Softwarequalität und -test

5. Vorlesung

„Testmanagement in Softwareprojekten“

www.beuth-hochschule.de

Dipl.-Inform. Thomas Ziemer

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testkonzept

Die Grundbegriffe des Testens gehören zu den Grundpfeilern der Softwarequalität. Warum Tests notwendig sind, wird unterschiedlich begründet: „Testen ist die Ausführung eines Programms mit dem Ziel, Fehler zu finden“. Ohne das Ziel, möglichst viele, schwere Fehler zu finden, verliert der Test an Biss und Wirkung.

Herumprobieren gilt nicht als Test. Auch nicht, wenn er von sehr kompetenten Leuten gemacht wird. Es kann höchstens die Vorbereitung für einen richtigen Test bilden.

Für den systematischen Test kommt es sogar darauf an, sich *schriftlich* vorzubereiten und die Testfälle *schriftlich* zu dokumentieren.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testkonzept

Das Testkonzept ist die Zusammenfassung aller wesentlichen Festlegungen im Rahmen des folgenden Testprozesses und ist als solches bindend für die spätere Durchführung. Damit wird der Testprozess nachvollziehbar und revisionssicher.

Die Basis für das Testkonzept bildet das erstellte Fachkonzept des Softwareprojekts. Das Testkonzept enthält die im folgenden beschriebenen Vereinbarungen zu:

Testmanagement (Rollen- und Infrastrukturdefinition)
Testplan (Definition von Testinhalten und Abhängigkeiten)
Testvorbereitung (Methodenauswahl für Testfälle und -daten)
Testdurchführung („Nur nicht denken“, „Testen macht Spaß!“)
Testauswertung, Fehlermanagement

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testmanagement

Das Testmanagement nennt die am Testvorgehen beteiligten Personen innerhalb ihrer funktionellen Rollen und definiert die vorgesehene Testumgebung.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testmanagement

Testverantwortliche

Es wird der Personenkreis definiert, der für das Testvorhaben benötigt wird.

Es müssen Eskalationspfade definiert werden, die im Falle einer Ressourcenknappheit und/oder bei anderen auftretenden Problemen beschritten werden können. Damit wird sichergestellt, dass Entscheidungen bzgl. des Projektfortschritts von der entsprechenden Ebene (beispielsweise im Lenkungsausschuss) getroffen werden.

Bei der Auswahl der Personen, die den Test vorbereiten, durchführen und auswerten, müssen die Qualitätsanforderungen mit den tatsächlich vorhandenen Qualitätsmerkmalen der Personen abgeglichen werden.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testmanagement (Fortsetzung)

Rollen

Berater und/oder Coach

- Testplanung und Testmanagement
- Testvorgehen
- Erstellung von Testhandbüchern.
- Einführung von Testmethoden und -techniken
- Unterstützung firmeninterner Mitarbeiter bei der Umsetzung der Testmethoden und -techniken.

Testmanagement in Softwareprojekten

Testmanagement (Fortsetzung)

Rollen

Testfallersteller

Definition von Testfallschablonen mittels entsprechender Methoden.

Testfallschablonen beschreiben beispielsweise Geschäftsprozesse auf der Basis kundenseitig vorgegebener fachlicher Anforderungen.

Testmanagement (Fortsetzung)

Rollen

Tester

Durchführung von Tests auf der Basis von Testprozeduren und von kundenseitig mit entsprechenden Testdaten ergänzten Testfallschablonen (Testdatenschablonen).

Testprozeduren beschreiben beispielsweise im GUI-Test präzise den durchzuführenden Ablauf.

Testplan

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testplan

Der Testplan beschreibt, wie die Testfälle angewendet und wie die Tests durchgeführt werden. Das ist eine Resourcen- und Zeitfrage: jeder Testfall kostet Zeit. Der Testplan legt fest, wer wann mit dem Testen an der Reihe ist, und welche Testfälle welcher Ressource zugeordnet sind. Im Idealfall sollte man den Testplan genauso ernst nehmen wie den Entwicklungsplan.

Wie werden die Testfälle angewendet?
Wie werden die Testfälle durchgeführt?
Wer ist wann mit dem Testen an der Reihe?
Welche Testfälle werden welcher Ressource zugeordnet?

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testplan

Im Zuge des Projektverlaufs werden aus Entwicklungsobjekten sogenannte Testobjekte.

Der Testplan listet zu jedem einzelnen Testobjekt die durchzuführenden Tests (Testaktivitäten) sowie deren Abhängigkeiten zueinander auf.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Vorteile eines Testplans sind

eine bessere Planung des Testprozesses,

frühzeitige Aussagen über Termine der Testaktivitäten der Fachbereiche,

präzise Informationen für den Projektleiter bzgl. der Fertigstellungsgrade und

eine bewusste Integration der Testaktivitäten in den Entwicklungsprozess.

Der Nachteil eines Testplans ist

in Abhängigkeit vom Detaillierungsgrad der Planung eine drastische Erhöhung der Anzahl der Aktivitäten im Projektplan.

Softwarequalität und -test

5. Vorlesung

„Testmanagement in Softwareprojekten“

(Softwaremetriken)

www.beuth-hochschule.de

Dipl.-Inform. Thomas Ziemer

Softwaremetriken

Softwaremetriken vermessen und vergleichen Software und ihren Entwicklungsprozess:

Man möchte etwas über die Eigenschaften des Messobjekts wissen (beispielsweise Laufzeiteffizienz etc.) und wendet daher die Messung an. Was dabei herauskommt, ist eine direkte Charakterisierung der interessierenden Eigenschaften.

Diese Information kann man verwenden, um etwas zu verbessern. Man ändert etwas und misst dann noch einmal. Am Ergebnis kann man ablesen, ob die Veränderung auch wirklich zur gewünschten Verbesserung geführt hat.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Def.: *„Eine Softwaremetrik ist allgemein eine Funktion, die eine Softwareeinheit in einen Zahlenwert abbildet. Dieser Wert ist interpretierbar als der Erfüllungsgrad eines Qualitätsziels für die Softwareeinheit.“*
(IEEE Standard 1061)

Diese Softwareeinheit ist im Allgemeinen eine *Methode*, eine *Klasse*, eine *Vererbungshierarchie*, ein *Modul*…

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Skalen für die Resultate der Metriken

Eine Berechnungsskala ist das "*Raster, mit dem die Zahlen interpretiert und verrechnet warden"* können. Je nach Skala sind unterschiedliche Operationen auf den Resultaten möglich und erlaubt, beispielsweise Vergleichen, Addieren oder Multiplizieren.

Man muss die Eingabedaten für eine Metrik bewusst wählen. Von ihnen hängt es ab, wie verlässlich und aussagekräftig die Resultate sein können. Aber was kann man aus diesen Resultaten wirklich ablesen? Mathematisch liegt das Problem darin, auf welcher Skala die Resultate liegen.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Skalen für die Resultate der Metriken

Nominalskala
Die schwächste Skala erlaubt nur, verschiedene Werte zu unterscheiden.

Weder darf man mit den Werten rechnen, noch sie anordnen. Wenn ein Resultat mit „Nr. 1“ und das andere mit „Nr. 2“ bezeichnet wird, ist auf einer Nominalskala dadurch keinerlei Reihenfolge gemeint, nur eine Unterscheidung der beiden Resultate. Eine Metrik auf der Nominalskala klassifiziert ihre Eingaben, bildet sie also auf Klassen ab.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Skalen für die Resultate der Metriken

Ordinalskala
Auf einer Ordinalskala gibt es eine Reihenfolge, man darf aber nicht rechnen.

Man kann damit nur ausdrücken: Resultat „Nr. 1“ *ist besser* als „Nr. 2“. Man kann ausdrücken „*das Steak schmeckt besser als die Pizza*“, aber nicht „*das Steak schmeckt doppelt so gut wie die Pizza*“.

Ordinalskalen sind für Bewertungen gut geeignet.

Softwaremetriken (Fortsetzung)

Skalen für die Resultate der Metriken

Rationalskala
In Rationalskalen darf man sinnvoll multiplizieren und dividieren, also auch Zahlen ins Verhältnis setzen.

Eine Spezifikation mit 200 Seiten *ist doppelt so dick* wie eine mit 100 Seiten.

Lines of Code (LoC)

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Einige Softwaremetriken sind weithin bekannt. Sie gehören zum Grundstock der Softwarequalität, man sollte sie kennen. Drei davon als Beispiel:

Lines of Code (LoC) mit Legende
Es werden die Quellcode-Zeilen eines Programms (häufig einer Methode) gezählt. Allerdings gibt es verschiedene Definitionen, was *genau* als Programmzeile gezählt wird:

Zeile mit Anweisung

Zeile mit Semikolon

Kommentarzeile

Programmzeile ohne Anweisung

Leere Zeile

```
public void buyTicket() {
   // adults pay more
   if(isAdult) {
      // pay here
      payFullFee();
   }
   else {
     // pay less
     payReducedFee();
   }

   // print ticket
   printTicket();
}
```

Zyklomatische strukturelle Komplexität („McCabe-Metrik“)

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Zyklomatische strukturelle Komplexität von *McCabe*
Anders als *Lines of Code* ist die zyklomatische strukturelle Komplexität exakt definiert. Es gibt also keine abweichenden Varianten. Es ist keine Legende nötig.

Oft wird sie auch einfach „McCabe-Metrik“ genannt. Die Metrik erhält ein Programmstück als Eingabe (häufig eine Methode) und liefert eine ganze Zahl auf einer Rationalskala (alle mathematischen Operationen erlaubt, siehe oben). Die Werte beginnen bei 1 und sind nach oben hin nicht beschränkt.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Die McCabe-Metrik geht davon aus, dass eine einfache Folge von Programmbefehlen nicht schwierig zu verstehen ist. Die Komplexität steigt, sobald der Programmfluss verzweigt. Bei jeder Verzweigung kommen neue „Zweige“, also Ablaufmöglichkeiten, hinzu. Je mehr es sind, desto schwieriger ist das Testen. Nebenbei führt die höhere Komplexität auch zu mehr Fehlern, weil das Programm schwerer zu verstehen ist. Die zyklomatische strukturelle Komplexität ist folgendermaßen definiert:

Zyklomatische Komplexität [auf Codebasis] =

Anzahl der Verzweigungen	(if)
+ Anzahl der Schleifen	(for, while, repeat usw.)
+ je: Anzahl der Zweige – 1	(case, switch)
+ 1	

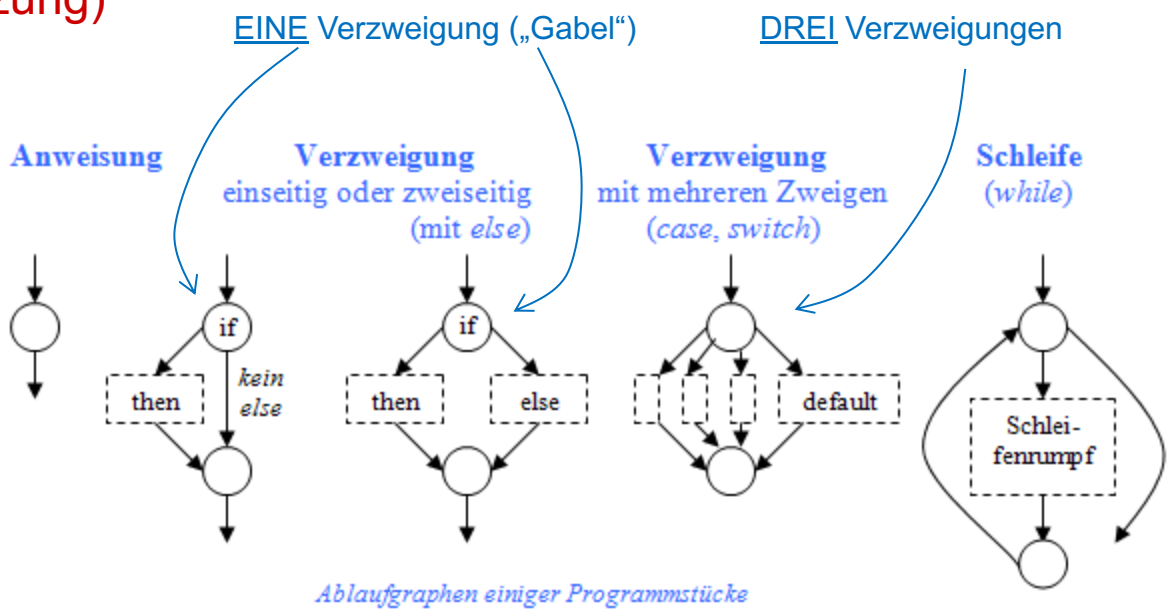
Ergebnis liefert immer mindestens 1

Weil nur die Struktur des Programmstücks in die Metrik einfließt, muss keine Information über den genauen Anweisungsinhalt vorliegen. Man benötigt nur den Programmablaufgraphen.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)



Ablaufgraphen einiger Programmstücke

Zyklomatische Komplexität [auf Codebasis] =

Anzahl der Verzweigungen	(if)
+ Anzahl der Schleifen	(for, while, repeat usw.)
+ je: Anzahl der Zweige – 1	(case, switch)
+ 1	

Ergebnis liefert immer mindestens 1

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Mit der resultierenden Zahl allein kann man dennoch wenig anfangen. Erst wenn man sie mit den Maßen anderer Programme vergleicht, wird aus der Zahl eine Aussage. Das hat *McCabe* getan:

Er sagt im Kontext einer Methode, eine zyklomatische strukturelle Komplexität

- bis 10 sei „niedrig“
- bis 20 „mittel“
- darüber hoch und
- über 50 „undurchschaubar“.

In der Literatur wird 10 oft als maximal vertretbarer Wert genommen (für OO-Programme geringer, z.B. 5)

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Interessant an der McCabe-Metrik: Rekursionen nehmen keinen Einfluss auf die Komplexität. Sie werden behandelt wie jeder andere Methodenaufruf auch.

Wir empfinden Rekursionen wohl eher als komplex. ;)

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Dennoch: Komplexität ist etwas sehr subjektives.

```
switch(day) {
      case 0: return "Montag";
      case 1: return "Dienstag";
      case 2: return "Mittwoch";
      case 3: return "Donnerstag";
      case 4: return "Freitag";
      case 5: return "Samstag";
      case 6: return "Sonntag";
      default: return "Häh?";
}
```

Test: Welche Komplexität?

Antwort:
7 VERzweigungen + 1 (Konstante) = 8

Softwarequalität und -test

Originale Berechnung der McCabe-Komplexität

1. Man konstruiere den Kontrollflussgraphen
2. Man messe die strukturelle Komplexität

   Die zyklomatische Zahl $z(G)$ eines Kontrollflussgraphen G ist:

   $z(G) = e - n + 2$ mit

   e = Anzahl der Kanten des Kontrollflussgraphen

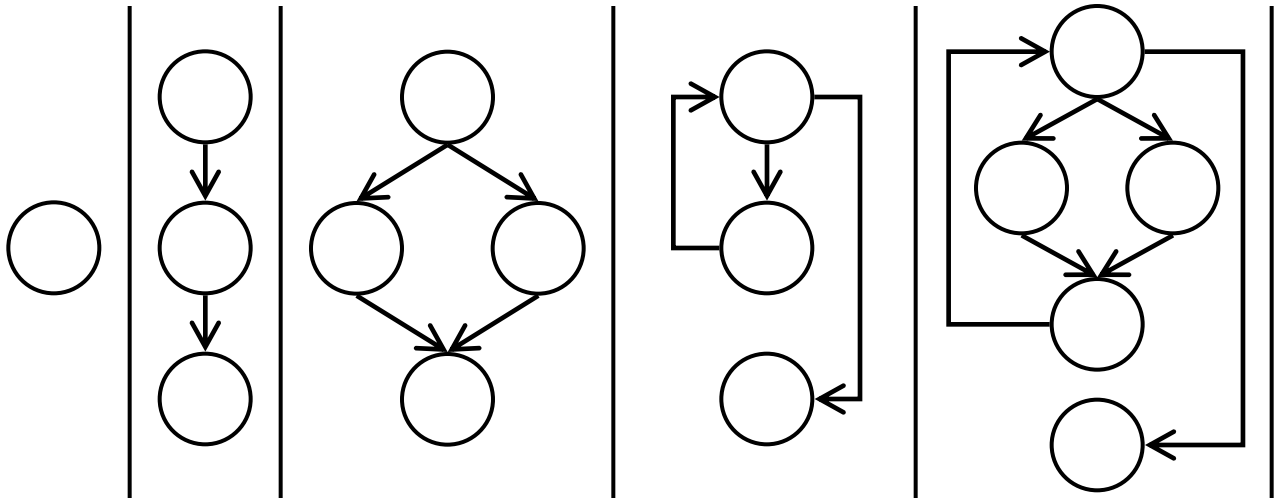
   n = Anzahl der Knoten

- Interessant: Zyklomatische Komplexität gibt Obergrenze für die Testfallanzahl für den Zweigüberdeckungstest an
- In der Literatur wird 10 oft als maximal vertretbarer Wert genommen (für OO-Programme geringer, z. B. 5)
- für Java: #if + #do + #while + #switch-cases

Kleuker, S. (2013). Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten (3. Aufl.). Wiesbaden: Springer.

Softwarequalität und -test

Originale Berechnung der McCabe-Komplexität



Anzahl Kanten *e*	**0**	**2**	**4**	**3**	**6**
Anzahl Knoten *n*	**1**	**3**	**4**	**3**	**5**
McCabe Zahl *z(G)*	**1**	**1**	**2**	**2**	**3**

Kleuker, S. (2013). Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten (3. Aufl.). Wiesbaden: Springer.

Halstead-Metriken

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Halstead-Metriken
Die Halstead-Metriken bedienen sich der Annahme, dass ausführbare Programmteile aus Operatoren und Operanden aufgebaut sind. Die Definition, was die zu betrachtenden Operatoren und Operanden sind, ist dabei eine der Aufgaben vor dem Einsatz der Halstead-Metriken.

Typischerweise werden beispielsweise Variablen und Konstanten als Operanden betrachtet; Schlüsselwörter, logische und Vergleichsoperatoren usw. als Operatoren.

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Es werden dann für jedes Programm folgende Basismaße gebildet:

Die Anzahl der verwendeten unterschiedlichen Operatoren (n_1) und Operanden (n_2) liefern zusammen die Vokabulargröße n (*vocabulary size*).

Die Anzahl der insgesamt verwendeten Operatoren (N_1) und Operanden (N_2) liefern zusammen die Implementierungslänge N.

Softwarequalität und -test

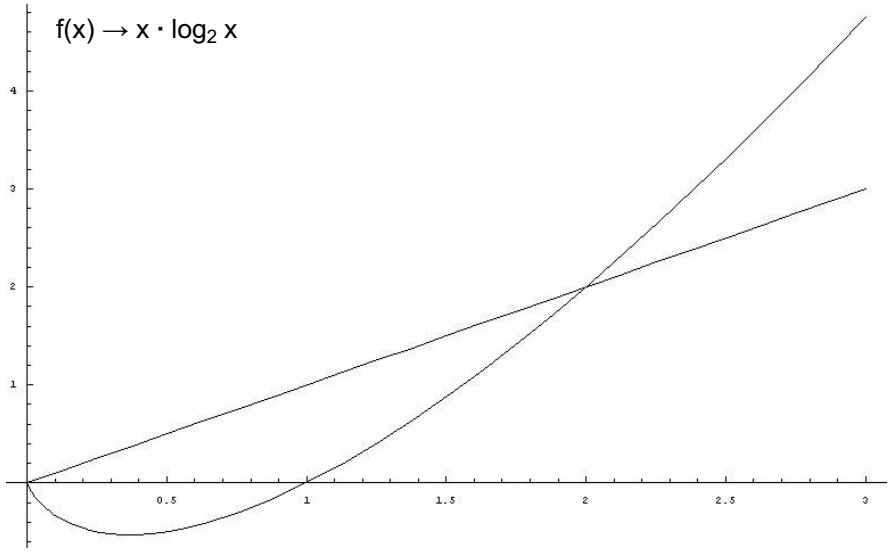
Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Hieraus werden die Basisgrößen Halstead-Länge *HL* (*program length*) und Halstead-Volumen *HV* (*program volume*) errechnet. Das Volumen beschreibt hierbei die Größe der Implementierung eines Algorithmus':

$$HL = n_1 \cdot \log_2 n_1 + n_2 \cdot \log_2 n_2$$
$$HV = N \cdot \log_2 n$$



Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Aus den Basismaßen kann man noch weitere Kennzahlen berechnen, beispielsweise:

Schwierigkeit *D* (*difficulty level*)
Ein Grad der Schwierigkeit, das Programm zu schreiben oder zu verstehen.

$$D = \frac{n_1}{2} \cdot \frac{N_2}{n_2}$$

Programmniveau *L* (*program level*)
Es handelt sich hierbei um die Fehlerneigung eines Programms, die aus dem Kehrwert der Schwierigkeit *D* berechnet wird. Je niedriger das Niveau, desto fehlerträchtiger das Programm.

$$L = \frac{1}{D}$$

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Implementierungsaufwand *E* (*effort to implement*)
Der Implementierungsaufwand *E* ist proportional zum Volumen *HV* und zum Schwierigkeitsgrad *D* des Programms.

$$E = HV \cdot D$$

Implementierungszeit *T* (*time to implement*)
Die Implementierungszeit *T*, sind proportional zum Implementierungsaufwand. Empirische Untersuchungen haben ergeben, dass man eine gute Näherung in Sekunden erhält, wenn man *E* durch *t* = 18 teilt.

$$T = \frac{E}{t} \text{ Sekunden}$$

–

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Bekannte Softwaremetriken

Anzahl der ausgelieferten Fehler *B* (*number of delivered bugs*)
Dieser Wert liefert eine Schätzung über die Zahl der Fehler *B*, die in der ausgelieferten Implementierung noch enthalten sind. Die Konstante *e* (*experience*) = 3.000 liefert erfahrungsgemäß eine recht treffende Aussage.

$$B = \frac{E^{2/3}}{e}$$

Fazit: Die Halstead-Metriken sind leicht zu ermitteln und zu berechnen, automatisierbar, für alle Programmiersprachen einsetzbar, und überraschenderweise meist ein wirklich gutes Maß für die Komplexität. Der Nachteil ist, dass sie nur die lexikalische bzw. textuelle Komplexität messen.

Weitere Softwaremetriken

Softwarequalität und -test

Testmanagement in Softwareprojekten

Software metric	Description
Fan-in / Fan-out	Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.

Sommerville, I. (2015). Software Engineering, Global Edition (10. Aufl.). Harlow, Essex: Pearson Education.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Software metric	Description
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability.
Length of identifiers	This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value of a document's Fog index, the more difficult the document is to understand.

Sommerville, I. (2015). Software Engineering, Global Edition (10. Aufl.). Harlow, Essex: Pearson Education.

Softwarequalität und -test

The *Chidamber & Kemerer* object-oriented metrics suite

Object-oriented metric	Description
Weighted methods per class (WMC)	This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
Depth of inheritance tree (DIT)	This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
Number of children (NOC)	This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.

Sommerville, I. (2015). Software Engineering, Global Edition (10. Aufl.). Harlow, Essex: Pearson Education.

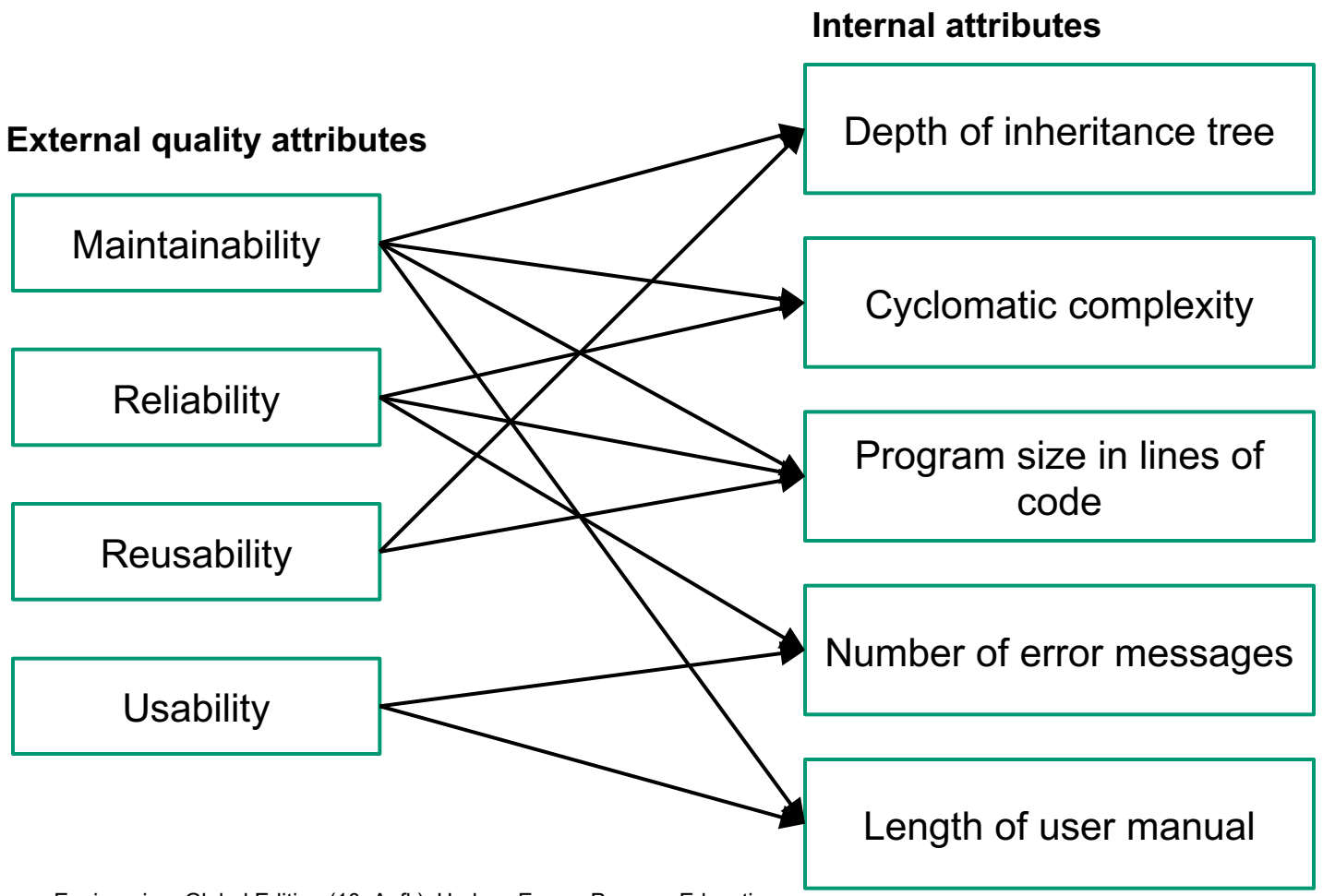
The *Chidamber & Kemerer* object-oriented metrics suite

Object-oriented metric	Description
Coupling between object classes (CBO)	Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists. A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program.
Response for a class (RFC)	RFC is a measure of the number of methods that could potentially be executed in response to a **message received by an object** **(polymorph)** of that class. Again, RFC is related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors.

Sommerville, I. (2015). Software Engineering, Global Edition (10. Aufl.). Harlow, Essex: Pearson Education.

Softwarequalität und -test

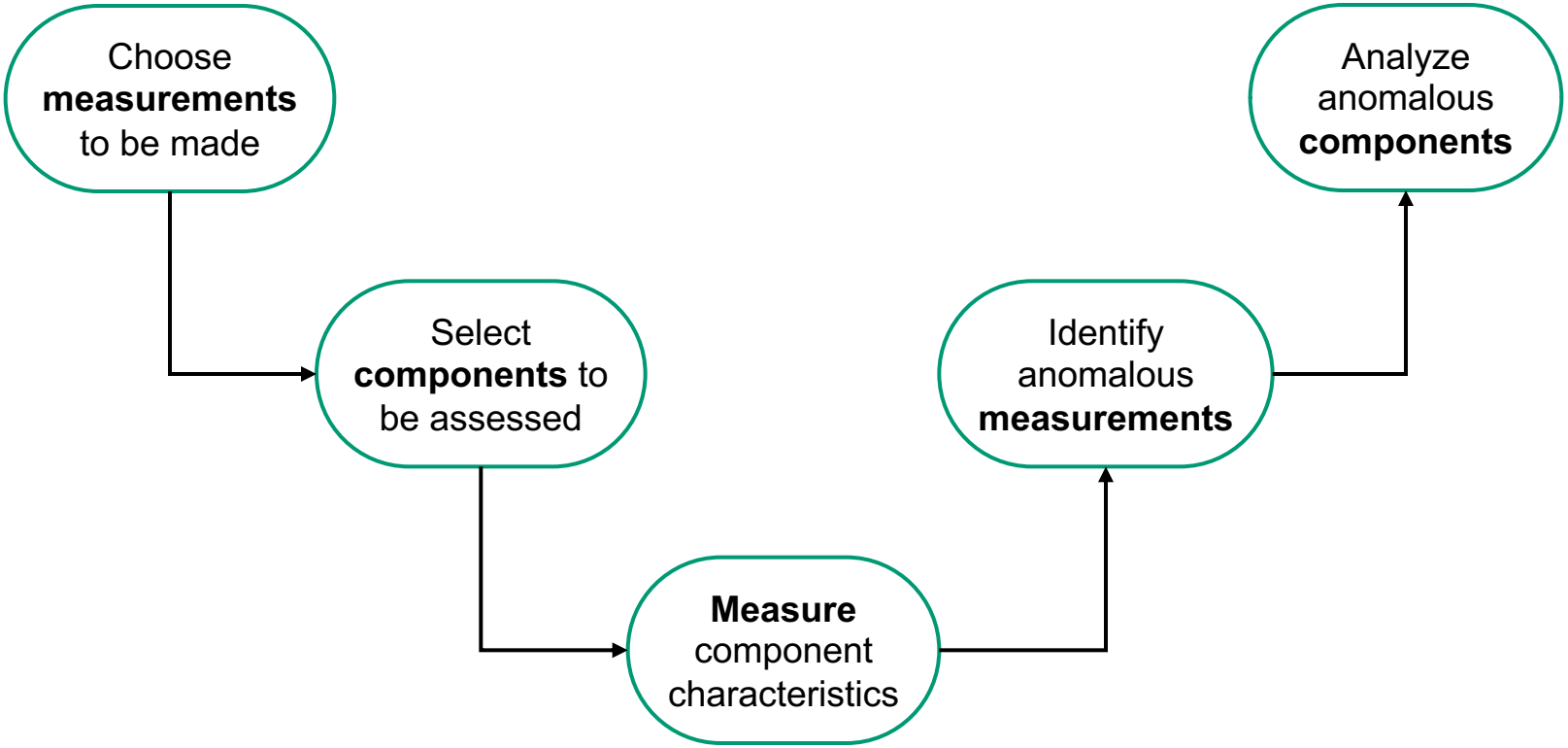
Relationships between internal and external quality attributes



Sommerville, I. (2015). Software Engineering, Global Edition (10. Aufl.). Harlow, Essex: Pearson Education.

Softwarequalität und -test

The process of product measurement



Sommerville, I. (2015). Software Engineering, Global Edition (10. Aufl.). Harlow, Essex: Pearson Education.

Programmablaufgraph
(Kontrollflussgraph)

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

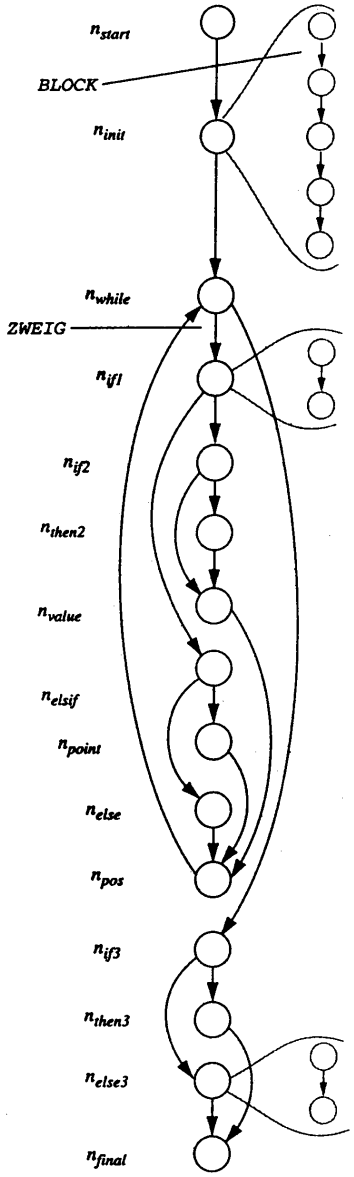
Codeüberdeckungsgrad

- Testdaten müssen Pfade durch das Programm systematisch durchlaufen; mehrere Varianten: Anweisungsüberdeckung ... (s. u.)

Programmablaufgraph

- Anweisungsstruktur → *Kontrollflussgraph (KFG)*
  Knoten: lineare Anweisungsfolge oder Ausdruck
  Kante: mögliche Verbindung

Softwarequalität und -test




```
Wert = 0.0;
Genauigkeit = 1.0;
WoBinIch = VorDemKomma;
Fehlerfrei = true;
Position = 0;

while ( (Position < inZiffernString.length()) && Fehlerfrei) {
        Zchn = inZiffernString.charAt(Position);
        if Ziffer(Zchn) {
           if (WoBinIch == NachDemKomma) {
                  Genauigkeit = Genauigkeit / 10.0;
           }
           Wert = 10.0 * ZchnZuWert(Zchn);
        }
        else
        if ( (Zchn == '.') && (WoBinIch == VorDemKomma)) {
               WoBinIch = NachDemKomma;
        }
        else {
            Fehlerfrei = false;
        }

        Position++;
}

if ( !Fehlerfrei ||  (inZiffernString.length() == 0) ||
     (WoBinIch == NachDemKomma) && (inZiffernString.length() == 1) ) {
         return FehlerCode;
}
else {
    Wert = Wert * Genauigkeit;
    return Wert;
}
```

Kontrollflussgraph (KFG)

D. Macos nach K. Bothe, Software Engineering

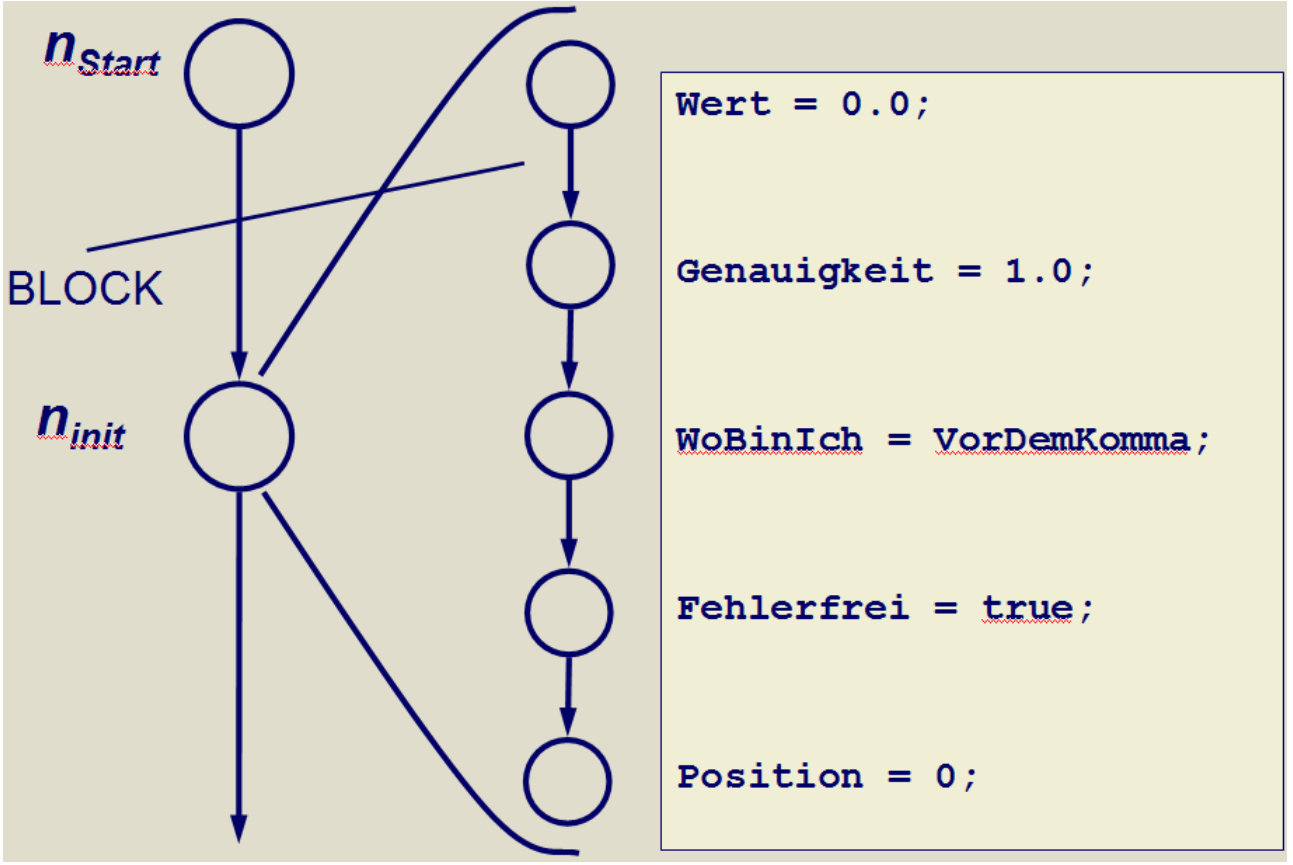
Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Kontrollflussgraph

Mehrere Anweisungen können zu einem Block zusammenge-fasst werden.



Softwarequalität und -test

Testmanagement in Softwareprojekten

Folgende Regeln gelten, mit denen mehrere Knoten k_1, k_2, ..., k_n nacheinander durchlaufen werden können, die also $k_1 \rightarrow k_2 \rightarrow ... \rightarrow k_n$ zu einem einzigen Knoten verschmelzen:

a) Die Knotenfolge wird bei jedem Durchlauf immer nur über k_1 betreten, es gibt außer den genannten Kanten keine weiteren Kanten, die in k_2, ..., k_n enden.

b) Die Knotenfolge wird bei jedem Durchlauf immer nur über k_n verlassen, es gibt außer den genannten Kanten keine weiteren Kanten, die in k_1, ..., k_{n-1} beginnen.

c) Die Knotenfolge ist maximal bezüglich a) und b).

Kleuker, S. (2013). Grundkurs Software Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten (3. Aufl.). Wiesbaden: Springer.

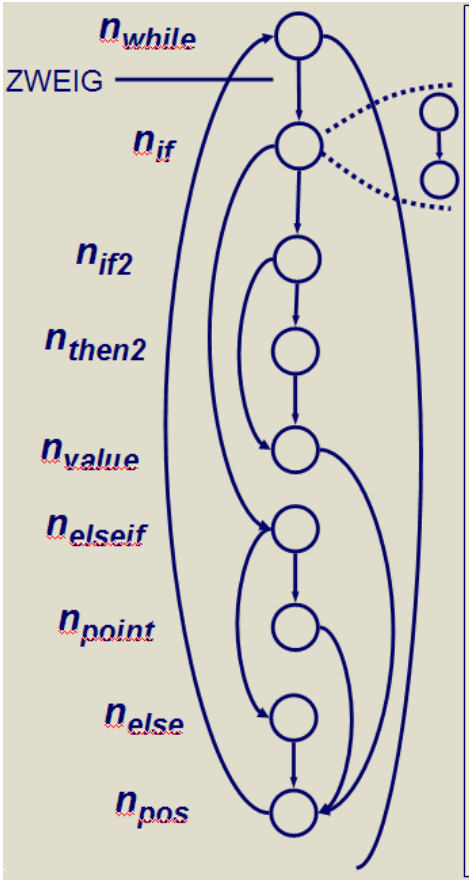
Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Kontrollflussgraph

Die Pfeile stellen mögliche Abläufe durch das betrachtete Programmelement dar.




```
while ( (Position < inZiffernString.length()) && Fehlerfrei) {

    Zchn = inZiffernString.charAt(Position);

    if Ziffer(Zchn) {

        if (WoBinIch == NachDemKomma) {

            Genauigkeit = Genauigkeit / 10.0;
        }

        Wert = 10.0 * ZchnZuWert(Zchn);
    }
    else if ( (Zchn == '.') && (WoBinIch == VorDemKomma)) {


        WoBinIch = NachDemKomma;
    }
    else {
        Fehlerfrei = false;
    }
    Position++;
}
```

Testmanagement in Softwareprojekten

- Ein vollständiger Pfad ist eine Folge von verbundenen Knoten (über Kanten) im KFG, die mit n_{start} beginnt, und mit n_{final} endet.
- Jede mögliche Ausführungsreihenfolge des Programms ist ein Teil der Menge der vollständigen Pfade.
- Wunsch: Durchlauf „repräsentativer“ vollständiger Pfade beim Test, denn Überdeckung *aller* vollständigen Pfade ist im allgemeinen nicht ausführbar.
- Ansatz: Verschiedene Approximationsstufen (Anweisungsüberdeckungstest, ..., Mehrfach-Bedingungsüberdeckungstest) für die Menge der vollständigen Pfade bei Auswahl der Testdurchläufe wählen.

Softwarequalität und -test

5. Vorlesung
„Testmanagement in Softwareprojekten“
(Codeüberdeckungsgrade)

www.beuth-hochschule.de

Dipl.-Inform. Thomas Ziemer

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Codeüberdeckungsgrad

In vielen Softwareunternehmen gibt es die Regel, eine *"Testabdeckung von mindestens 80%"* zu erreichen. Aber was bedeutet das?

Bleiben etwa 20% der Programmelemente (z.B. Methoden) ungetestet???

Nein, natürlich nicht!

ALLE Programmelemente werden getestet, aber eben nicht vollständig.
Einzelne Pfade durch das Programmelement bleiben ungetestet.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Codeüberdeckungsgrad

Das übergeordnete Testziel besteht darin, mit möglichst wenigen Testfällen alle Anweisungen, Zweige oder sogar Pfade im Testobjekt abzudecken.

Der Prozentsatz der abgedeckten Programmelemente heißt Codeüberdeckungsgrad (*code coverage*, oder auch Test**ab**deckungsgrad).

Das Maß besagt, welcher Anteil der Pfade, Zweige oder Anweisungen mit Hilfe gegebener Testfälle wirklich ausgeführt wird. Damit werden die Testfälle im Verhältnis zum Quellcode bewertet:

Bessere Testfälle zum gleichen Code führen zu einer höheren Codeüberdeckung.

Softwarequalität und -test

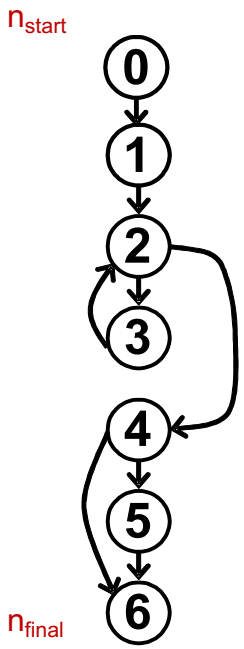
Testmanagement in Softwareprojekten

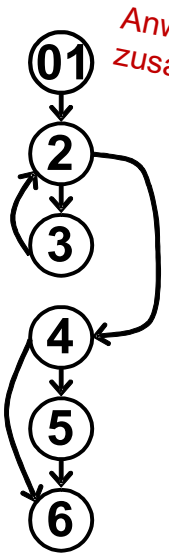
```
public int aha(int ein) {
 int erg = - ein / 2;          // 0
 int i = ein;                  // 1

 while (i > 0) {               // 2
  erg = erg + (i--);           // 3
 }

 if (ein < 0 || ein % 2 == 1) // 4
  erg = 0;                     // 5
 }

 return erg + 2;               // 6
}
```





Anweisungen 0 und 1 zusammenfassen

Kleuker, S. (2013). Grundkurs Software Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten (3. Aufl.). Wiesbaden: Springer.

Anweisungsüberdeckung C^0

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

C^0

Codeüberdeckungsgrad

Je nach Kritikalität des Testobjekts sind bei der Testplanung die angestrebten Testabdeckungsgrade vorzugeben. Für den Modultest werden die so genannten C^k-Abdeckungsmaße unterschieden. Dabei bedeutet:

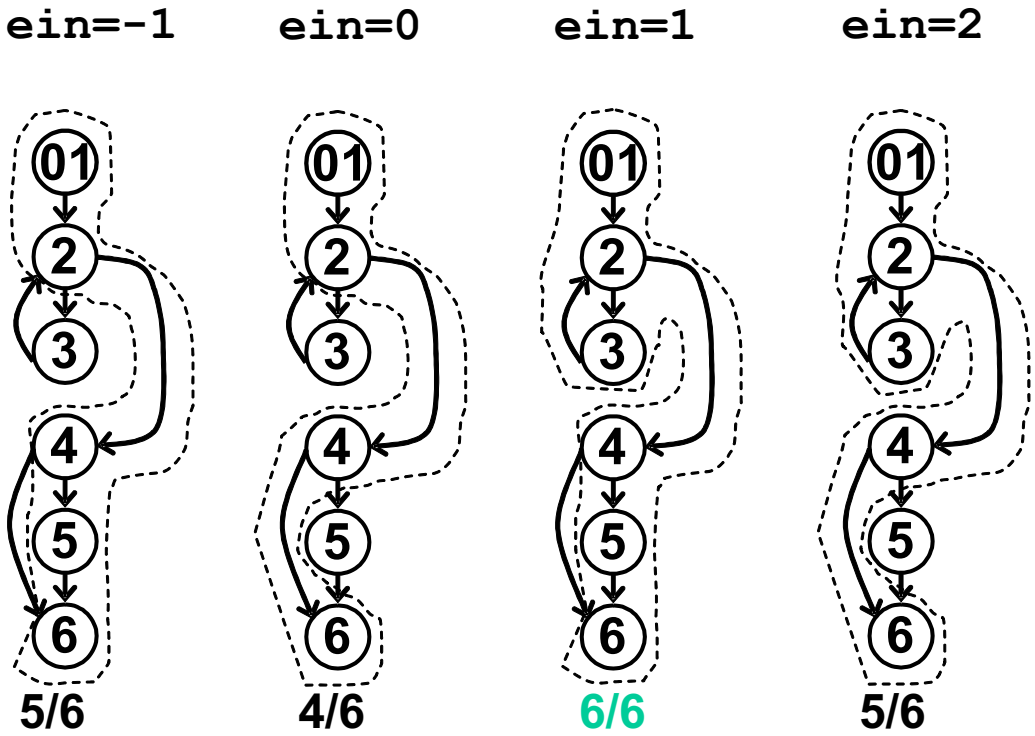
C^0 (Anweisungsüberdeckung)
Die Anweisungsüberdeckung (*statement coverage*) ist das Verhältnis der Anzahl der durchlaufenen Anweisungen (Methodenaufrufe, Wertzuweisungen etc.) zur Gesamtzahl der Anweisungen eines Testobjekts.

$C^0 = 100\%$ bedeutet, dass alle Anweisungen mindestens einmal mit einem Testfall ausgeführt wurden.

Softwarequalität und -test

Testmanagement in Softwareprojekten

- Ziel: Alle Anweisungen des Programms durch Wahl geeigneter Testdaten mindestens einmal ausführen, alle Knoten des KFG mindestens einmal besuchen.
- Testmaß
- Ziel: $C^0=1$ (=100%)
- für {-1}
  $C^0=$ 5/6
- für {-1,0}
  $C^0=$ 5/6
- für {1} und für {-1,2}
  $C^0=$ 6/6



Kleuker, S. (2013). Grundkurs Software Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten (3. Aufl.). Wiesbaden: Springer.

Zweigüberdeckung C^1

Softwaremetriken (Fortsetzung)

C^1

Codeüberdeckungsgrad

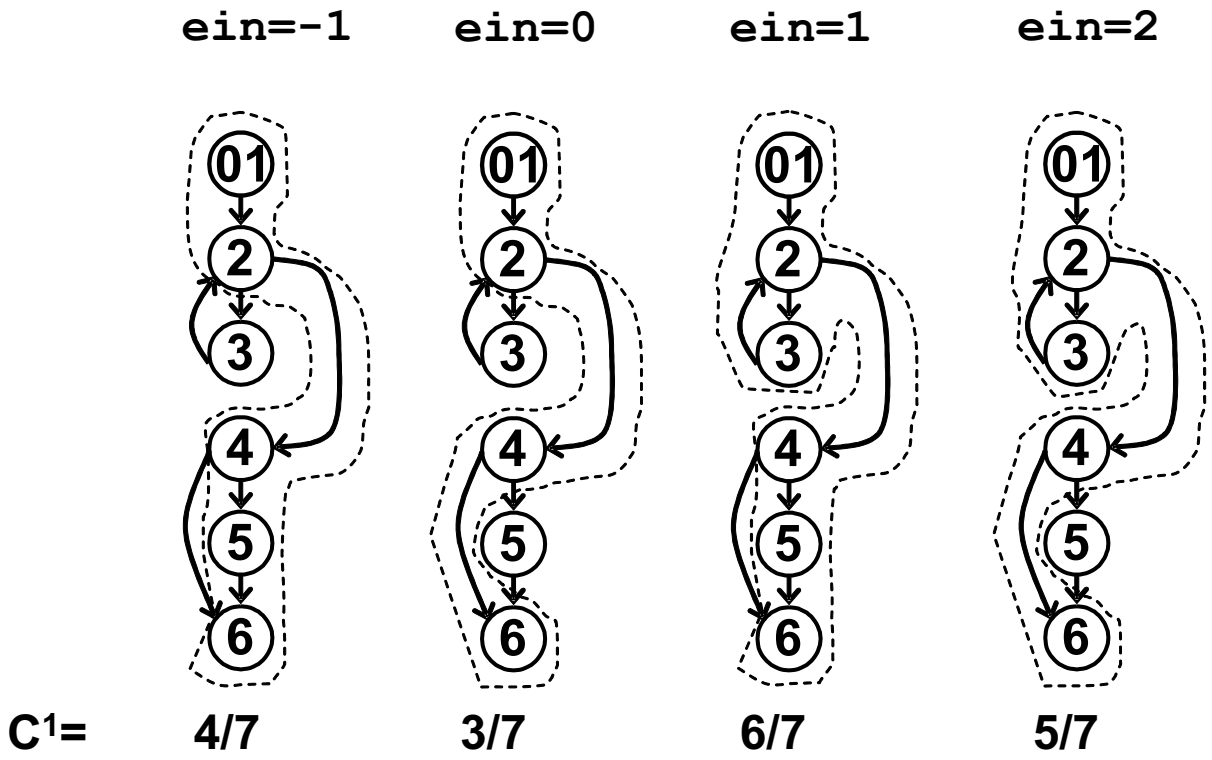
C^1 (Zweigüberdeckung)
Die Zweigüberdeckung (*branch coverage*) ist das Verhältnis der durchlaufenen Zweige (if, Schleifen, switch) zur Gesamtzahl der Zweige eines Testobjekts.

C^1 = *100%* bedeutet, dass alle Entscheidungen mindestens einmal mit „Ja“ und einmal mit „Nein“ ausgeführt wurden.

Softwarequalität und -test

Testmanagement in Softwareprojekten

- Ziel: alle Kanten des KFG überdecken, d.h. alle Zweige des Programms einmal durchlaufen
- Testmaß
- Ziel: $C^1=1$
- für {-1}
  $C^1=$ 4/7
- für {-1,0}
  $C^1=$ 5/7
- für {-1,2}
  $C^1=$ 7/7



Kleuker, S. (2013). Grundkurs Software Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten (3. Aufl.). Wiesbaden: Springer.

Bedingungskombinations-überdeckung C^2

Softwaremetriken (Fortsetzung)

C^2

Codeüberdeckungsgrad

C^2 (Überdeckung aller Bedingungskombinationen)
Die Überdeckung aller Bedingungskombinationen (*condition coverage*) wird auch C^{1p}- oder Prädikatüberdeckung genannt.

Die Überdeckung aller Bedingungskombinationen ist das Verhältnis der Anzahl der durch Testfälle überdeckten Bedingungskombinationen zur Gesamtzahl der möglichen Bedingungskombinationen.

Aber was ist eine Bedingungskombination?

Softwarequalität und -test

Testmanagement in Softwareprojekten

„Jeder einzelne Term muss durch Testfälle mindestens einmal zu *true* und einmal zu *false* auswerten.“

Softwaremetriken (Fortsetzung)

Codeüberdeckungsgrad C^2

```
public int aha(int ein) {
 int erg = - ein / 2;          // 0
 int i = ein;                  // 1

 while (i > 0) {               // 2
  erg = erg + (i--);           // 3
 }

 if (ein < 0 || ein % 2 == 1) // 4
  erg = 0;                     // 5
 }

 return erg + 2;               //6
}
```

Die Prädikatenabdeckung beobachtet, ob auch jeder einzelne Teilterm dabei jeden Wahrheitswert angenommen hat:

1. *(ein<0, ein%2==1)* → false, false
2. *(ein<0, ein%2==1)* → true, false
3. *(ein<0, ein%2==1)* → false, true
4. *(ein<0, ein%2==1)* → true, true

Erst durch diese Kombinationen ist C^2 zu 100 % erfüllt.

Kleuker, S. (2013). Grundkurs Software Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten (3. Aufl.). Wiesbaden: Springer.

Pfadüberdeckung C^{∞}

Softwaremetriken (Fortsetzung)

C^{∞}

Codeüberdeckungsgrad

C^{∞} (Pfadüberdeckung)
Die Pfadüberdeckung (*path coverage*) ist das Verhältnis der Anzahl der durch Testfälle überdeckten Pfade zur Gesamtzahl der Pfade eines Testobjekts.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

C∞

Codeüberdeckungsgrad

Ein Pfad ist eine konkrete Ablaufmöglichkeit des Programms. An einer if-Verzweigung (Gabel) gibt es zwei Möglichkeiten, weiterzulaufen, damit also zwei Pfade.

Zu noch viel mehr Pfaden führen Schleifen:
- Wird eine Schleife nullmal durchlaufen (übersprungen), so ist das ein Pfad.
- Wird eine Schleife einmal durchlaufen, so ist das ein Pfad.
- Wird sie zweimal durchlaufen, ist das ein weiterer Pfad usw.

Kommen gar Schleifen mit potenziell unbegrenzter Durchlaufzahl vor, so ist auch die Anzahl verschiedener Pfade unendlich groß, was den Namen C∞ erklärt.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

C^{∞}

Codeüberdeckungsgrad

$C^{\infty a}$ - vollständiger Pfadüberdeckungstest
Es werden alle möglichen Pfade getestet.
Problem: Bei Programmen mit Schleifen kann es praktisch unendlich viele Pfade geben, die nicht vollständig abdeckbar sind.

$C^{\infty b}$ - Boundary-Interior-Pfadüberdeckungstest
Im Prinzip wie der $C^{\infty a}$-Test, nur dass nun die Schleifendurchläufe auf <= 2 (null Durchläufe, ein Durchlauf, zwei Durchläufe) reduziert werden.

Softwaremetriken (Fortsetzung)

C^{∞}

Codeüberdeckungsgrad

$C^{\infty c}$ - Strukturierter Pfadüberdeckungstest
Im Prinzip wie der $C^{\infty b}$-Test, nur dass nun die Anzahl der Schleifendurchläufe auf eine vorgegebene natürliche Zahl n reduziert wird.

Vorteil
Hohe Fehlererkennungsrate

Nachteil
nicht ansprechbare Pfade auf Grund von Bedingungen

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Codeüberdeckungsgrad

C^{∞} = *100%* bedeutet, dass alle möglichen Pfade durch Testfälle abgedeckt wurden.

Falls das Testobjekt keine Wiederholungsschleife beinhaltet, ist $C^2 = C^{\infty}$.

Softwaremetriken (Fortsetzung)

Codeüberdeckungsgrad

Anweisungs- und Zweigüberdeckung sind am weitesten verbreitet. Die weiteren Überdeckungsmaße bieten nur Möglichkeiten für Verfeinerung und Perfektionierung.

In der Praxis begnügt man sich in der Regel mit $C^0 = 95\%$ und $C^1 = 80 - 85\%$, da der Testaufwand sonst zu hoch wird.

Softwaremetriken (Fortsetzung)

Objektorientierung und Überdeckungsgrade

Natürlich können nicht nur die oben genannten Konstrukte wie Pfade, Zweige und Anweisungen überdeckt werden.

Analog könnte man in objektorientierten Programmiersprachen wie Java, C++ oder Smalltalk fragen, ob jede Klasse und jede Operation ausgeführt wird. Das würde man Klassenüberdeckung bzw. Operationenüberdeckung nennen.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Softwaremetriken (Fortsetzung)

Puh, fertig...

Softwarequalität und -test

5. Vorlesung
„Testmanagement in Softwareprojekten“
(Testobjekt und Testziel)

www.beuth-hochschule.de

Dipl.-Inform. Thomas Ziemer

Testmanagement in Softwareprojekten

Testobjekt und Testziel

Testobjekt

Die Entstehungsreihenfolge der Testobjekte hängt von der Entwicklungsstrategie ab.

Bottom-Up (ist die im Beispiel gewählte Entwicklungsstrategie)
Top-Down
Hardest-First

Teststrategien

Softwarequalität und -test

Testmanagement in Softwareprojekten

Teststrategien

Top-down-Strategie

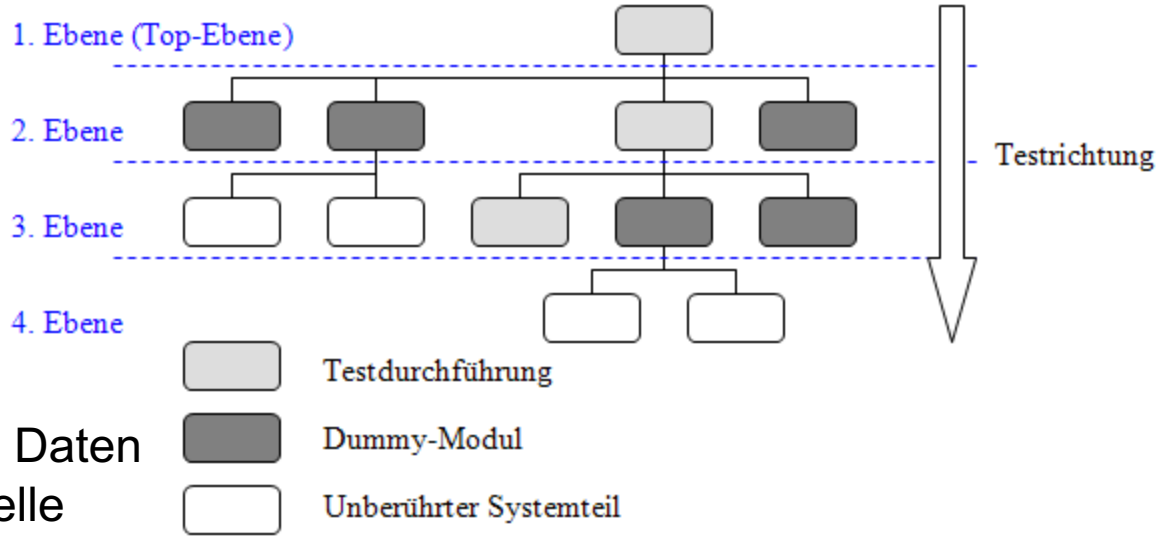
Beim Einsatz der Top-down-Strategie („von oben nach unten“) wird eine Software entlang ihrer hierarchischen Ebenen ausgetestet. Dabei wird zunächst die **Steuerung erstellt** und ausgetestet. Anschließend werden die jeweils erstellten Module in der Reihenfolge ihrer Fertigstellung getestet. Da zu diesem Zeitpunkt die darunter liegenden (auf**zu**rufenden) Module noch nicht vorhanden sind, werden so genannte Dummy-Module definiert, die danach sukzessive durch die echten Module ersetzt werden.
Die Top-down-Strategie eignet sich für den Test von Programmen, die sehr klar strukturiert sind, und deren Modulsteuerung immer im hierarchisch darüber liegenden Modul liegt.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Teststrategien (Fortsetzung)

Top-down-Strategie



Als Hilfsmittel werden Dummy-Module benötigt. Ein Dummy-Modul kann sowohl Daten über eine Schnittstelle erhalten (Eingabeparameter) als auch Daten über eine definierte Schnittstelle weitergeben (Ausgabeparameter).

Diese übergebenen Daten werden jedoch nicht in diesem Modul erzeugt. Stattdessen wird eine Möglichkeit vorgesehen, welche das manuelle Eingeben der Daten (beispielsweise über den Bildschirm) gestattet. Außerdem sollte jedes Dummy-Modul über die Möglichkeit der Ausgabe seiner Modulbezeichnung verfügen.

Testmanagement in Softwareprojekten

Teststrategien (Fortsetzung)

Bottom-up-Strategie

Beim Einsatz der Bottom-up-Strategie („von unten nach oben“) werden **zunächst die Verarbeitungsmodule**, dann anschließend die Steuerung erstellt und ausgetestet.

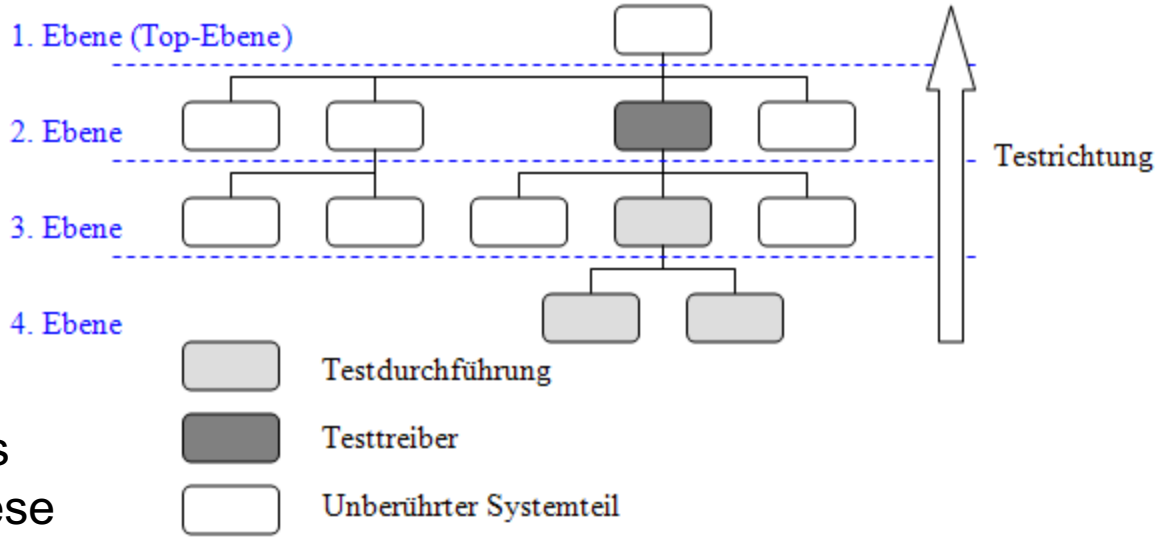
Da zu diesem Zeitpunkt die darüber liegenden (aufrufenden) Module noch nicht vorhanden sind, wird ein so genannter Testtreiber definiert, der die Übergabe und den Empfang der im Modul erzeugten Daten über die Schnittstelle ermöglicht.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Teststrategien (Fortsetzung)

Bottom-up-Strategie



Als Hilfsmittel wird ein Testtreiber benötigt, der für die Möglichkeit der Eingabe von Daten (Eingabeparameter) über die definierte Schnittstelle an das auf**ge**rufene Modul sorgt. Diese Eingabemöglichkeit sollte dabei über den Bildschirm bestehen. Nach Verarbeitung der Daten im Modul übernimmt der Testtreiber die in der Schnittstelle zur Verfügung gestellten Ausgabedaten (Ausgabeparameter). Nach Durchführung des Tests werden sowohl die Eingabeparameter als auch die Ausgabeparameter dokumentiert.

Testmanagement in Softwareprojekten

Teststrategien (Fortsetzung)

Up-down-Strategie (Hardest-First)

Mit der Up-down-Strategie (auch „Inside-out“ oder „Hardest-first-Strategie“ genannt) wird eine Mischform beschrieben, bei der zunächst die schwierigsten Module erstellt und ausgetestet werden.

Da zu diesem Zeitpunkt unter Umständen weder die darüber liegenden (aufrufenden) noch die darunter liegenden (aufgerufenen) Modul vorhanden sind, werden sowohl Testtreiber als auch Dummy-Module benötigt.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Teststrategien (Fortsetzung)

Strategiewahl in Abhängigkeit vom Anwendungstyp

Die folgende Tabelle zeigt einige typische Anwendungssysteme und die für sie jeweils günstigsten Teststrategien:

Typ der Anwendung	*Top-down*	*Bottom-up*
Systemsoftware (beispielsweise Betriebssystem)		√
Dateisystem		√
Software in Echtzeit, z.B. Autopilot im Flugzeug		√
Kommerzielle Anwendung, z.B. Lohnabrechnung	√	
Anwendung dominierender Mensch-Maschine-Schnittstelle	√	
Anwendung mit großen interaktiven Anteilen	√	
Spielprogramm	√	

Quelle: G. E. Thailer: Software-Test, heise-Verlag, 2002

Testobjekt

Testmanagement in Softwareprojekten

Testobjekt und Testziel

Testobjekt

Um den Test zu optimieren und damit effizienter zu gestalten, müssen im Testplan Testobjekte definiert warden **(“alles ist Testobjekt!”)**.

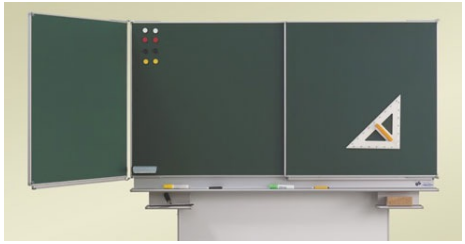
Diese Testobjekte (auch Testkandidaten genannt) bilden die Basis für die Planung der Tests hinsichtlich der Abhängigkeiten der Testobjekte zueinander, für die Definition spezieller Testziele sowie für die Testdurchführung.

Je nach Testphase werden die Testobjekte vom feinsten Detaillierungsgrad ausgehend immer größer und komplexer.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testobjekt und Testziel (Fortsetzung)



TF

Testobjekt

Teilfunktionalität (TF)

Fachlich: eine Teilfunktionalität (ein „Ausschnitt“) innerhalb eines Geschäftsprozesses (beispielsweise *Fette Darstellung eines Texts wählen*).

Technisch: ein definiertes Modul (z.B. eine *Klasse* oder eine einzelne *Methode*).

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testobjekt und Testziel (Fortsetzung)

FK

Testobjekt

Geschäftsprozess oder Funktionalität (FK)

Fachlich: ein durchgängiger Ablauf eines Geschäftsprozesses innerhalb eines Bereiches der Anwendung (beispielsweise *Alle Auszeichnungen eines Texts auswählen können*).

Technisch: Verknüpfung mehrerer Module zu einem vollst. Geschäftsablauf.

Testobjekt und Testziel (Fortsetzung)

IO

Testobjekt

Informationsobjekt (IO)

Eine Datei oder eine Datenbank.

Testmanagement in Softwareprojekten

Testobjekt und Testziel (Fortsetzung)

MA

Testobjekt

Maske (MA)

Eine logische Eingabemaske mit alle ihren Eingabefeldern, Schaltflächen, Menüs usw. innerhalb einer Funktionalität (Geschäftsprozess) oder Teilfunktionalität

(beispielsweise *Erfassungsmaske bei der Anlage eines Kunden*).

Testobjekt und Testziel (Fortsetzung)

LI

Testobjekt

Liste, Bericht oder Dokument (LI)

Eine Liste (Bericht) oder ein Dokument als Ausgabe einer Funktionalität bzw. Maske (beispielsweise Bericht als *Auflistung alle Kunden*).

Testmanagement in Softwareprojekten

Testobjekt und Testziel (Fortsetzung)

ANW

Testobjekt

Anwendung

Die Zusammenfassung aller Geschäftsprozesse innerhalb eines Bereiches des Systems (beispielsweise *Textverarbeitung* oder Tabellenkalkulation).

Testobjekt und Testziel (Fortsetzung)

SYS

Testobjekt

System

Ein durchgängiger Ablauf, der sich über mehrere verschiedene Bereiche erstreckt (beispielsweise *Office-System* etc.).

Testziel

Testmanagement in Softwareprojekten

Testobjekt und Testziel

Testziel

Durch die Definition der Testziele wird die Anzahl der Testfälle reduziert, der Test prinzipiell zielorientiert gestaltet und damit gestraffter.

Die jeweils mit dem Testziel verbundene Qualität des Produktes wird messbar.

Testobjekt und Testziel (Fortsetzung)

FT

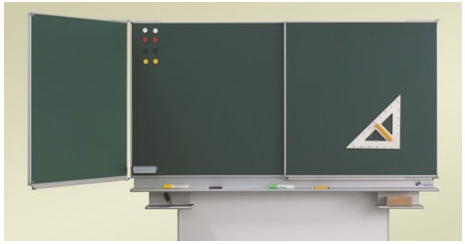
Testziel

Test der Funktionalität (FT – Funktionalitäts- bzw. Modultest)

Dieser Test bezieht sich auf die **Verifikation der Algorithmen** eines Testobjekts (z.B. einer Methode) **auf Anweisungsebene**.

Der Funktionalitätstest auf der Anweisungsebene kann nur bei Testobjekten mit einem hohen Detaillierungsgrad (beispielsweise Teilfunktionalität, Ausschnitt eines Geschäftsprozesses) durchgeführt werden.

Testobjekt und Testziel (Fortsetzung)



I1

Testziel

Test der Steuerung (I1 – Integrationstest 1)

Dieser Test erfolgt immer innerhalb des definierten Testobjektes und bezieht sich auf die Steuerung eines anderen, angestoßenen Objekts, also auf die im Testobjekt definierte Logik, ein anderes anzustoßen (d.h. dessen Schnittstelle zu verwenden).

Verwende ich die Schnittstelle zu einem anderen Objekt nach meinem Ermessen korrekt? Ein “Dummy-Gegenüber” hilft.

Testobjekt und Testziel (Fortsetzung)

I2

Testziel

Test der Schnittstellen (I2 – Integrationstest 2)

Dieser Test erfolgt zwischen dem definierten Testobjekt und einem anderen, referenzierten Objekt, wobei der Test sowohl in Richtung zum aufgerufenen Objekt als auch zum aufrufenden Objekt betrachtet werden muss.

Verwenden eigenes **und** Partner-Testobjekt die Schnittstelle in Kooperation korrekt?

Testobjekt und Testziel (Fortsetzung)

IH

Testziel

Überprüfung datenorientierter Inhalte (IH – Inhaltstest)

Der inhaltliche Test bezieht sich bei Masken und Listen oder Dokumenten auf die Überprüfung der adressatengerechten Information (richtiges Attribut auf richtiger Maske).

Fehler werden nur bei der geschickten Wahl der Testfälle sichtbar. In **mehrere** Felder einer Maske **dieselben** Testdaten einzugeben, macht diese Fehler unsichtbar.

Testmanagement in Softwareprojekten

Testobjekt und Testziel (Fortsetzung)

ER

Testziel

Überprüfung von Masken und Listen auf Ergonomie (ER – Ergonomie)

Bei der Bearbeitung kann beispielsweise überprüft werden, ob die Erfassungssequenz (die Reihenfolge der Eingabefelder) in Masken der natürlichen Lesesequenz des Bedieners oder den Informationsfolgen auf Erfassungsbelegen entspricht).

Testobjekt und Testziel (Fortsetzung)

ST

Testziel

Test der fachlichen Abläufe (ST – Standard)

Dieser Test prüft alle Möglichkeiten des Prozessablaufes, die in der Realität existieren.

Er ist die Zusammenfassung des Tests der Steuerung, der Funktionalität sowie der Schnittstellen aus fachlicher Sicht auf der Basis einer einzelnen Teilfunktionalität oder Funktionalität.

Die im Pflichtenheft als **Akzeptanzkriterien der Userstories** angegebenen Tests können verwendet warden.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testobjekt und Testziel (Fortsetzung)

Testobjekt vs. Testziel

Nachdem Testobjekte sowie Testziele festgelegt wurden, muss betrachtet werden, welche Testziele in Bezug auf welche Testobjekte prinzipiell sinnvoll sind. Eine Matrix stellt die im vorangegangenen beschriebenen Testobjekte und Testziele einander gegenüber, denn nicht jedes Testziel ist auf jedes Testobjekt anwendbar:

Testobjekt / *Testziel*		Funkt. **FT**	Integr. 1 **I1**	Integr. 2 **I2**	Inhalt **IH**	Ergon. **ER**	Stand. **ST**
Teilfunktionalität	**TF**	√	√	√			
Funktionalität	**FK**	(√)	√	√			
Maske (Eingabe)	**MA**	√ (1)	√ (1)	√	√	√	√
Informationsobj.	**IO**	√ (2)	√ (2)	√ (2)	√		
Liste (Ausgabe)	**LI**				√	√	√
Anwendung	**ANW**		√	√	√	√	√
System	**SYS**		√	√	√	√	√

(1) Plausibilitätsprüfungen
(2) Triggers, Stored Procedures

Clientseitige Feldvalidierung

Datenbankseitige Programmierung z.B. PL/SQL.

Testmanagement in Softwareprojekten

Testobjekt und Testziel (Fortsetzung)

Puh, fertig...

Softwarequalität und -test

5. Vorlesung

„Testmanagement in Softwareprojekten“

(Testphasen, -vorbereitung und -umgebung)

www.beuth-hochschule.de

Dipl.-Inform. Thomas Ziemer

Testphasen

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Testphase

Der Testplan definiert die im Testprozess vorhandenen Phasen, und welche Testobjekte in ihnen jeweils getestet werden.

Jede Entwicklungsphase im Softwareprojekt hat eine ihr entsprechende Testphase:

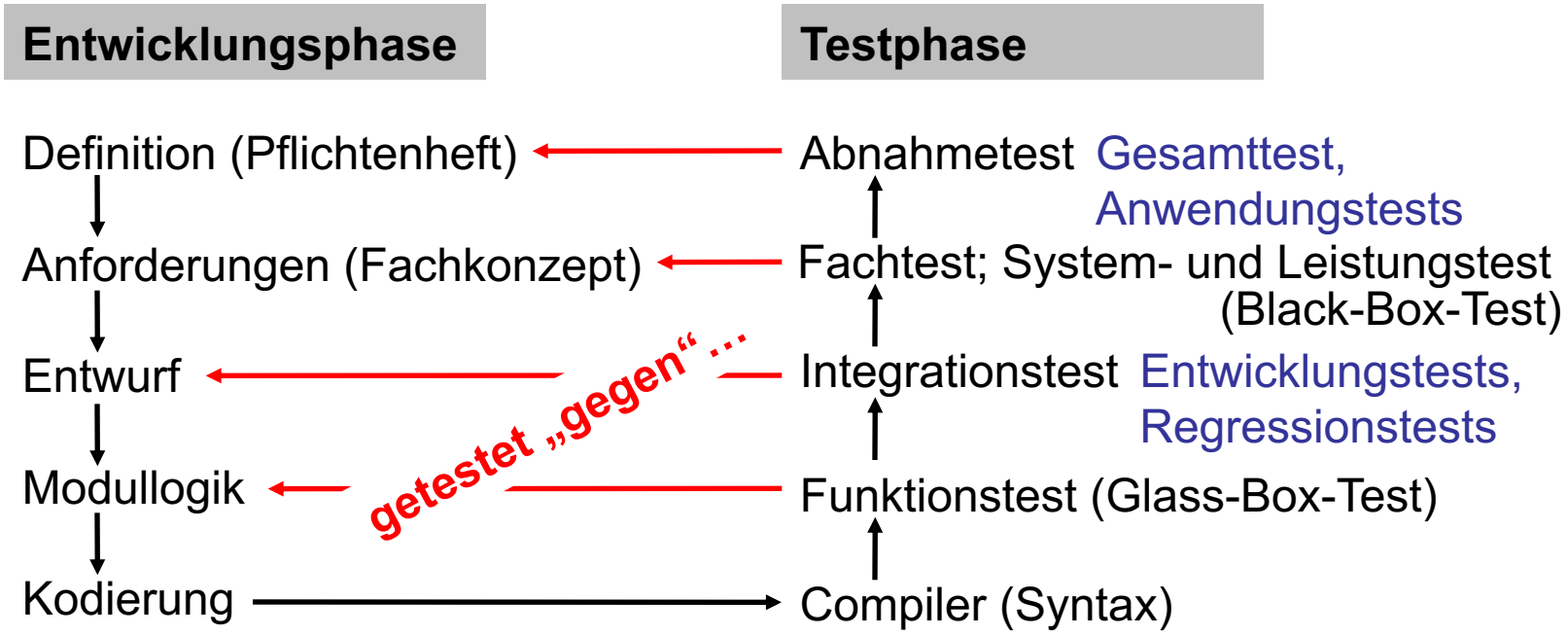
Softwarequalität und -test

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Regressionstest =
Wiederholung eines Tests nach einer Programmänderung

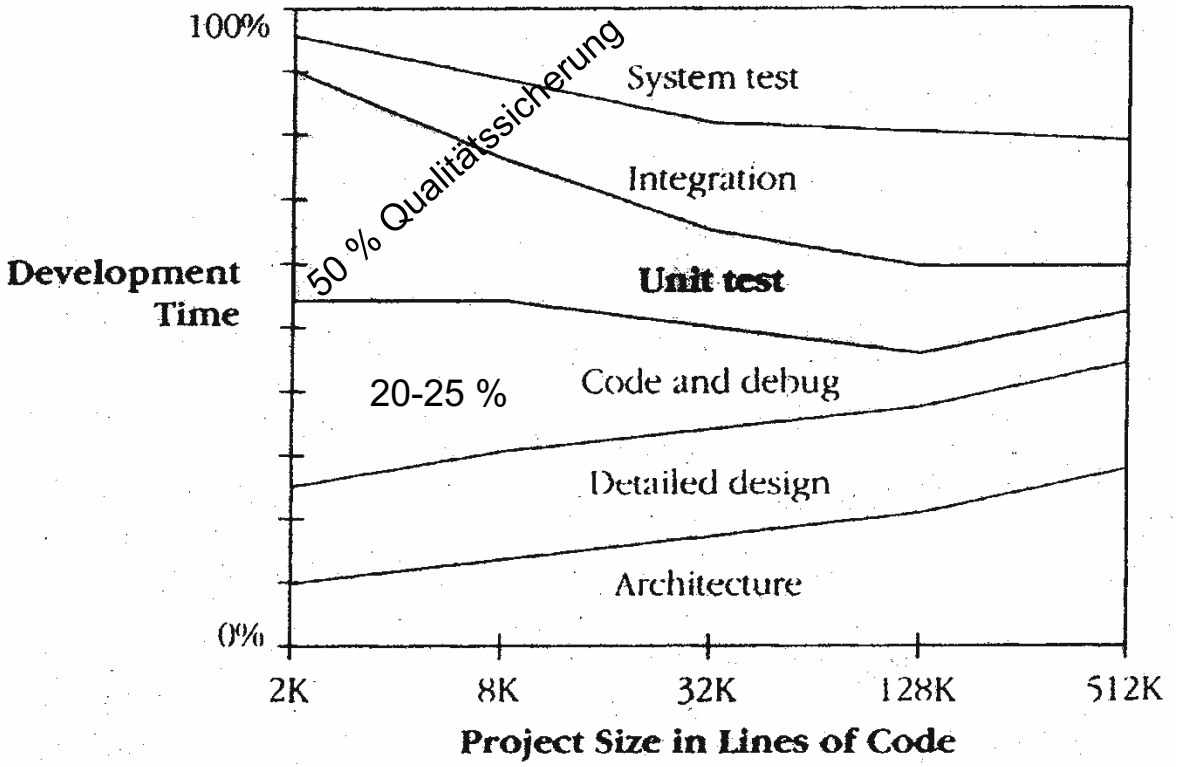
Testphase



Softwarequalität und -test

Testmanagement in Softwareprojekten

Anteil der Entwicklungs- und Testphasen an der Gesamtentwicklung



Quelle: McConnel: Code Complete, Microsoft Press, 1993

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Testphase

Entwicklungstest

Im Rahmen der Entwicklungstests (“Entwicklertest”) werden die Testobjekte

TF	Teilfunktionalität
FK	Geschäftsprozess bzw. Funktionalität
MA	Maske
IO	Datei oder Datenbank
LI	Liste oder Dokument

in (möglicherweise autarken) Testumgebungen gegen die Vorgabe getestet.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Testphase

Anwendungstest

Nach Durchführung der Entwicklungstests und Erreichen eines definierten Fertigstellungsgrades (Status) werden die

Anwendungen

(anwendungsintern) in (möglicherweise autarken) Testumgebungen getestet. Hierfür sollten pro Anwendungsbereich getrennte Umgebungen zur Verfügung stehen.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Testphase

Gesamttest

Nach Durchführung der Anwendungstests und erreichen eines definierten Fertigstellungsgrades werden die

Systeme

(anwendungsübergreifend) in einer gemeinsamen Testumgebung (Vorproduktion) getestet. In dieser Umgebung findet eine Produktionssimulation der Abläufe statt, so weit dies möglich ist.

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Testphase

Abnahmetest

Nach Durchführung der internen Systemtests beim Auftragnehmer wird gemeinsam mit dem Auftraggeber der Abnahmetest durchgeführt.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Testaktivität

Um die projektspezifisch benötigten Testaktivitäten und damit den eigentlichen Testprozess planen zu können, müssen, ausgehend von den gegebenen Qualitätsanforderungen (siehe **Messungen im QM-Handbuch**, die entsprechenden Tätigkeiten zum Erreichen der Testziele in Relation zu den definierten Testobjekten definiert werden.

Die Testaktivitäten ergeben sich aus den ausgewählten Relationen zwischen Testobjekten und Testzielen und beziehen sich prinzipiell auf ein einzelnes Testobjekt.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Testaktivität

Die Relation Testziel (Testziel) zu Testobjekt wird in der projektspezifischen Testmatrix dargestellt:

Aktivitätenliste zur Funktionalität_X

Testobjekt	Testaktivität	
Funktionalität_X	(I1)	Test der Steuerung der Teilfunktionalität(en)
	(I1)	Test der Steuerung der Maske(n)
	(FT)	Test der Funktion der *Funktionalität_X*
	(I2)	Test der Schnittstelle zu *Teilfunktionalität_A*
	(I2)	Test der Schnittstelle zu *Teilfunktionalität_B*
	...	

Testmatrix

Testaktivitäten

Funktionalitätstest (*FT*)
Integrationstest 1 (*I1*)
Integrationstest 2 (*I2*)
Standard (*ST*)
Inhaltstest (*IH*)
Ergonomie (*ER*)

Aufwandsschätzung

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Aufwandsschätzung

Wie auch bei den Aufwandsschätzungen für andere Projektaktivitäten sind meist nur die persönlichen Erfahrungen des Schätzers verfügbar.

Besser ist es, auf Werte aus dem Projektcontrolling bzw. dem innerbetrieblichen Rückmeldesystem zurückzugreifen, falls diese vorhanden sind.

Dies setzt bereits abgeschlossene oder zumindest in Bearbeitung befindliche andere Projekte voraus.

Testmanagement in Softwareprojekten

Testplan (Fortsetzung)

Aufwandsschätzung

Prozentsatzverfahren

Beim Prozentsatzverfahren handelt es sich um eine Auflistung der Anteile der verschiedenen Testaktivitäten (in Prozent) am gesamten Projektaufwand.

Z.B.: "*Für Tests werden 10 % des Aufwands der Entwicklungsphase geplant.*"

Dieses Verfahren kann eingesetzt werden, um schon zu einem frühen Zeitpunkt eine grobe Aufwandsschätzung für den Testplan abgeben zu können.

Testplan (Fortsetzung)

Aufwandsschätzung

Analogieschätzverfahren

Die Basis für das Analogieschätzverfahren bilden entweder bereits erfasste Rückmeldungen über benötigte Aufwände für die verschiedenen Testaktivitäten im Bezug auf die Testobjekte oder, wenn diese nicht verfügbar sind,

die eigenen Erfahrungswerte des Schätzers.

Testvorbereitung

Testvorbereitung

Die Testvorbereitung beschäftigt sich mit der Auswahl der Methoden, mit denen Testfälle und Testdaten erstellt werden, sowie mit der Erstellung selbst.

- Anwendung auf Testdaten
- Auswertung der Ergebnisse

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Methode der Testfallermittlung

Manchmal hört man den Ausdruck, ein Programm „sei ausgetestet“. Es scheint auf den ersten Blick vernünftig, einfach sämtliche Möglichkeiten eines Programms auszuprobieren. Aber geht das überhaupt?

Nehmen wir einmal an, eine Operation habe drei Parameter. Jeder davon sei mit 16 Bits darzustellen. Es gibt also 2^{16} mögliche Werte pro Parameter, zusammen 2^{48} Werte. Das sind *281.474.976.710.656* (281 Billionen!) verschiedene Parameterinhalte. Davon ausgehend, dass in jeder Sekunde 100 mal das Programmstück aufgerufen und das Ergebnis mit dem korrekten Resultat verglichen werden könnte, benötigte man für einen vollständigen Test aller Kombinationen fast *90.000 Jahre*. Vollständiges Testen ist offensichtlich kaum durchführbar.

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Methode der Testfallermittlung

Es wird festgelegt, welche Technik der Testfalldefinition (beispielsweise **Grenzwertanalyse**, prozessorientierte, optimierte Entscheidungstabellen-technik, vollständiges Testen etc.) zum Einsatz kommen soll.

Diese Testfälle werden in einer Testfallschablone abgebildet, die genau der Vorgabe hinsichtlich der Testabdeckung (C^0, C^1, C^2, C^∞) entspricht.

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Testfall

Ein Testfall…

…ist eine Verarbeitungseinheit im Testprozess mit logischen bzw. fachlichen Zusammenhängen und Abhängigkeiten der einzelnen Testschritte untereinander.

…ist rücksetzbar (bspw. in der Datenbank und restlicher Testumgebung).

…hat eine eindeutig spezifizierte Datenbasis.

Zu einem Testfall wird das Testobjekt, die Testumgebung und das erwartete Ergebnis (Soll) angegeben. Wichtig ist insbesondere, dass Testfälle VOR der Testdurchführung formuliert sein müssen.

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Testfall

Das Ziel der technischen Testfälle ist „der Beweis“, dass die Funktionalität des Systems bzw. Programms gewährleistet ist. Ihre Erstellung kann nach dem Entwurf des Systems erfolgen, wenn alle technischen Anforderungen definiert sind. Sie können auch aus dem Pflichtenheft stammen (Akzeptanzkriterien).

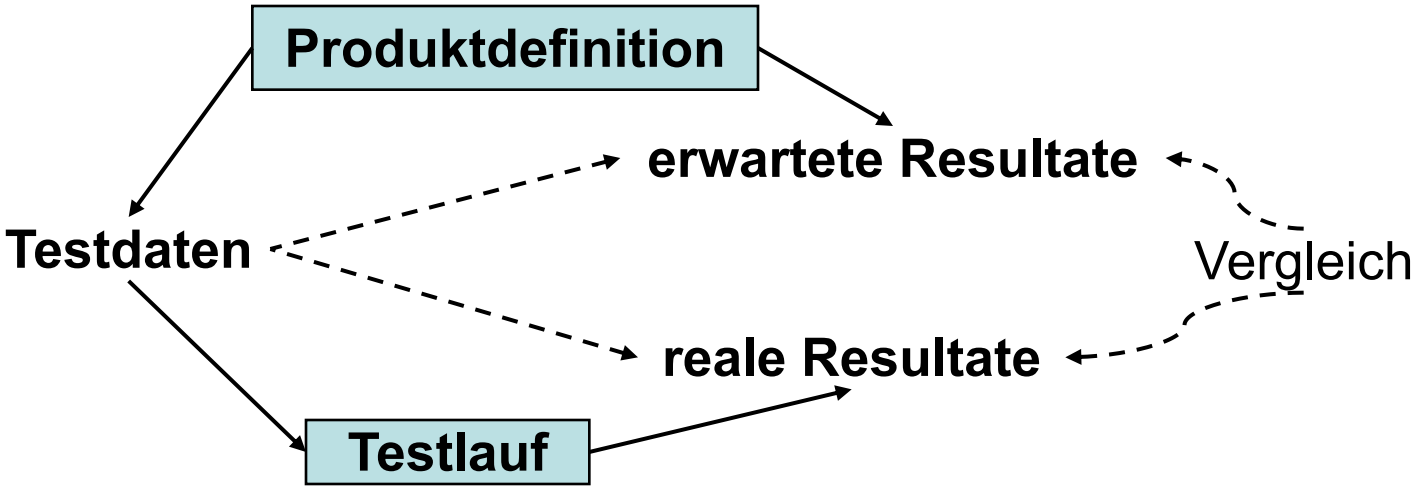
Ziel der fachlichen Testfalldefinition ist der Beweis, dass das System oder Programm die fachlichen Anforderungen erfüllt. Weiter müssen die Dateninhalte auf ihre Richtigkeit überprüft werden. Prinzipiell werden diese Testfälle auf der Basis der fachlichen Vorgaben erstellt.

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Testfall



D. Macos nach K. Bothe, Software Engineering

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Testfall

Es muss alles dokumentiert werden, was benötigt wird, um den Testfall zu reproduzieren. Das erleichtert die Testdurchführung überhaupt und ermöglicht es, nach dem Finden eines Fehlers und dessen Korrektur den Testfall unter denselben Bedingungen erneut durchführen zu können.

Die Eingabedaten gehören zum Testfall. Sie werden Testdaten genannt (dazu später mehr). Ebenso benötigt man die Sollresultate, damit beurteilt werden kann, ob ein Test erfolgreich war.

Ein Testfall

ID	*Eingabe*	*Sollresultat*	*Kommentar*
T01	4	„Kind: gratis“	
T02	14	„jugendlich: 2€“	
T03	**18**	**„volljährig: 3€“**	**Gerade erst volljährig**
T04	19	„volljährig: 3€“	
T05	77	„volljährig: 3€“	Senioren zahlen voll!
T06	130	„Fehleingabe“	Unmögliches Alter (noch…)
T07	0	„Kind: gratis“	Baby unter einem Jahr
T08	-1	„Fehleingabe“	Sinnlose Eingabe abfangen
T09	„Kind“	(nicht möglich)	Buchstaben nicht akzeptieren

Softwarequalität und -test

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Fehlerhaftes Testfallergebnis

Aber was ist eigentlich ein “Fehler”, der während der Durchführung eines Tests gefunden wurde? Auf den ersten Blick scheint das klar zu sein: wenn das Programm sich „falsch verhält“. Falsch oder richtig bezieht sich in einem Softwareprojekt immer darauf, was in der Spezifikation steht. Wenn man aber beim Testen feststellt, dass ein Resultat nicht mit dem erwarteten Wert überein-stimmt, muss nicht immer ein fehlerhaftes Programm der Grund sein.

Sollwert korrekt?		
	nein	Kein Sollwert dokumentiert
		Anforderung falsch verstanden
		Sollwert falsch ermittelt
	ja	Vergleich unangemessen
		Istwert „falsch“

nur dies ist ein Fehler

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Methoden der Testdatendefinition

Die Testdaten werden auf der Basis der definierten Testfälle gebildet. Die Menge und die Qualität der Testdaten ist von der jeweiligen Testphase abhängig. Für den Modultest gilt:

wenige, aber hoch-qualifizierte Testdaten.

Beispielsweise werden für die Überprüfung von Ausdrucken wenige entsprechende Testdaten benötigt. Für den Systemtest und den systemübergreifenden Test gilt den Anforderungen entsprechend:

viele, jedoch nicht so hoch qualifizierte Testdaten.

Wenn der Modultest in einer autarken Testumgebung stattfindet, müssen sämtliche Testdaten, die von außen kommen, simuliert werden.

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Testdaten

Die Definition der technischen Testdaten sollte möglichst abstrakt erfolgen. Da dies in der Praxis aufgrund der weiteren Verarbeitung auf fehlende Akzeptanz stoßen kann, müssen synthetische (automatisch generierte) Testdaten akzeptiert werden („hoffähig gemacht werden“).

Die Definition der fachlichen Testdaten muss hingegen praxisrelevant erfolgen. Dennoch sollten auch diese Testdaten immer synthetisch erstellt werden.

Testumgebung

Testmanagement in Softwareprojekten

Testmanagement (Fortsetzung)

Definition der technischen Infrastruktur

Die Umgebung, in der Tests ablaufen, hat Einfluss auf die Ergebnisse! Jedes System hat einen Zustand, der die Ergebnisse beeinflusst:

Ein Programm reagiert möglicherweise ganz anders, wenn:

- Rechner stark belastet
- Datenbankinhalt verändert
- Netzwerk belastet
- Hauptspeicherinhalt verändert
- Uhrzeit/Datum verändert
- …

Testmanagement in Softwareprojekten

Testmanagement (Fortsetzung)

Definition der technischen Infrastruktur

Die Definition der Infrastruktur beinhaltet in erster Linie Überlegungen bzgl. der benötigten technischen Testumgebungen.

So ist es bei geplantem Einsatz automatischer Tests mit anschließendem automatischen Vergleich unumgänglich, autarke Testumgebungen zur Verfügung zu stellen.

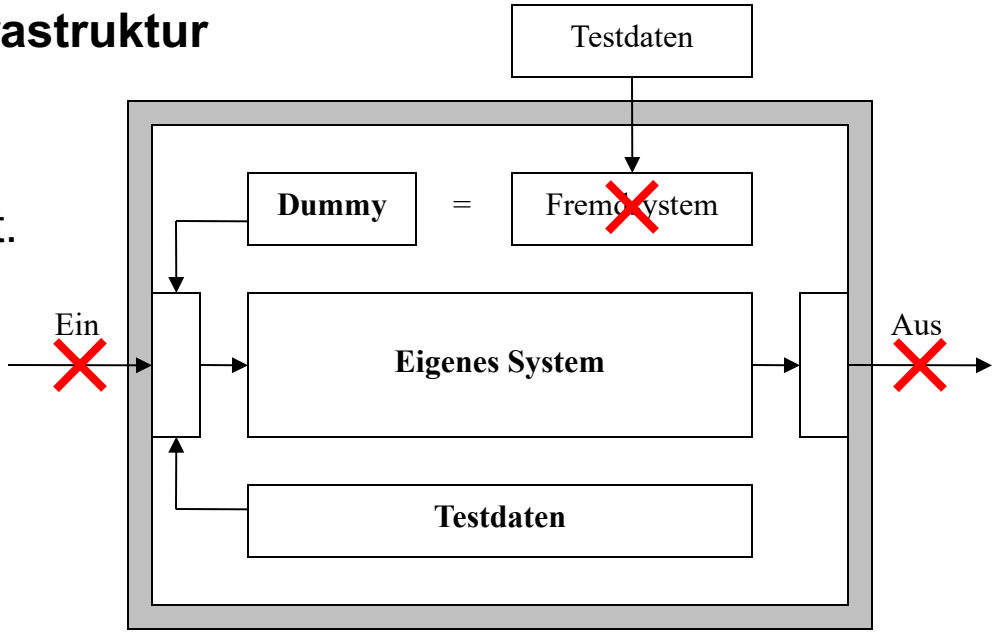
Interessant sind natürlich auch virtuelle Testumgebungen.

Testmanagement in Softwareprojekten

Testmanagement (Fortsetzung)

Definition der technischen Infrastruktur

Die Definition der Testumgebung spezifiziert das Umfeld, in dem später der Test durchzuführen ist.



Autarke Testumgebung

Testmanagement in Softwareprojekten

Testvorbereitung (Fortsetzung)

Puh, fertig...