Analyse
Bewertung
Tipps und Erfolgsfaktoren
Reviews in der Praxis
Fallstricke
Vorteile
Grundlegende Arbeitsschritte
Prüfen, Bewerten, Festhalten der Ergebnisse
Kick-Off
Planung
Individuelle Vorbereitung
Reviewsitzung
Nachbereitung
Überarbeitung
Rollen im Reviewprozess
Manager
Moderator
Autor
Gutachter
Protokollant
Arten von Reviews
Management-Reviews
Produkt-Reviews
Auswahlkriterien
Informelles Review
Inspektion
Walkthrough
Technisches Review

Reviews

Review (survey)

[Duden Wörterbuch Englisch, Mannheim, 1999]

- Übersicht, die (*of* über + Akk.);
- Überblick, der (*of* über + Akk.);
- (*of past events*) Rückschau, die (*of* auf + Akk.);
- *be a review of sth.* einen Überblick od. eine Übersicht über etw. (Akk.) geben;

Bezeichnung für ein bestimmtes Vorgehen bei der Prüfung von Dokumenten als auch ein gebräuchlicher Oberbegriff für die verschiedenen statischen Prüfverfahren, die von Personen durchgeführt werden.

Oft sind Reviews die einzige Möglichkeit, die Semantik solcher Dokumente zu überprüfen.

Weiterer Begriff, in meist analoger Bedeutung: Inspektion.

Reviews: Grundlegende Arbeitsschritte

Bei einem (formalen) Review sind i.d.R. sechs Arbeitsschritte durchzuführen:

Reviews: Planung

Das Management entscheidet, welche Dokumente welchem Reviewverfahren unterzogen werden sollen. Der zu veranschlagende Aufwand für die einzelnen Reviews ist bei der Projektplanung vorzusehen.

Die Eingangs- und Ausgangskriterien für die Durchführung des Reviews sind festzulegen (vor allem bei formaleren Reviewarten). Weiterhin werden Prüfkriterien festgelegt

Der Manager wählt die fachlich geeigneten Personen aus und stellt ein Reviewteam zusammen.

Er vergewissert sich in Kooperation mit dem Autor des Prüfobjektes, dass dieses einen »review-fähigen« Status hat, d.h., die Arbeiten am Dokument einen gewissen Abschluss gefunden und eine ausreichende Vollständigkeit erreicht haben.

Überprüfung, ob Eingangskriterien für die Durchführung des Reviews erfüllt sind.

Findet eine Kick-Off-Sitzung statt, ist Ort und Zeit festzulegen.

Reviews: Kick-Off

Versorgung der am Review beteiligten Personen mit allen benötigten Informationen.

Schriftliche Einladung oder sofortiges erstes Treffen des Reviewteams, um über Bedeutung, Sinn und Zweck der durchzuführenden Reviewsitzung zu informieren.

Sind die beteiligten Personen mit dem Umfeld des zu prüfenden Dokumentes wenig vertraut, kann auf dem Treffen neben der kurzen Vorstellung des Prüfdokuments auch seine Einbettung in das Anwendungsgebiet dargestellt werden.

Neben dem Dokument, das einem Review unterzogen werden soll, müssen den beteiligten Personen zur Verfügung stehen:

- Dokumente, die herangezogen werden müssen, um zu entscheiden, ob eine Abweichung, ein Fehler oder eine korrekte Beschreibung des Sachverhalts vorliegt, also die Dokumente (z.B. Pflichtenheft, Richtlinien oder Standards), gegen die geprüft wird.
- Darüber hinaus sind Prüfkriterien (z.B. in Form von **Checklisten,** s. nächste Folie) sinnvoll, die ein strukturiertes Vorgehen unterstützen.

Reviews: Checkliste Beispiel

Überprüfe die geänderten Komponenten-Anforderungen des System-Entwurfs bzgl. Vollständigkeit (des Nachfolgedokuments):

- Hat jede geänderte oder neu entworfene Klasse des System-Entwurfs eine Repräsentation auf Codeebene ?
- Existieren die Attribute, Methoden mit Parametern und Rückgabetypen und Beziehungen (Assoziation, Aggregation, Vererbung) einer geänderten oder neu entworfenen Klasse des System-Entwurfs im Code ?
- Wenn die Klasse von Observer erbt: Meldet sich die Klasse beim beobachteten *Subject* an (*attach*) und wieder ab (*detach*) ?
- Wurde jedes Zustandsdiagramm aus dem System-Entwurf im Code umgesetzt? Ist jeder Zustand, Zustandsübergang (mit entsprechenden Aktionen, wenn vorhanden), Do-Aktivitäten, entry- und exit-Aktionen des System-Entwurfs im Code repräsentiert ?
- Wurde für einen Zustand vom Typ OR (vgl. Richtlinien für die Ableitung von Java-Code aus OMT/UML-Zustandsdiagrammen) ein Aufzählungstyp definiert ?
- Wurde für jeden Zustand vom Typ BASIC eine IF-Anweisung definiert ?
- Wurden die Unterzustände eines Zustands vom Typ OR mit Hilfe einer switch-Anweisung implementiert ?
- Wurden event-getriggerte Transitionen und interne Aktionen mit Hilfe eines Teil-Zustandsdiagramms realisiert ?
- Importiert die Klasse die im Entwurf unter Imports aufgeführten Klassen ?
- ...

Prof. Dr. H. D. Rombach, Praktikum Software Engineering I, Checkliste zur Inspektion der Implementierung http://wwwagse.informatik.uni-kl.de/teaching/se1lab/ss2002/Phase4/Verifikation_ChecklisteSKD.pdf.

Reviews: Individuelle Vorbereitung

Die beteiligten Personen des Reviewteams haben sich individuell auf die Sitzung vorzubereiten.

Gutachter setzen sich intensiv mit dem zu prüfenden Dokument auseinander und verwenden dabei die zur Verfügung gestellten Dokumente als Prüfgrundlage.

- Meist ist es ratsam, eine Beschränkung auf bestimmte Aspekte vorzunehmen, gegeben durch die gezielte Auswahl der verwendeten Checklisten oder durch weitere Dokumente.

Erkannte potenzielle Fehlerzustände, Fragen oder Kommentare werden notiert.

Reviews: Sitzung (1)

Vom Sitzungsleiter (Moderator) geführt.

In der Regel auf festen Zeitrahmen beschränkt (max. 2 Stunden). Falls nötig weitere Sitzung, frühestens am nächsten Tag, einberufen.

Ziel: Beurteilung des Prüfobjektes in Bezug auf die Einhaltung und Umsetzung der Vorgaben und Richtlinien. Die Bewertung der einzelnen Befunde und die Gesamtbeurteilung soll von allen Gutachtern getragen werden.

Der Moderator hat das Recht, eine Sitzung abzusagen oder abzubrechen, wenn einer oder mehrere Experten (Gutachter) nicht erschienen oder ungenügend vorbereitet sind.

Das Resultat (also das zu prüfende Dokument, das Prüfobjekt) und nicht der Autor stehen zur Diskussion.

- Gutachter müssen auf ihre Ausdrücke und Ausdrucksweisen achten.
- Autor darf weder sich noch das Resultat verteidigen (d.h. der Autor sollte keinen »Angriffen« ausgesetzt werden, die ihn zur »Verteidigung« zwingen).
- Rechtfertigung seiner Entscheidungen wird teilweise als legitim und hilfreich angesehen.

Der Moderator darf nicht gleichzeitig als Gutachter fungieren.

Keine allgemeinen Stilfragen (außerhalb der Richtlinien) diskutieren.

Nach Frühauf, K.; Ludewig, J,; Sandmayr, H.: Software-Prüfung: eine Fibel. Vdf, Verlag der Fachvereine, Zürich, 4. Aufl. 2000

Reviews: Sitzung (2)

Entwicklung und Diskussion von Lösungen ist nicht Aufgabe des Reviewteams.

Jeder Gutachter muss Gelegenheit haben, seine Befunde angemessen zu präsentieren. Der Konsens der Gutachter zu einem Befund ist zu protokollieren.

Befunde (z.B. Fehlerzustände) nicht in Form von Anweisungen an den Autor protokollieren. Zusätzliche konkrete Korrekturvorschläge werden allerdings z.T. durchaus als sinnvoll und hilfreich für die Qualitätsverbesserung angesehen.

Einzelne Befunde gewichten, z. B. als:

- **Kritischer Fehlerzustand („Fehler“):** Prüfling oder Prüfobjekt ist für den vorgesehenen Zweck unbrauchbar, Fehlerzustand muss vor der Freigabe behoben werden.
- **Hauptfehler:** Nutzbarkeit des Prüflings beeinträchtigt, Fehlerzustand sollte vor der Freigabe behoben werden.
- **Nebenfehler:** Geringfügige Abweichung, z.B. Rechtschreibfehler oder Mangel im Ausdruck, beeinträchtig den Nutzen kaum.
- **Gut:** Fehlerfrei, bei Überarbeitung nicht ändern.

Reviews: Sitzung (3)

Reviewteam gibt Empfehlung über die Annahme des Prüflings ab:

- **Akzeptieren** (ohne Änderungen)
- **Akzeptieren mit Änderungen:** kein weiteres Review
- **Nicht akzeptieren:** weiteres Review oder andere Prüfmaßnahme erforderlich

Protokoll enthält Liste aller Befunde (z.B. Fehlerzustände), die in der Sitzung diskutiert wurden, und zusätzlich:

- Informationen zum Prüfobjekt und den verwendeten Dokumenten
- Beteiligte Personen und ihre Rollen
- Kurze Zusammenfassung der wichtigen Punkte
- Ergebnis der Reviewsitzung mit der Empfehlung der Gutachter

Am Schluss unterschreiben alle Sitzungsteilnehmer das Protokoll.

Reviews: Protokoll

Beispiel einer einseitigen Zusammen-fassung

Review Laufzettel und Ergebniszusammenfassung

Projekt						
Autor:						
Dokument:			Version:			
Review-Art						
Planung						
Verantwortlich für die Durchführung des Review						(1)
Unterlagen geprüft:	Ja ☐		Datum:		Nein ☐	
Unterlagen verteilt:	Ja ☐		Datum:		Nein ☐	
Vorbereitung						
Inspektor 1				Dauer:	h	(2)
Inspektor 2				Dauer:	h	(2)
Inspektor 3				Dauer:	h	(2)
Inspektor 4				Dauer:	h	(2)
Inspektor 5				Dauer:	h	(2)
Inspektor 6				Dauer:	h	(2)
Review-Sitzung						
Moderator:						
Datum:		Teilnehmer:		Dauer:	h	
			Summe Aufwand:		h	(3)
Ergebnis						
Schwere Fehler:			Reinspektion erforderlich:	Ja ☐	Nein ☐	
Leichte Fehler:						
Unklarheiten:			Nacharbeit erledigt bis:			
Vorschläge:						
Dokument-Fehler:			Gesamtaufwand:			=(2)+(3)
Abschluss						
Arbeitsergebnis Ok, Nacharbeit fertig.						
Datum:		Unterschrift (1):				

Reviews: Überarbeitung

Beheben der aufgedeckten Fehlerzustände und Protokollierung des aktuellen Status

- typischerweise durch den Autor.

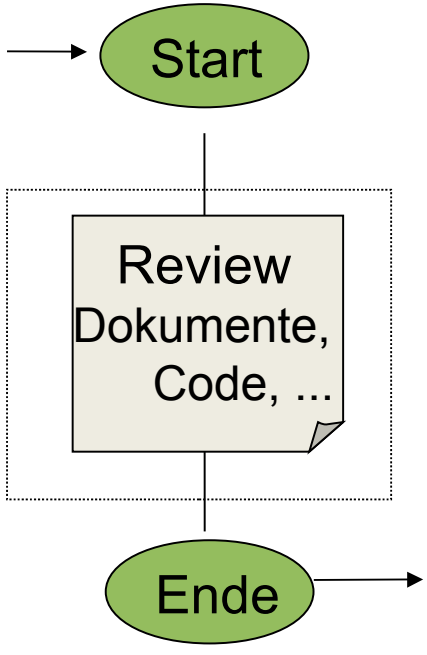
In Abhängigkeit der Nachvollziehbarkeit und Verständlichkeit der Befunde ist ggf. Rücksprache mit den Gutachtern sinnvoll.

Reviews: Nachbereitung

- Kontrolle der ordnungsgemäßen Durchführung der Überarbeitung.
- Prüfung, dass die aufgedeckten Fehlerzustände durch die Korrekturen auch beseitigt wurden.
- Sammeln von Metriken.
- Prüfung von Ausgangskriterien (bei mehr formalen Reviewarten).
- Manager entscheidet, ob er der Empfehlung der Gutachter folgt oder eine andere Entscheidung trifft, für die er dann aber die alleinige Verantwortung trägt.
- War das Resultat des ersten Reviews nicht akzeptabel und muss ein weiteres Review durchgeführt werden, so ist die hier beschriebene Vorgehensweise für das zweite Review entsprechend durchzuführen, jedoch meist verkürzt.

Reviews: Reviewprozess

- zu begutachtendes Objekt/ Dokument
- Anpassung des Review-Prozesses
- Review-Checklisten
- Projektdaten



- Befunde, Abweichungen
- (ausgefüllte) Review-Checkliste
- Ergebnisse, Aufgabenliste

Metriken, beispielsweise:

- Anzahl der Fehlerzustände, Abweichungen, Befunde
- Anzahl der festgelegten Änderungen
- Aufwand der Überprüfung

Reviews: Rollen

Manager

Entscheidet über die Durchführung von Reviews, stellt Zeit im Projekt-Plan zur Verfügung und überprüft, ob die Reviewziele erfüllt sind.

Moderator

Die Person, die das Review eines Dokuments bzw. von zusammengehörenden Dokumenten leitet, einschließlich der Reviewplanung, der Leitung der Sitzung und der Nachbereitung nach der Sitzung. Falls nötig, kann der Moderator zwischen den verschiedenen Standpunkten vermitteln. Er ist häufig die Person, von der der Erfolg des Reviews abhängt.

Autor

Der Verfasser oder die Person, die für das zu prüfende Dokument (bzw. die zu prüfenden Dokumente) hauptverantwortlich ist.

Gutachter

Personen mit einem spezifischen technischen oder fachlichen Hintergrund (auch Prüfer oder Inspektoren genannt), die nach der nötigen Vorbereitung im Prüfobjekt Befunde identifizieren und beschreiben (z.B. Fehlerzustände).

Protokollant

Die Person, die alle Ergebnisse, Probleme und offenen Punkte dokumentiert, die im Verlauf der Sitzung identifiziert werden.

Reviews: Arten

Reviews variieren zwischen sehr informell und sehr formal (d.h. gut strukturiert und geregelt). Formalismus des konkret anzuwendenden Reviewprozesses ist abhängig von Faktoren wie:

- Reife des Entwicklungsprozesses,
- gesetzliche oder regulatorische Anforderungen oder
- dem Bedarf für einen Prüfnachweis.

Art und Weise, wie ein Review durchgeführt wird, ist abhängig von den festgelegten Zielen, z.B.:

- das Finden von Befunden und Fehlerzuständen,
- dem Erwerben von Verständnis oder
- einer Diskussion mit Entscheidung durch Konsens.

Arten anhand des untersuchten Prüfobjektes:

- **Management-Reviews:** analysieren Projektpläne und den Entwicklungsprozess als solches
- **Produkt-Reviews:** beziehen sich auf Produkte oder Teilprodukte, die während des Entwicklungsprozesses erstellt werden: **Informelles Review**, **Walk-through**, **Technisches Review**, **Inspektion.**

Management-Reviews – Besonderheiten

Es werden keine einzelnen Dokumente geprüft.

- Prüfobjekt ist das Projekt als Ganzes und die Feststellung seines augenblicklichen Zustands.
- Der Projektzustand wird hinsichtlich technischer, wirtschaftlicher, zeitlicher und managementmäßiger Aspekte bewertet.

Zur Vorbereitung können Vorträge zu ausgewählten Themen ausgearbeitet werden.

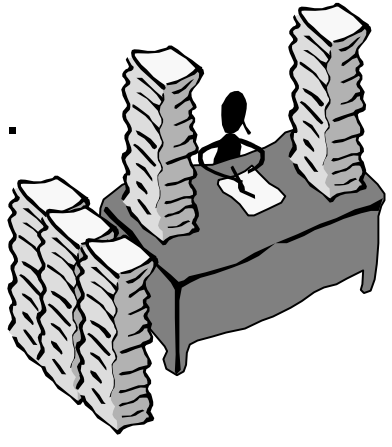
- Informationsgrundlage sind u.a. die Projektdokumentation und die Protokolle der bis zu diesem Zeitpunkt bereits durchgeführten Prüfungen einzelner Produkte und Dokumente.
- Die vorbereiteten Präsentationen werden in der Sitzung vorgetragen.

Management-Reviews...

- Können weit mehr als zwei Stunden dauern, sie können sogar die Dauer von bis zu zwei Arbeitstagen annehmen.
- Werden oft beim Erreichen eines Meilensteins der Projektplanung durchgeführt, wenn eine Hauptphase im Softwareentwicklungsprozess abgeschlossen werden soll oder als Post-Mortem-Analyse, um aus dem beendeten Projekt zu lernen.

Informelles Review

- Informelle Prüfung innerhalb einer Entwicklungsaktivität.
- Meist als Einzelreview, kein formaler Prozess, wahlweise kann dokumentiert werden oder auch nicht.
- »Schreibtischtest« des eigenen Programms oder (besser) des Programms eines anderen.
- »Vier-Augen-Prinzip« - kann integriert im Pair Programming (XP) sein oder ein technischer Leiter oder Senior-Entwickler unterzieht Entwurf und Quelltext einem Review.
- Nutzen variiert in Abhängigkeit von den Gutachtern.
- kostengünstiger Weg bei hohem Nutzen.



Walkthrough (1)

„Manuelle, informale Prüfmethode, um Fehler, Defekte, Unklarheiten und Probleme in schriftlichen Dokumenten zu identifizieren. Der Autor präsentiert das Dokument in einer Sitzung den Gutachtern.“ Balzert, H.: Lehrbuch der Softwaretechnik, Band II. Spektrum Akademischer Verlag, 1998

Schwerpunkt bildet die Sitzung (oft »open end«). Vorbereitung hat im Vergleich zu den anderen Review-Arten den geringsten Umfang, z.T. kann sogar auf sie verzichtet werden.

In der Sitzung übernimmt oft der Autor die Rolle des Moderators (nicht aber die des Protokollanten !); er stellt das Prüfobjekt den Gutachtern vor.

- Da der Autor die Sitzung leitet, kann er ihren Verlauf stark beeinflussen; dies kann sich nachteilig auf das Ergebnis auswirken, wenn der Autor die Diskussion der kritischen Stellen des Prüfobjektes nicht intensiv genug durchführen lässt oder die Diskussion gar nicht erst zulässt.

Meist werden typische Anwendungsfälle (Benutzungssituationen, Szenarien, Probeläufe) im Kreis gleichgestellter Mitarbeiter ablauforientiert durchgespielt.

- Es können auch einzelne Testfälle nachgespielt werden;
- Gutachter versuchen durch meist spontane Fragen, mögliche Fehler(zustände) und Probleme aufzudecken.

Walkthrough (2)

Geeignet für kleine Entwicklungsteams und für Prüfungen „unkritischer“ Dokumente. Wenig Aufwand, da Vor- und Nachbereitungen von geringem Umfang bzw. nicht zwingend erforderlich.

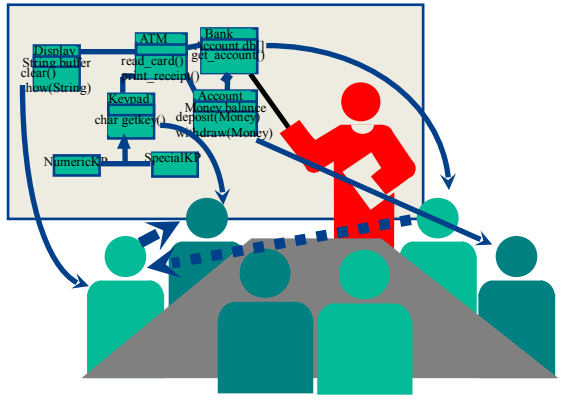
Liste der Befunde und Bericht kann in der Praxis von informell bis sehr formal variieren.

Für die Nacharbeit ist der Autor verantwortlich, eine Kontrolle findet nicht statt.

Hauptzweck: Lernen, Verständnis erzielen, Fehler(zustände) finden.

Entwurfs-Walkthrough:

- Jeder Reviewer übernimmt die Verantwortung für eine oder mehrere Komponenten des Entwurfs.
- Anhand konkreter Szenarien werden die Interaktionen zwischen den Komponenten nachgespielt.
  - Schnittstellen und Parameter beachten.
  - Welche Komponente hat welches „Wissen“ ?



Technischer Review

Dokumentierter, definierter Fehlerfindungsprozess, der gleichgestellte Mitarbeiter und technische Experten einschließt. Kann als »Peer Review« ohne Teilnahme des Managements ausgeführt werden und in der Praxis von informell bis sehr formal variieren. **Untersuchungsziele:**

- Übereinstimmung des zu prüfenden Dokuments mit der Spezifikation; Eignung für den beabsichtigten Einsatz; Einhaltung von Standards.
- Bei hohem Formalisierungsgrad definierte Eingangs- und Ausgangskriterien der einzelnen Prüfschritte.
- Diskussion, Entscheidungen treffen, Alternativen bewerten, Fehler(zustände) finden, technische Probleme lösen.

Varianten: Nutzung von Checklisten, Erstellung des Reviewberichts oder Liste der Befunde, Managementteilnahme

Hoher Aufwand in der Vorbereitungsphase: Fachexperten als Gutachter prüfen das Prüfobjekt anhand der festgelegten Prüfkriterien. Anmerkungen der Gutachter zu den einzelnen Prüfkriterien schriftlich festhalten und dem Sitzungsleiter vorab übergeben. Sitzungsleiter priorisiert diese nach vermuteter Wichtigkeit.

In der Sitzung: Idealerweise durch einen geschulten Moderator geleitet (nicht der Autor). Nur die wesentlichen Anmerkungen diskutieren. Protokollant dokumentiert alle Aussagen und erstellt abschließende Ergebnisdokumentation. Ergebnis muss einstimmig im Gesamturteil gefällt und von allen Personen getragen und unterzeichnet werden. Eventuelle Sondervoten können zu Protokoll gegeben werden.

Es ist nicht Aufgabe des technischen Reviews, über die Konsequenzen des Ergebnisses zu entscheiden.

Inspektion (1)

Vorgeschriebener formaler, durch Regeln definierter Ablauf:

- Viel formaler als ein Walkthrough.
- Ziele bei Planung festgelegt.
- Wahlweise auch zur Prozessverbesserung.
- Jede beteiligte Person hat eine vorgeschriebene Rolle.
- Formaler Prozess für Folgeaktivitäten.

Gewöhnlich Prüfung durch gleichgestellte Mitarbeiter.

Hauptzweck: Fehlerzustände (oder sogar Fehlerursachen) finden !

Begrenzte Anzahl von Aspekten pro Gutachter (Inspektor).

Prüfkriterien zu den einzelnen Aspekten meist in Form von Checklisten mit definierten Eingangs- und Ausgangskriterien für die einzelnen Prüfschritte.

Beinhaltet oft Metriken zur Bewertung des Produkts.

Inspektion (2)

Sitzung durch geschulten Moderator (nicht Autor) geleitet.

- Beginnt mit einer kurzen Einführung, ein Gutachter trägt in einer straffen und möglichst logischen Weise die Inhalte des Prüfobjektes vor; wenn sinnvoll, auch Passagen vorlesen.
- Gutachter stellen Fragen und diskutieren die ausgewählten Aspekte der Prüfung intensiv.
- Autor beantwortet an ihn gerichtete Fragen, bleibt aber ansonsten passiv.
- Moderator greift beim Abschweifen der Diskussion ein.
- Alle festgestellten Unstimmigkeiten und Fehlerzustände werden vom Protokollant erfasst.
- Am Ende der Sitzung werden alle aufgezeichneten Fehler(zustände) vorgestellt und von allen Beteiligten auf Vollständigkeit geprüft.

Inspektionsergebnisse sollen neben der Qualitätsbeurteilung des untersuchten Dokuments auch Bewertungen des Entwicklungs- und Inspektionsprozesses enthalten.

Ein **Prüfliste** mit häufigen Fehlern sollte als Grundlage für die Inspektion dienen.

- Fehler-Prüflisten sind Programmiersprachen-abhängig und reflektieren die charakteristischen Fehler, die in dieser Sprache häufig auftreten.
- Generell gilt: Je weniger die Eingabe überprüft wird, desto länger ist die Prüfliste.
- Beispiele: Initialisierung, konstante Namensgebung, Schleifenabbruchsbedingungen, Arraygrenzen, etc..

Prüfliste einer Inspektion (1)

Fehlerklasse	Inspektionsaktionen
Datenfehler	• Wurden alle Variablen initialisiert, bevor ihre Werte benutzt werden ? • Wurden alle Konstanten benannt ? • Sollte die obere Grenze des Arrays gleich der Größe des Arrays sein oder um 1 kleiner ? • Wenn 'char strings' benutzt werden, wurde ein Trennsymbol explizit angegeben ? • Gibt es die Möglichkeit eines 'buffer overflows' ?
Kontrollfehler	• Für jede bedingte Anweisung: Ist die Bedingung korrekt ? • Ist bei jeder Schleife sichergestellt, dass sie terminiert ? • Ist bei geklammerten Bedingungen die Klammerung korrekt ? • Bei Fallunterscheidungen: Ist jeder Fall abgedeckt ? • Wenn 'breaks' bei den Fallunterscheidungen benötigt werden: Wurden sie alle gesetzt ?
Ein-/Ausgabe-Fehler	• Wurden alle Eingabevariablen benutzt ? • Werden allen Ausgabevariablen ein Wert zugewiesen, bevor sie ausgegeben werden ? • Können unerwartete Eingaben Verfälschungen herbeiführen ?

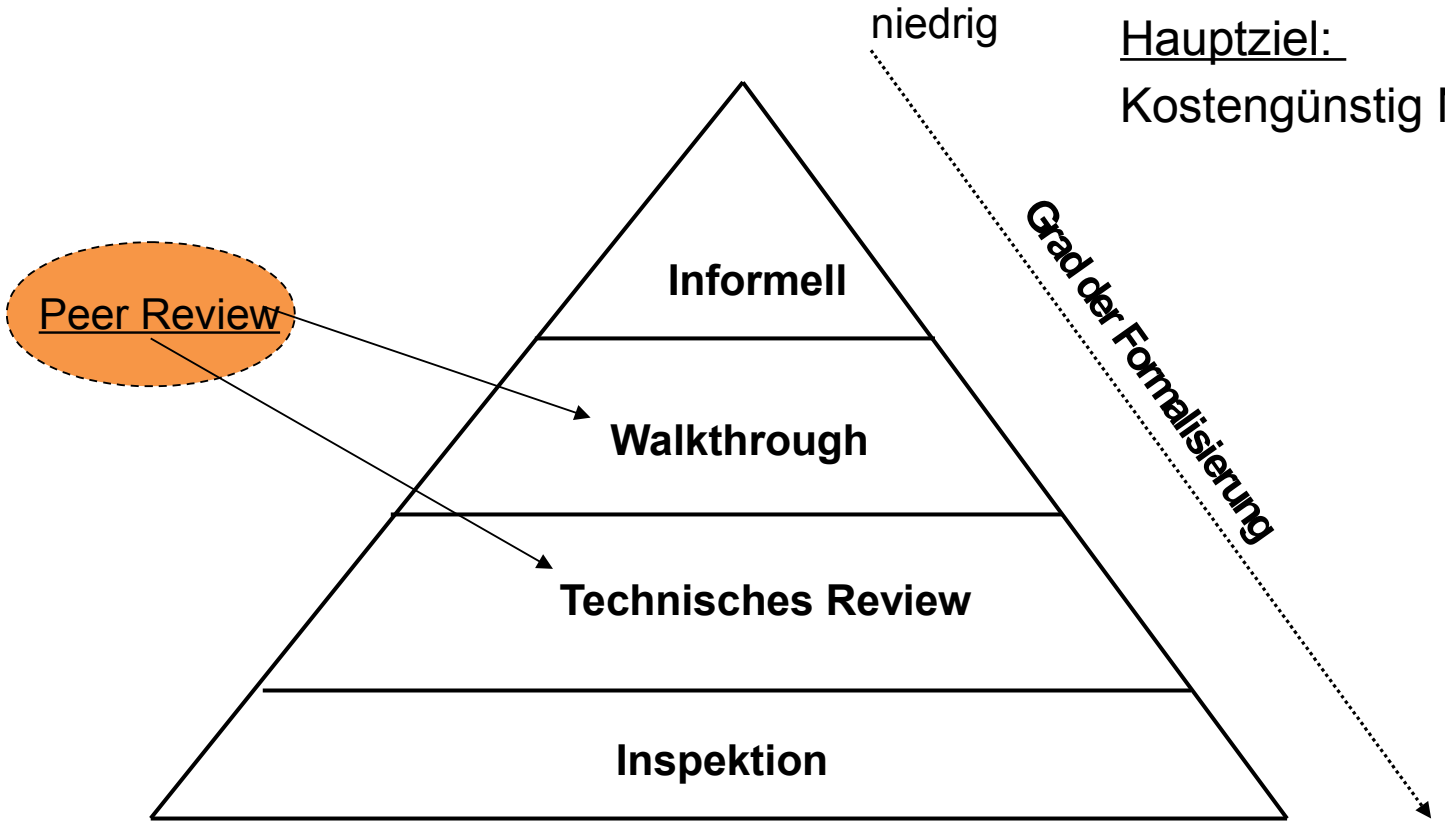
Prüfliste einer Inspektion (2)

Fehlerklasse	Inspektionsaktionen
Interfacefehler	• Haben alle Funktions- und Methodenaufrufe die korrekte Anzahl an Parametern ? • Sind die Parametertypen korrekt ? • Ist die Reihenfolge der Parameter richtig ? • Wenn Komponenten gemeinsamen Speicher benutzen, verwenden sie dann die gleiche Speicherstruktur ?
Speicher-verwaltungs-fehler	• Wenn eine verkettete Struktur modifiziert wird, wurden alle Verkettungen korrekt neu gesetzt ? • Wenn dynamischer Speicher benutzt wird, wurde der Speicher korrekt reserviert ? • Wird der Speicher wieder korrekt freigegeben, wenn er längere Zeit nicht benutzt wird ?
Ausnahme-behandlungs-fehler	• Wurden alle möglichen Fehlerbedingungen in Betracht gezogen und behandelt ?

- Agile Prozesse verwenden selten formale Inspektionen oder paarweises Überprüfen.
- Vielmehr bauen sie auf Teammitglieder auf, die kooperativ gegenseitig ihren Code überprüfen und informelle Richtlinien wie „check before check-in“. Die Programmierer sollen also auch ihren eigenen Code überprüfen.
- Extreme-Programming-Verfechter argumentieren, dass paarweises Programmieren ein effektiver Ersatz für Inspektionen sind, da im Endeffekt eine ständige Überprüfung stattfindet.
- Zwei Menschen überprüfen jede Zeile Code, bevor sie akzeptiert wird.

Review-Arten (Zusammenfassung)

Review-Arten nach IEEE 1028



Kein formaler Prozess, keine/wenige Vorgaben an Gutachter

Hauptziel:

Kostengünstig Nutzen erzielen

Einbindung eines Review-Teams, dokumentierte Ergebnisse und Abläufe

Hauptziele:

- Fehlerzustände finden.
- Lernen, Verständnis erwerben.

Reviews: Auswahlkriterien

Erlaubt der Formalisierungsgrad des Prüfobjektes werkzeuggestützte Analysen ?

- **Ja** ⇒ Zunächst Werkzeuge einsetzen. **Nein** ⇒ Eine der Review-Arten einsetzen.

Wie ist der geforderte Formalisierungsgrad des Prüfungsergebnisses ?

- Ausführliche formalisierte Dokumentation ⇒ Inspektion oder technisches Review
- Undokumentierte Umsetzung der Prüfergebnisse ⇒ Walkthrough oder Informelles Review

Ist Fachwissen aus mehreren Gebieten erforderlich ?

- **Ja** ⇒ Technisches Review. **Nein** ⇒ Walkthrough oder Inspektion.

Wie viel Fachwissen über das Prüfobjekt benötigen die Gutachter ?

- **Viel** ⇒ Zunächst Walkthrough durchführen. **Wenig** ⇒ Direkt Inspektion / technisches Review.

Terminkoordination einfach oder schwierig ?

- **Einfach** ⇒ Inspektion / Technisches Review. **Schwierig** ⇒ Walkthrough / Informelles Review.

Steht der Vorbereitungsaufwand in einem angemessenen Verhältnis zum erwarteten Ergebnis?

- **Ja** ⇒ Inspektion oder Technisches Review. **Nein** ⇒ Walkthrough oder Informelles Review.

Wie hoch ist das Engagement der vorgesehenen Gutachter?

- **Hoch** ⇒ Inspektion oder Technisches Review. **Niedrig** ⇒ Walkthrough (oder: kein Review !)

Reviews: Vorteile

Werden Fehler(zustände) frühzeitig erkannt und beseitigt, führt dies zu einer Produktivitäts-steigerung in der Entwicklung und im dynamischen Test (s. Abschnitt 4.4), da **weniger Fehlerzustände** im Testobjekt vorhanden sind und weniger Ressourcen für die **Fehler-erkennung** und **-beseitigung** benötigt werden. Es ergeben sich oft kürzere Entwicklungszeiten.

Reduzierte Fehlerhäufigkeit im Einsatz des Systems lässt eine **Kostenreduzierung** während der **Produkt-Lebenszeit** erwarten:

- Ungenauigkeiten der Kundenwünsche in den Anforderungen werden oft durch Reviews aufgedeckt und können sofort geklärt werden.
- Die Anzahl der Änderungswünsche nach der Inbetriebnahme des Softwaresystems kann somit verringert werden.

Da die Überprüfungen im Team durchgeführt werden, führt dies zu einem **Wissensaustausch** unter den beteiligten Personen, was die Arbeitsmethoden der einzelnen Personen verbessert und somit auch die Qualität der nachfolgenden Produkte dieser Personen. Gegenseitiges Lernen und mögliche Prozessverbesserungen sind also weitere positive Auswirkungen.

Da mehrere Personen an einem Review beteiligt sind, ist eine klare und verständliche Darstellung der Sachverhalte notwendig. Oft bringt bereits der Zwang zu einer klaren Darlegung den Autor zu **Einsichten**, die er vorher nicht hatte. Die Verantwortung für die Qualität des untersuchten Prüfobjekts trägt das Team.

Reviews kosten und sparen!

- Kosten für Reviews werden mit 10%-15% des Entwicklungsbudgets veranschlagt.
- Einsparungen sind zwischen 14% und 25% möglich, wobei die Mehrkosten für die Reviews schon eingerechnet sind.
- Bei konsequenter Nutzung lassen sich bis zu 70% der Fehler in einem Dokument finden.
- Reduzierung der Fehlerkosten bis zu 75%.

Zahlenwerte aus:

• Gilb, T.; Graham, D.: Software Inspections. Addison-Wesley, 1996
• Bush, M.: Software Quality: The use of formal inspections at the Jet Propulsion Laboratory. In: Proc. 12th ICSE, IEEE 1990, 196-199
• Frühauf, K.; Ludewig, J,; Sandmayr, H.: Software-Prüfung: eine Fibel. vdf, Verlag der Fachvereine, Zürich, 4. Aufl. 2000

Reviews: Fallstricke

Nicht ausreichend Personal mit der erforderlichen Ausbildung vorhanden: Schulungsmaßnahmen durchführen oder entsprechendes Personal von Consulting-Firmen ausleihen (besonders für Moderator: eher psychologische als fachbezogene Fähigkeiten).

Fehlende Zeit (Fehleinschätzung bei der Ressourcen-Planung): Evtl. kann eine weniger aufwändige Review-Art Abhilfe schaffen.

Fehlende Vorbereitung: Andere Gutachter auswählen. Bei fehlender Einsicht der Gutachter in die Bedeutung der Reviews und ihren hohen Wirkungsgrad zur Qualitätssteigerung der untersuchten Dokumente ist dies mit entsprechendem Zahlenmaterial zu belegen.

Fehlende oder unklare Ziele (Fehler(zustände) finden ? Prozess verbessern ? Menschen „anschwärzen“ ?): Ziele während der Reviewplanung festlegen, die durch ein Review erreicht werden sollen (vor der Reviewdurchführung !).

Fehlende oder unzureichende Dokumentation: Vorab prüfen, ob alle benötigten Dokumente in ausreichender Beschreibungstiefe vorhanden sind. Nur wenn das der Fall ist, ist ein Review durchzuführen.

Fehlende Unterstützung durch das Management (in der Praxis leider oft): Reviewprozess nicht erfolgreich, wenn die benötigten Ressourcen nicht bereitgestellt und die erzielten Ergebnisse nicht zur Prozessverbesserung genutzt werden.

Reviews in der Praxis

- Reviews als Methode akzeptiert.
- Die einzelne Durchführung oft wenig systematisch.
- Kaum spezielle Schulung für Software-Reviews.
- Spezielle Vorgehensweisen wie Software-Inspektionen werden bei kleineren Unternehmen (weniger als 50 Mitarbeiter in der Software-Entwicklung) deutlich seltener angewandt als bei großen.
- Nur eine kleine Anzahl der Befragten gab an, mit begleitenden Messungen die Qualität der Reviews selbst zu bewerten. (50% sammeln überhaupt keine Daten, 20% Aufwandsdaten, 20% Fehlerdaten.)

Ergebnisse einer Umfrage vom Fraunhofer Institut IESE im Rahmen des ViSEK Kompetenzzentrums im Sommer 2002. 121 Teilnehmer aus Unternehmen aller Branchen mit eigener Softwareentwicklung.

Reviews: Tipps

Dokumente mit formaler Struktur vorab mit einem diese Struktur überprüfenden **Werkzeug** analysieren, das viele Aspekte untersuchen und Fehler(zustände) oder Abweichungen feststellen kann, die dann nicht mehr bei einem Review zu kontrollieren sind.

Gute **Vorbereitung** aller Personen ist für erfolgreiche Reviewsitzung notwendig.

Auswertung der jeweiligen Reviewsitzungen und deren Ergebnisse dazu nutzen, den **Reviewprozess** weiter zu **verbessern** und die verwendeten Dokumente (Richtlinien und Checklisten) den jeweiligen Gegebenheiten anzupassen und aktuell zu halten.

Gefundene Fehler(zustände) positiv aufnehmen und objektiv zur Sprache bringen.

Mit den Fragen der Beteiligten **konstruktiv** umgehen, psychologische Aspekte beachten: Z.B. das Review für den Autor zu einer positiven Erfahrung werden lassen.

Wiederkehrende oder häufig vorkommende Fehlerarten weisen auf **Defizite** im Softwareentwicklungsprozess oder im Fachwissen der jeweiligen Personen hin.

- Notwendige Verbesserung des Entwicklungsprozesses planen und umsetzen.
- Mangel an Fachwissen durch Schulung ausgleichen.

IEEE 1028-1997 ist der „IEEE Standard for Software Reviews“.

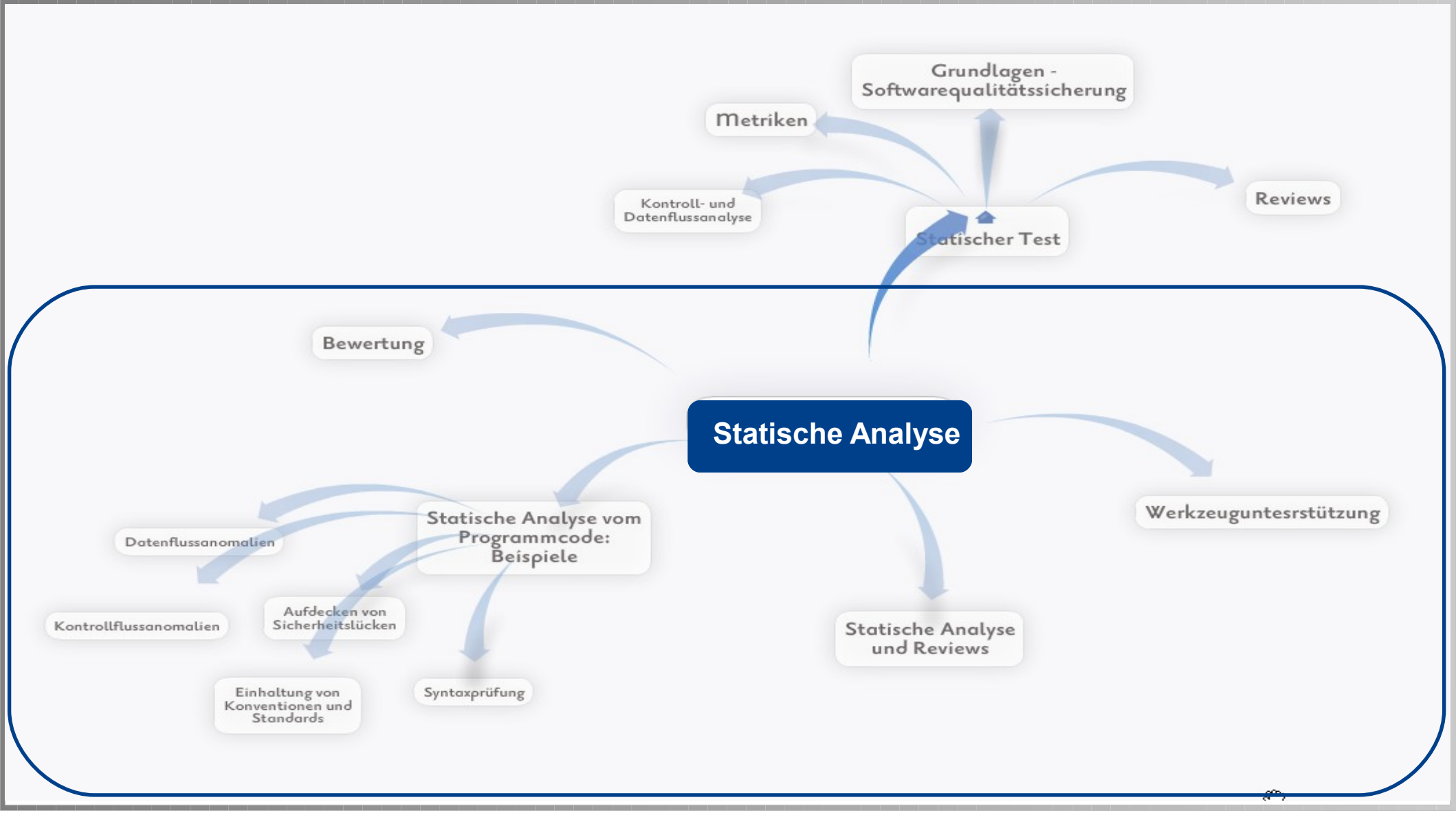
- Orientieren Sie sich an diesem Standard !

4.3 Statischer Test

4.3 Statischer Test

- Grundlagen
- Reviews
- **Statische Analysen**
- Kontroll- und Datenflussanalyse
- (Metriken: Abschnitt 3.2)

Statische Analyse



Statische Analysen

Ziel: Aufdeckung vorhandener Fehlerzustände oder fehlerträchtiger Stellen in einem Dokument.

- »Statische Analyse«: auch diese Form der Prüfung beinhaltet keine Ausführung der Prüfobjekte (eines Programms).

Beispiele:

- Rechtschreibprüfprogramme als eine Art statische Analysatoren, die (Rechtschreib- und – eingeschränkt – Grammatik-) Fehler in Texten nachweisen und damit zur Qualitätsverbesserung beitragen.
- Compiler führen eine statische Analyse des Programmtextes durch, indem sie die Einhaltung der Syntax der Programmiersprache prüfen.
- Spezielle Werkzeuge (eine Art erweiterter Compiler) prüfen die Einhaltung von Programmierkonventionen.

Weiteres Ziel: Ermittlung von Messgrößen oder Metriken, um eine Qualitätsbewertung durchführen und somit Qualität messen zu können.

Statische Analyse ist nur mit Werkzeugunterstützung sinnvoll !

Das zu analysierende Dokument muss nach einem vorgegebenen Formalismus aufgebaut sein, also einer formalen Struktur unterliegen, um durch ein Werkzeug überprüft werden zu können.

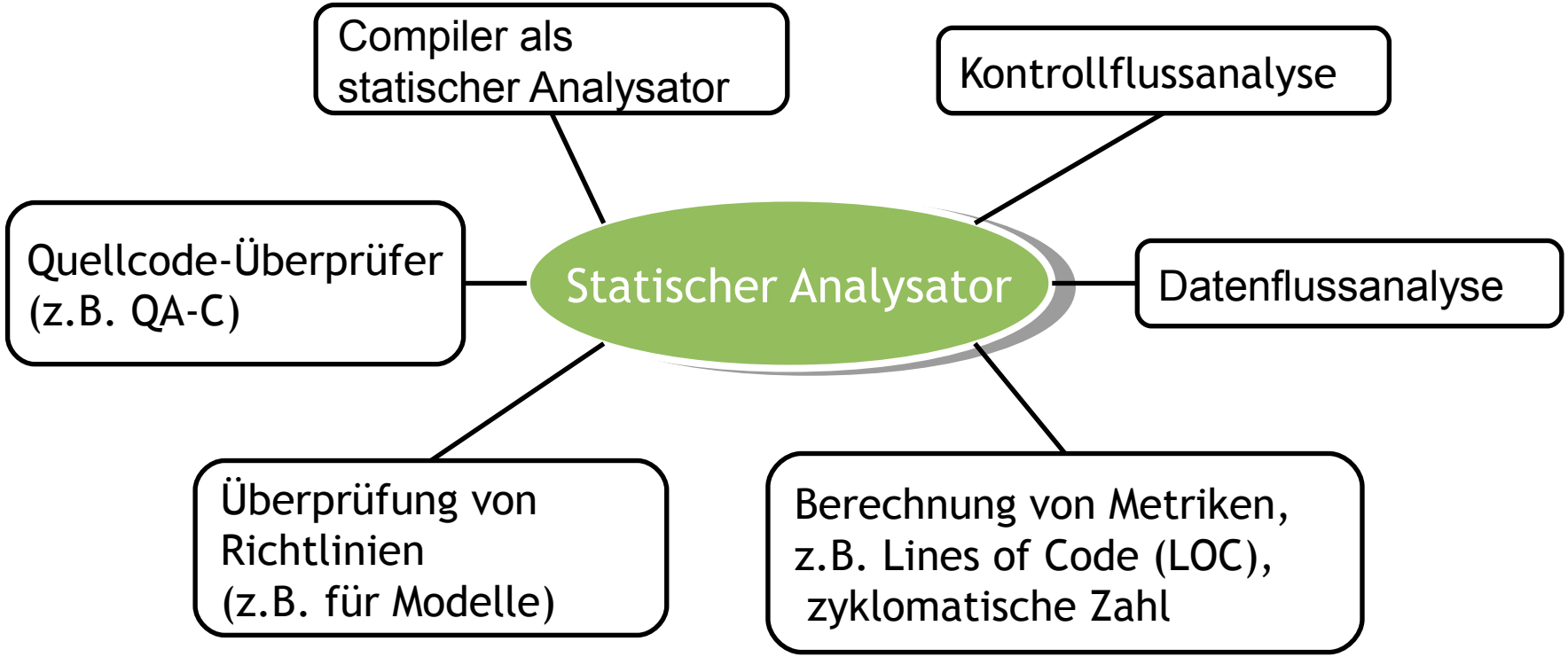
Dokumente, die einem Formalismus unterliegen können, sind z. B.:

- Technische Anforderungen
- Softwarearchitektur
- Softwareentwurf

Informeller Text kann unterhalb der Oberflächenstruktur (Rechtschreibung und elementare Grammatik) nur mit Methoden der KI (künstliche Intelligenz) analysiert werden.

- Linguistische semantische Analyse ist aktueller Forschungsgegenstand.
- Aber: Einführung einer »Normsprache« ermöglicht auch hier einfachere Analysen.

Elemente einer werkzeug-gestützten statischen Analyse



Stehen in einem engen Zusammenhang:

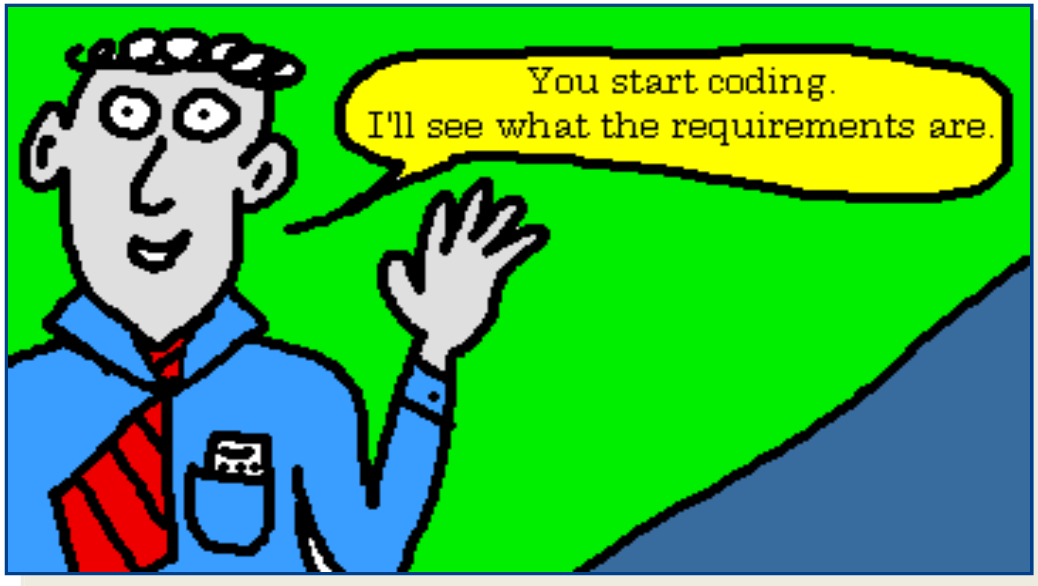
- Wird vor dem Review eines formalen Dokumentes eine statische Analyse durchgeführt, können bereits eine Anzahl von Fehler(zustände)n und Unstimmigkeiten nachgewiesen werden und die Menge der im Review zu berücksichtigenden Aspekte ist erheblich geringer.
- Da statische Analysen werkzeuggestützt durchgeführt werden, ist der Aufwand wesentlich niedriger als beim Review.

Also:

1. Statische Analyse und
2. Review

Statische Analyse in der Praxis

- Weit verbreitete Praxis der Software-Entwicklung:



- Folge: Leider ist der Programmcode oft das erste (und einzige) formale Dokument der Softwareentwicklung, das einer statischen Analyse unterzogen werden kann.
- Frage: Stellt das ein Problem dar ?

Alle Compiler führen eine statische Analyse des Programmtextes durch, indem sie die Einhaltung der Syntax der jeweiligen Programmiersprache prüfen. Die meisten Compiler bieten darüber hinaus noch zusätzliche Informationen, die durch statische Analyse ermittelt werden.
Neben den Compilern gibt es Werkzeuge, sogenannte Analysatoren, die speziell zur gezielten Durchführung einzelner oder ganzer Gruppen von Analysen eingesetzt werden.

Fehler(zustände) bzw. fehlerträchtige Situationen, die mit der statischen Analyse von Programmcode nachgewiesen werden können, sind beispielsweise:

- Verletzung der Syntax
- Abweichungen von Konventionen und Standards
- Sicherheitslücken
- Kontrollflussanomalien
- Datenflussanomalien

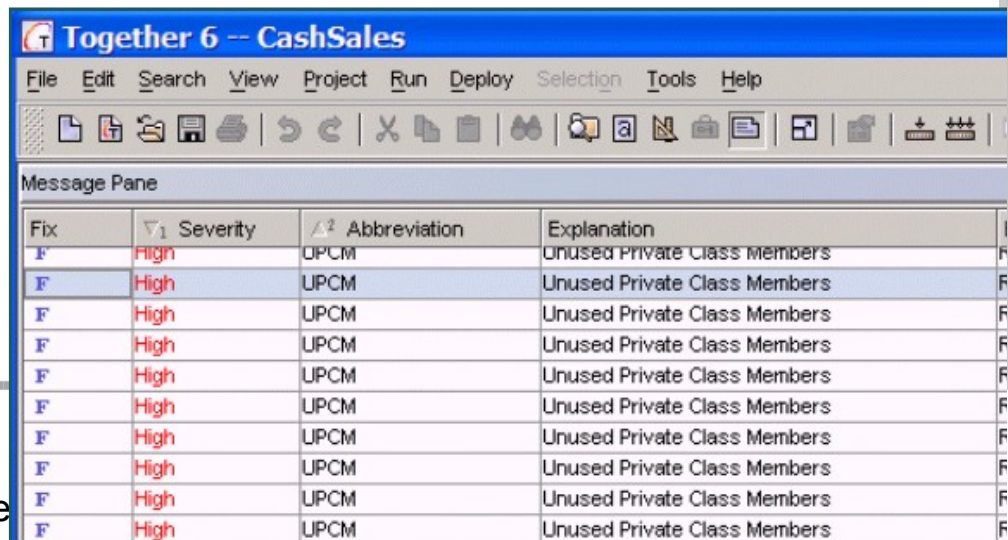
Syntaxprüfung

Verletzung der Syntax der Programmiersprache als Fehler oder Warnung gemeldet.
Viele Compiler führen darüber hinaus weitere Überprüfungen durch:

- Verwendung der einzelnen Programmelemente (Variablen, Funktionen, ...)
- Prüfung der typgerechten Verwendung der Daten und Variablen bei streng typisierten Programmiersprachen (C, C++, Java)
- Ermittlung von nicht deklarierten Variablen
- Unerreichbarer Code (»toter Code«)
- Fehler bei Über- oder Unterschreitung von Feldgrenzen
- Prüfung der Konsistenz von Schnittstellen
- Prüfung der Verwendung aller Marken als Sprunganweisung und Sprungziel

Die Ergebnisse werden meist in Form von Listen zur Verfügung gestellt.

Nicht immer kann anhand der Ergebnisse sofort auf einen Fehlerzustand geschlossen werden, es sind dann weitere Untersuchungen anzuschließen.



Einhaltung von Konventionen und Standards

Einhaltung der Programmier- oder Modellierkonventionen durch entsprechende Analysatoren nachweisbar:

- Überprüfung benötigt wenig Zeit und so gut wie keine Personalressourcen.
- Darüber hinaus ist meist ein weiterer Vorteil gegeben: Ist den Programmierern bzw. Software-Designern bewusst, dass der Programmcode bzw. die Modelle von einem Werkzeug auf Einhaltung der Richtlinien überprüft wird, ist deren Bereitschaft zu ihrer Umsetzung bedeutend höher als ohne eine automatische Prüfung.

Tipp: Nur solche Richtlinien in einem Projekt zulassen, zu denen Prüfwerkzeuge vorhanden sind (andere Vorschriften erweisen sich in der Praxis ohnehin schnell als bürokratischer Ballast).

Aber: Werkzeuge für statische Analyse können große Mengen von Warnungen und Hinweisen erzeugen, die gut verwaltet werden müssen, um eine effektive Nutzung des Werkzeugs zu erlauben.

Aufdecken von Sicherheitslücken

Durch die Verwendung bestimmter fehleranfälliger Programmkonstrukte und durch fehlende notwendige Überprüfungen treten viele Sicherheitsprobleme auf.

Beispiele:

- Fehlendes Abfangen von Speicherüberläufen
- Keine Überprüfung des Einhaltens von Datenbeschränkungen bei der Eingabe

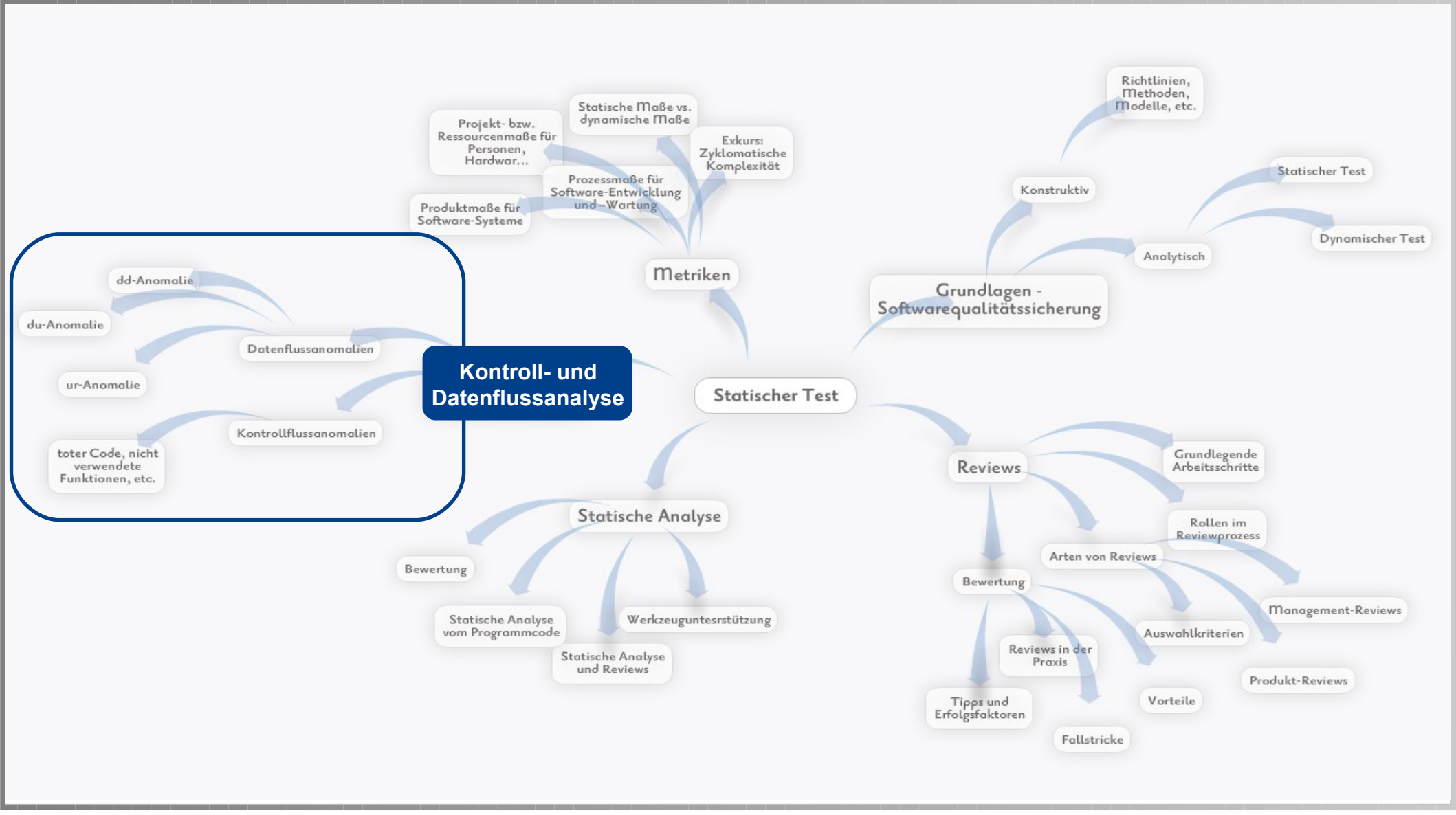
Analysewerkzeuge können diese Mängel aufdecken, da diese Mängel meistens einem bestimmten „Muster“ unterliegen, das von den Werkzeugen gesucht und analysiert werden kann.

4.3: Statischer Test

4.3 Statischer Test

- Grundlagen
- Reviews
- Statische Analysen
- **Kontroll- und Datenflussanalyse**
- (Metriken: Abschnitt 3.2)

Kontroll- und Datenflussanalyse



Suche nach Anomalien im Programmtext

Anomalie: Unstimmigkeit, die zu einer Fehlerwirkung führen kann, aber nicht zwingend dazu führt.

- Anomal: unregelmäßig, regelwidrig (laut Lexikon)
- Kann ein Fehlerzustand sein, muss es aber nicht.

Mit der statischen Analyse lassen sich nicht alle Fehlerzustände und Unstimmigkeiten nachweisen: Es gibt Fehlerzustände, die erst bei der Ausführung, also zur Laufzeit des Programms, als Fehlerwirkung auftreten und vorher nicht ermittelbar sind.

- Wird z.B. bei einer Division der Wert des Divisors in einer Variablen gehalten, so kann diese Variable zur Laufzeit den Wert Null annehmen, was zu einer Fehlerwirkung führt.
- Statisch ist dieser Fehlerzustand meist nicht erkennbar.

Kontrollfluss

Kontrollfluss: Abstrakte Repräsentation von allen möglichen Reihenfolgen von Ereignissen während einer Programmausführung.

- Abfolge der während eines Programmlaufs ausgeführten Anweisungen.
- Muss nicht die Reihenfolge der Anweisungen im Programmtext sein.

Wird durch »steuernde« Anweisungen determiniert:

- Unbedingte Sprünge
- Bedingte Verzweigungen
- Schleifen

Der Wahrheitswert von Bedingungen, z.B. in IF-Anweisungen im Programmtext, steuert den Kontrollfluss.

- Ist der ermittelte Wahrheitswert der Bedingung »wahr«, dann wird die Ausführung des Programms mit dem THEN-Teil der IF-Anweisung fortgeführt, im anderen Fall (»unwahr«) wird der ELSE-Teil durchlaufen.

Schleifen führen zu vorherigen Anweisungen zurück, wodurch deren mehrfache Ausführung bewirkt wird, oder der Schleifenkörper wird umgangen.

Kontrollflussgraph

Gerichteter Graph

Knoten: Anweisungen des Programms

- Sequenzen von Anweisungen können dann als ein einziger Knoten dargestellt werden (Block, Segment), wenn die Sequenz nur durch ihren ersten Knoten betreten werden kann und es innerhalb der Sequenz zu keiner Änderung des Programmablaufs kommen kann, d.h. genau dann und nur dann, wenn die erste Anweisung der Sequenz ausgeführt wird, werden auch alle weiteren Anweisungen der Sequenz ausgeführt.

Kanten: Mögliche Ausführungsreihenfolgen der durch die Knoten repräsentierten Anweisungen:

- Knoten mit mehreren ausgehenden Kanten repräsentieren den Kontrollfluss steuernde Anweisungen
- IF-Anweisung und Schleifensteuerung: Zwei abgehende Kanten
- CASE-Anweisung: mehrere abgehende Kanten

Genau ein Startknoten (Programmanfang, Kopf der Methode)

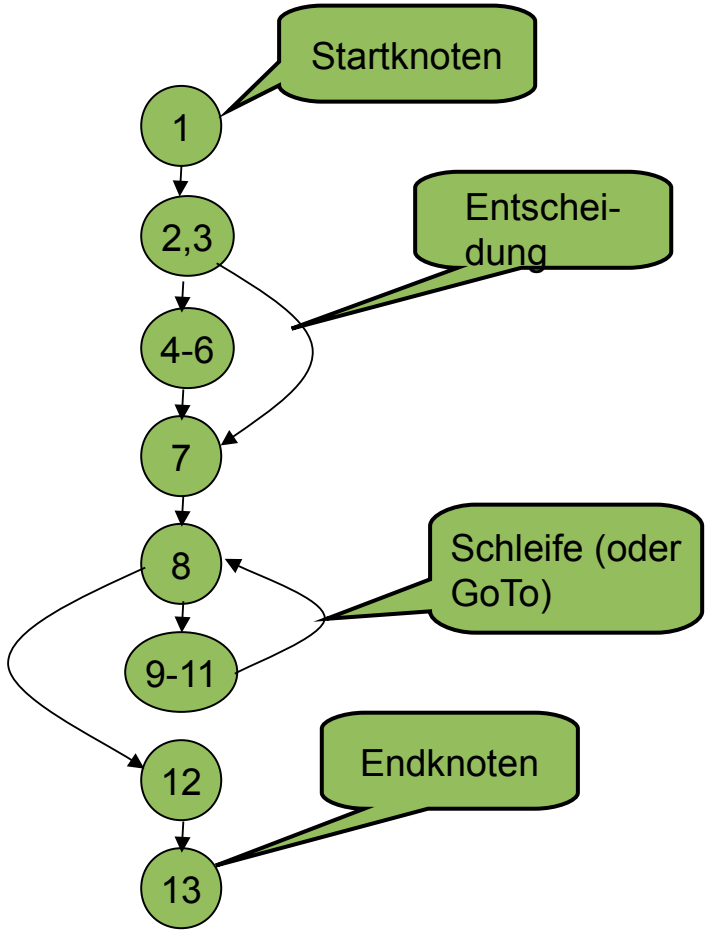
Genau ein Endknoten (Programmende, Ende der Methode)

Beispiel eines Kontrollflussgraphen

Bestimmung des größten gemeinsamen Teilers (ggT) zweier ganzer Zahlen m und n:

```
1.  public int ggt(int m, int n) {

2.      int r;
3.      if (n > m) {
4.          r = m;
5.          m = n;
6.          n = r;
        }
7.      r = m % n;
8.      while (r != 0) {
9.          m = n;
10.         n = r;
11.         r = m % n;
        }
12.     return n;
13. }
```



Durch die Anschaulichkeit des Kontrollflussgraphen lassen sich die Abläufe durch ein Programmstück leicht erfassen.

- Sind Teile des Graphen oder der ganze Graph sehr unübersichtlich und die Zusammenhänge und der Ablauf kaum nachvollziehbar, sollte eine Überarbeitung des Programmtextes erfolgen, da komplexe Ablaufstrukturen oft fehlerträchtig sind.

Kontrollflussanomalie: Statisch feststellbare Unstimmigkeit beim Ablauf des Testobjekts (z.B. nicht erreichbare Anweisungen):

- Sprünge aus Schleifen heraus
- Sprünge in Schleifen hinein
- Programmstücke mit mehreren Ausgängen

Diese Unstimmigkeiten müssen keine Fehlerzustände sein, widersprechen aber den Grundsätzen der strukturierten Programmierung.

Voraussetzung: Graph nicht manuell, sondern von einem Werkzeug erstellt, das eine eins-zu-eins–Abbildung zwischen Programmtext und Graph gewährleistet.

Kontrollflussanalyse: Vorgänger-Nachfolger-Tabellen

Neben den Graphen können auch Vorgänger-Nachfolger-Tabellen erstellt werden, aus denen hervorgeht, ob und wie eine Anweisung mit anderen Anweisungen in Beziehung steht.

Gibt es zu einer Anweisung keinen Vorgänger, so ist diese Anweisung nicht erreichbar (sogenannter »toter Code«).

- Ein Fehlerzustand oder zumindest eine Unstimmigkeit ist erkannt.
- Nur für die erste und letzte Anweisung eines Programm(teil)s gelten die Ausnahmen, dass es keine Vorgänger- bzw. Nachfolge-Anweisung geben darf.
- Für Programm(teil)e mit mehreren Eintritts- bzw. Austrittspunkten gilt entsprechendes.

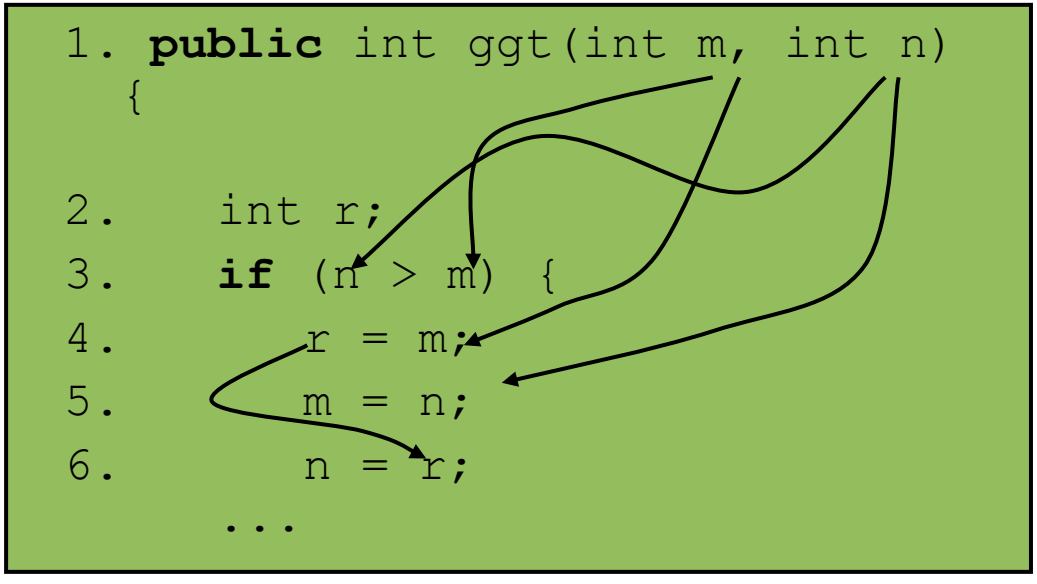
Beispiel: Vorgänger-Nachfolger-Tabelle


```
1.  public int ggt(int m, int n) {

2.     int r;
3.     if (n > m) {
4.        r = m;
5.        m = n;
6.        n = r;
       }
7.     r = m % n;
8.     while (r != 0) {
9.        m = n;
10.       n = r;
11.       r = m % n;
       }
12.    return n;
13. }
```

Anweisung	Nachfolger	Vorgänger
1	2	-
2	3	1
3	4, 7	2
4	5	3
5	6	4
6	7	5
7	8	3, 6
8	9, 12	7, 11
9	10	8
10	11	9
11	8	10
12	13	8
13	-	12

Verwendung von Daten auf »Pfaden« durch den Programmcode.



Nicht immer können Fehlerzustände nachgewiesen werden, oft wird in diesem Zusammenhang dann ebenfalls von Anomalien oder Datenflussanomalien gesprochen:

- Lesen einer Variablen ohne vorherige Initialisierung oder
- Nicht-Verwendung eines zugewiesenen Wertes einer Variablen

Datenfluss-Anomalien

Analysiert wird die Verwendung jeder einzelnen Variablen.

Drei Verwendungen oder Zustände der Variablen werden unterschieden:

- Undefiniert (**u**): Die Variable hat keinen definierten Wert (z.B. am Programm-Beginn oder wenn sie freigegeben oder ihr Gültigkeitsbereich verlassen wird).
- Definiert (**d**): Der Variablen wird ein Wert zugewiesen.
- Referenziert (**r**): Der Wert der Variablen wird gelesen bzw. verwendet.

Drei Arten von Datenflussanomalien:

- **ur**-Anomalie: Ein undefinierter Wert (u) einer Variablen wird auf einem Programmpfad gelesen (r).
- **du**-Anomalie: Die Variable erhält einen Wert (d) der allerdings ungültig (u) wird, ohne dass er zwischenzeitlich verwendet wurde.
- **dd**-Anomalie: Die Variable erhält auf einem Programmpfad ein zweites Mal einen Wert (d), ohne dass der erste Wert (d) verwendet wurde.

Zusammengefasst: Eine Datenfluss-Anomalie ist die:

- referenzierende Verwendung einer Variablen ohne vorherige Initialisierung
- oder die Nicht-Verwendung eines Wertes einer Variablen.

Beispiel: Datenfluss-Anomalien

```
1.  void Tausch (int min, int max) {   // d(min, max)
2.    int hilf;                        // u(hilf) (in Java hilf=0!!)
3.    if (min > max) {                 // r(min, max)
                                       // d(max), r(hilf)
4.      max  = hilf;                   // d(max), r(min)
5.      max  = min;                    // d(hilf), r(min)
6.      hilf = min;                    // u(hilf)
7.    }
8.  }
```

Welche Anomalien sehen Sie ?

Beispiel: Datenfluss-Anomalien

```
1.  void Tausch (int min, int max) {   // d(min, max)
2.    int hilf;                        // u(hilf) (in Java hilf=0!!)
3.    if (min > max) {                 // r(min, max)
4.        max  = hilf;                 // d(max), r(hilf)
5.        max  = min;                  // d(max), r(min)
6.        hilf = min;                  // d(hilf), r(min)
7.    }
8.  }                                  // u(hilf)
```

ur (hilf)

ur-Anomalie der Variablen `hilf`

- Der Gültigkeitsbereich der Variablen ist auf die Funktion beschränkt. Die erste Verwendung der Variablen ist auf der rechten Seite einer Zuweisung. Die Variable hat zu diesem Zeitpunkt noch einen undefinierten Wert, der an dieser Stelle referenziert wird. Eine Initialisierung bei der Deklaration der Variablen wurde nicht vorgenommen.

Beispiel: Datenfluss-Anomalien

```
1.  void Tausch (int min, int max) {   // d(min, max)
2.    int hilf;                        // u(hilf) (in Java hilf=0!!)
3.    if (min > max) {                 // r(min, max)
4.      max  = hilf;                   // d(max), r(hilf)
5.      max  = min;                    // d(max), r(min)
6.      hilf = min;                    // d(hilf), r(min)
7.    }
8.  }                                  // u(hilf)
```

dd (max)

dd-Anomalie der Variablen `max`

- Die Variable wird hintereinander jeweils auf der linken Seite einer Zuweisung verwendet, bekommt somit zweimal einen Wert zugewiesen. Entweder kann die erste Zuweisung entfallen, oder eine Verwendung des ersten Wertes ist vergessen worden (oder es wurden - wie hier - Variablennamen und Reihenfolgen vertauscht).

Beispiel: Datenfluss-Anomalien

```
1.  void Tausch (int min, int max) {   // d(min, max)
2.    int hilf;                        // u(hilf) (in Java hilf=0!!)
3.    if (min > max) {                 // r(min, max)
4.        max  = hilf;                 // d(max), r(hilf)
5.        max  = min;                  // d(max), r(min)
6.        hilf = min;                  // d(hilf), r(min)
7.    }
8.  }                                  // u(hilf)
```

du (hilf)

du-Anomalie der Variablen `hilf`

- In der letzten Anweisung der Funktion bekommt die Variable `hilf` noch einen Wert zugewiesen, der nirgends verwendet werden kann, da die Variable nur innerhalb der Funktion gültig ist.

Im Beispiel sind die Anomalien sehr offensichtlich.

Aber: Zwischen den jeweils betroffenen Anweisungen, die zu den Anomalien führen, können beliebig viele andere Anweisungen mit anderen Variablen stehen.

- Anomalien dann nicht mehr so offensichtlich.
- Können bei einer manuellen Prüfung leicht übersehen werden.
- Werkzeug zur Datenflussanalyse deckt die Anomalien auf.

Nicht jede Anomalie führt direkt zu fehlerhaftem Verhalten:

- Beispielsweise hat eine du-Anomalie nicht immer direkte Auswirkungen, das Programm kann korrekt laufen.
- Es stellt sich nur die Frage, warum die Zuweisung an dieser Stelle des Programms vor dem Ende des Gültigkeitsbereichs der Variablen vorhanden ist.
- Meist lohnt sich eine genauere Untersuchung der anomalen Programmstellen, um weitere, semantisch tiefsinnigere Unstimmigkeiten ausfindig zu machen.

Symbolische Ausführung

Erweiterung des Begriffs der **Ausführung** eines Programms mit konkreten Werten:

- Statt konkreter Werte werden **symbolische Werte** für die Eingabevariablen verwendet.
- Mit diesen wird dann entsprechend ‚gerechnet'.

Beispiel

Symbolische Ausführung der Anweisung:

C := A + 2 B.

Die symbolischen Werte seien mit w() bezeichnet. Vor Ausführung der Anweisung gelte w(A) = a und w(B) = b. Dann ergibt die symbolische Ausführung der obigen Zuweisung:

w(C) = w(A) + 2 w(B) = a + 2 b.

Symbolische Ausführung - Vorteile

- Es ist keine formale Programmspezifikation erforderlich.
- Ein symbolischer ‚Test' deckt eine Vielzahl normaler Testdaten ab.
- Insbesondere werden Fehler entdeckt, bei denen für eine Teilmenge der Eingaben die Ergebnisse falsch berechnet werden.

Symbolische Ausführung - Nachteile

- Ergebnisse sind schwieriger als beim konventionellen Testen zu überprüfen, da die symbolischen Ausdrücke mit der Spezifikation verglichen werden müssen.
- Die verwendete Programmiersprache muss formal definiert sein, damit ein symbolischer Interpreter arbeiten kann.
- Als alleinige Testmethode ist die symbolische Ausführung nicht ausreichend, da ein funktionsorientierter Test fehlt.
- Es muss ein Theorembeweiser zur Verfügung stehen, der alle notwendigen Umformungen erzeugen kann (z.B. $a + 2 \cdot b – a = 2 \cdot b$) und korrekt arbeitet.
- Für hinreichend mächtige Programmiersprachen existiert kein vollständiger automatischer Theorembeweiser (sonst wäre die Äquivalenz von Programmen entscheidbar).
- Daher kann es vorkommen, dass der Theorembeweiser weder w(B) = true noch w(B) = false bei einer Verzweigung mit Prädikat B beweisen kann und daher beide Verzweigungsausgänge gewählt werden, obwohl tatsächlich einer nicht ausführbar ist.

Nutzen statischer Analysen

- Frühe Erkennung von Fehlerzuständen vor der Testdurchführung.
- Frühe Warnung vor verdächtigen Aspekten in Code oder Design, durch Berechnung von Metriken wie z.B. hohes Komplexitätsmaß.
- Identifizierung von Fehlerzuständen, die durch dynamischen Test nicht effektiv und effizient aufzudecken sind.
- Aufdecken von Abhängigkeiten und Inkonsistenzen in Softwaremodellen, wie z.B. redundante oder die Architektur verletzende Links.
- Verbesserte Lesbarkeit, Änderbarkeit und Wartbarkeit von Code und Design.
- Vorbeugung von Fehlerzuständen, falls aus Erfahrung gelernt wird und sich dies in der Entwicklung niederschlägt.

Mit statischer Analysen findbare Fehlerzustände

- Referenzierung einer Variablen mit nicht definiertem Wert
- Inkonsistente Schnittstellen zwischen Modulen und Komponenten
- Variablen, die nie verwendet werden
- Unerreichbarer (toter) Code (»dead code«)
- Verletzung von Programmierkonventionen
- Sicherheitsschwachstellen
- Syntax-Verletzungen von Code und Softwaremodellen
- ...

Zusammenfassung

Der **Reviewprozess** umfasst typischerweise: Planung, Kick-off, individuelle Vorbereitung, Reviewsitzung, Überarbeitung und Nachbereitung

Rollen sind: Manager, Moderator, Autor, Gutachter und Protokollant

Arten von Reviews sind: informelles Review, Walkthrough, technisches Review und Inspektion.

Werkzeuggestützte statische Analysen ohne Ausführung des Prüfobjektes sind für Dokumente mit gewissem Formalisierungsgrad möglich.

Compiler decken Fehlerzustände in der Syntax des Programmcodes auf, bieten aber meist noch weitere Möglichkeiten.

Analysewerkzeuge melden Verstöße gegen Standards und andere Konventionen.

Anomalien im Daten- und Kontrollfluss der Programmtexte weisen oft auf fehlerträchtige Stellen hin.

Nach dieser Vorlesung sollten Sie...

- Arbeitsergebnisse der Softwareentwicklung, die mit den verschiedenen statischen Prüftechniken geprüft werden, erkennen können
- Bedeutung und Nutzen statischer Methoden für die Bewertung von Arbeitsergebnissen der Softwareentwicklung beschreiben können
- den Unterschied zwischen statischem und dynamischem Test erläutern können – unter Beachtung der Ziele, Fehlerarten und der Rolle im Software-Lebenszyklus
- die grundlegende Vorgehensweise bei manuellen statischen Prüftechniken (Reviews) kennen und erläutern können
- unterschiedliche manuelle statische Prüftechniken für unterschiedliche Prüfsituationen auswählen und anwenden können, wobei Unterschiede und Faktoren für die erfolgreiche Durchführung erklärt werden
- automatisierbare statische Analyseverfahren kennen und anhand der identifizierbaren typischen Fehler(zustände) charakterisieren können
- die Kontrollfluss- und Datenfluss-Analyse für Programmcode kennen und voneinander abgrenzen können
- grundlegende Werkzeuge für statische Analysen kennen.

Die Begriffe sollten sie kennen....

Folgende Fragen sollten Sie jetzt beantworten können

- Was sind die grundlegenden Schritte, die bei einem Review durchzuführen sind. (Bitte beschreiben !)
- Welche Reviewarten werden unterschieden ?
- Welche Rollen wirken an einem technischen Review mit ?
- Warum sind Reviews ein effizientes Mittel zur Qualitätssicherung ?
- Was umfasst der Begriff „statische Analyse“ ? Bitte erklären !
- Wie stehen statische Analyse und Reviews in Zusammenhang ?
- Warum kann die statische Analyse nicht alle in einem Programm enthaltenen Fehlerzustände aufdecken ?
- Welche Arten von Datenflussanomalien werden unterschieden ?
- Was sind typische Fehlerzustände im Quellcode und Entwurf, die durch eine werkzeuggestützte statische Analyse identifiziert werden können ?