STUDIENSKRIPT



## Qualitätssicherung im Softwareprozess

IQSS01

Übergeordnete Lernziele

##### Einleitung 9



Zu den begleitenden Aktivitäten eines Softwareprozesses gehört die Qualitätssicherung. Von Beginn an müssen erstellte Artefakte (Dokumente, Modelle, Programmcode) qualitätsgesi- chert werden, denn je später ein Fehler in einem System erkannt wird, desto teurer wird die Fehlerbehebung. Dennoch können in der Praxis industrielle Informationssysteme nicht voll- ständig getestet werden.

Dieser Kurs vermittelt Techniken und Vorgehensweisen zur Qualitätsplanung, Qualitätssiche- rung und Qualitätslenkung. Dabei werden neben konstruktiven Maßnahmen zur Qualitätssi- cherung auch Techniken zur statischen und dynamischen Qualitätssicherung vermittelt. Dazu gehören die Aktivitäten zum systematischen Testen von Softwaresystemen, ebenso zählen wir die Qualitätssicherung von Anforderungsdokumenten, Architekturen und dem Softwarepro- zess selbst dazu.



# Lektion 1

## Einführung in die Softwarequalitätssicherung

#### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie wissen, …

… was die wichtigsten Begriffe aus dem Themenbereich Softwarequalitätssicherung bedeuten.

… was es für Prinzipien der Softwarequalitätssicherung gibt, welche Grundsätze für Softwaretests gelten und was deren Auswirkungen sind.

… warum es in der Regel nicht wirtschaftlich ist, fehlerfreie industrielle Informationssysteme auszuliefern.

DL-D-IQSS01-L01

1. Einführung in die Softwarequalitätssiche- rung

### Einführung

Diese Lektion bietet einen Einstieg in das Thema Softwarequalitätssicherung. Zunächst werden die wichtigsten Begriffe rund um das Thema Qualitätsmanagement vorgestellt. Anschließend werden sieben Prinzipen der Softwarequalitätssicherung benannt und deren Auswirkungen auf die QS-Aktivitäten beschrieben. Anschließend werden Grund- sätze im Softwaretest erläutert, bevor gezielt das Thema Kosten in Bezug auf Qualität betrachtet wird.

### Motivation und Begriffe

Zwar kann ein Softwaresystem erst nach der Fertigstellung ganzheitlich auf die Erfül- lung der geforderten Anforderungen getestet werden, jedoch ist das Risiko zu hoch im Softwareprozess mit der Qualitätssicherung bis zum Ende der Implementierung zu war- ten. Daher wird Qualitätssicherung im Software Engineering begleitend zu allen ande- ren Aktivitäten durchgeführt.

###### Softwarequalität

Ganz allgemein wird der Begriff Softwarequalität in der DIN ISO 9126 folgendermaßen deﬁniert:

„Softwarequalität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software- produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen.“

Demnach müsste man eine Software dahingehend überprüfen, ob das, was ausgeliefert wurde, die vorher festgelegten Anforderungen (Erfordernisse) erfüllt. Es wird die Wich- tigkeit einer Speziﬁkation deutlich: Laut DIN kann über die Qualität von Software nur auf Basis der Speziﬁkation („festgelegte Erfordernisse“) entschieden werden. In der Pra- xis zeigt sich allerdings, dass die wahrgenommene Qualität einer Software in erster Linie dadurch bestimmt wird, ob der Kunde feststellt, dass die Software seine tatsächli- chen Anforderungen erfüllt. Da viele Anforderungen oft erst während der Softwareent- wicklung erkannt werden, muss im Hinblick auf die Kundenzufriedenheit kontinuierlich dafür gesorgt werden, dass die Menge der speziﬁzierten Anforderungen auch die Menge der tatsächlichen Anforderungen enthält.

Neben der Bestimmung der Qualität eines Softwaresystems als Ganzes kann auch zu den im Verlauf der Softwareentwicklung erzeugten Artefakten (Softwaremodelle, Anfor- derungen, Architekturen, Dokumente, Softwarekomponenten) die Qualität bestimmt werden.

Einführung in die Softwarequalitätssicherung

###### Qualitätsmanagement

Unter dem Begriff Qualitätsmanagement (kurz: QM) werden alle organisierten Maßnah- men zusammengefasst, die der Verbesserung der Qualität von Produkten, Prozessen oder Leistungen jeglicher Art dienen. Da es im Software Engineering zu risikoreich ist, mit der Bestimmung der Softwarequalität bis zum Abschluss der Implementierung zu warten, wird schon während des Entwicklungsprozesses mit der Bestimmung der Quali- tät bereits erzeugter Softwarefragmente begonnen. Doch nicht nur der erzeugte Pro- grammcode ist beim Softwarequalitätsmanagement zu berücksichtigen. Da die Imple- mentierung direkt von den in den Aktivitäten Requirements Engineering, Speziﬁkation und Architektur getroffenen Vorgaben und Entscheidungen abhängig ist, pﬂanzen sich Fehler der Speziﬁkation oder der Architekturdeﬁnition unmittelbar auf die Implemen- tierung fort. Daher müssen diese Artefakte ebenfalls beim Qualitätsmanagement im Software Engineering miteinbezogen werden.

Typische Aktivitäten im Qualitätsmanagement sind:

* Qualitätsplanung: Erstellung und Dokumentation der Qualitätsanforderungen gemeinsam mit dem Auftraggeber.
* Qualitätslenkung: Überwachung, Steuerung und Kontrolle von Aktivitäten zur Quali- tätsprüfung im Softwareentwicklungsprozess.
* Qualitätssicherung (kurz: QS): Tätigkeiten, die sicherstellen, dass festgelegte Quali- tätsanforderungen für Produkte, Prozesse und Leistungen erfüllt werden.
* Qualitätsverbesserung: Auswertung von Produkt- und Prozessdaten zur Verbesse- rung des Qualitätsniveaus.

###### Konstruktives und analytisches Qualitätsmanagement

Grundsätzlich lässt sich Qualitätsmanagement in die Bereiche konstruktives und analy- tisches Qualitätsmanagement aufteilen.

Beim konstruktiven Qualitätsmanagement werden a priori (also vor der Erstellung) alle Qualitätseigenschaften für Produkte oder Prozesse deﬁniert, um Fehler während der Softwareentwicklung zu vermeiden und die Qualität der erstellten Artefakte zu gewährleisten beziehungsweise zu erhöhen. Die Maßnahmen zum konstruktiven Quali- tätsmanagement umfassen:

* technische Maßnahmen (z. B. Einsatz von Modellierungssprachen, Werkzeugen, Ent- wicklungsumgebungen),
* organisatorische Maßnahmen (z. B. Richtlinien, Standards, Templates, Checklisten) und
* zwischenmenschliche Maßnahmen (z. B. Trainings, Arbeitsklima, gemeinsame Aktivi- täten).

Qualitätsmanage- ment

Mit Qualitätsma- nagement werden alle organisierten Maßnahmen bezeichnet, die der Verbesserung der Qualität von Produk- ten, Prozessen oder Leistungen dienen: Qualitätsplanung, - lenkung, -sicherung und -verbesserung.

Konstruktives Quali- tätsmanagement Das konstruktive Qualitätsmanage- ment besteht aus A- priori-Maßnahmen zur Fehlervermei- dung vor deren Ent- stehung.

Analytisches Quali- tätsmanagement

Das analytische Qua- litätsmanagement besteht aus Maß- nahmen zur Prüfung und Bewertung des Qualitätsniveaus von

Prüfobjekten.

Beim analytischen Qualitätsmanagement werden ex post (also nach der Erstellung) Maßnahmen zur Prüfung und Bewertung des aktuellen Qualitätsniveaus der Prüfob- jekte durchgeführt, um Fehler systematisch aufzuspüren und ihre Ausmaße bestimmen zu können. Dazu werden sowohl statische als auch dynamische Testverfahren einge- setzt, deren wichtigste Eigenschaften in Abbildung 1 gegenübergestellt werden.

|  |  |  |
| --- | --- | --- |
| Abbildung 1: Vergleich statische und dynamische Verfahren | | |
|  | Analytische Maßnahmen | |
| Statische (analysierende) Verfahren | Dynamische (testende) Ver- fahren |
| Methode | Prüﬂing wird nur begutach- tet, jedoch nicht ausgeführt. | Ausführbare Software wird mit konkreten Eingabewer- ten ausgeführt. |
| Prüﬂing | Artefakte und Prozesse der Softwareentwicklung, z. B. Speziﬁkation, Quelltext, Architekturbeschreibung, Testdokumente, Richtlinien | Ausführbares Programm  bzw. einzelne Komponenten |
| Beispiele | * Durchsicht des Benutzer- handbuchs auf Vollstän- digkeit * Prüfung, ob die Program- mierrichtlinien eingehal- ten wurden | Programm mit konkreten Eingabewerten ausführen und die so erzeugten Ausga- ben mit den erwarteten Aus- gaben vergleichen |
| Beispiele für Test- verfahren | Review, Metriken, statische Codeanalyse | Funktionsorientierte Tests, Zufallstests |
| Verortung im Qualitätsmanage- ment | Qualitätssicherung, teilweise auch Qualitätslenkung | Qualitätssicherung |

Bei statischen Verfahren (auch: statische Tests) wird der Prüﬂing (ein Stück Software) im Rahmen eines Reviews analysiert, begutachtet, untersucht, die dabei gewonnenen Informationen zusammengetragen, gegebenenfalls in Metriken oder Kennzahlen ver- dichtet und schließlich ausgewertet. Bei dynamischen Verfahren (auch: dynamische Tests) wird der Prüﬂing mit konkreten Eingabewerten ausgeführt und das Ergebnis der Ausführung bewertet.

Einführung in die Softwarequalitätssicherung

Durch vorausschauendes konstruktives Qualitätsmanagement kann der Aufwand im analytischen Qualitätsmanagement reduziert werden.

###### Testgegenstand

Grundsätzlich können alle im Software Engineering erstellten Artefakte auf die Einhal- tung von Qualitätsanforderungen getestet werden. Das bedeutet, dass neben dem erzeugten Programmcode auch

* fachliche Anforderungen,
* technische Speziﬁkationen,
* Architekturbeschreibungen und selbst die
* Testfälle

getestet werden können. Dynamische Tests sind jedoch nur bei ausführbaren Artefak- ten möglich. Alle anderen Artefakte werden durch statische Testverfahren geprüft.

### Prinzipien der SW-Qualitätssicherung

Unabhängig von konkreten Techniken oder Vorgehensweisen, beschreibt Balzert (1998,

S. 284–293) die folgenden Prinzipen der Softwarequalitätssicherung, die als zu berück- sichtigende Grundsätze bei der konkreten Gestaltung von QS-Aktivitäten und QS-Proz- essen dienen.

###### Prinzip der produkt- und prozessabhängigen Qualitätszielbestimmung

Das Prinzip der produkt- und prozessabhängigen Qualitätszielbestimmung bedeutet, dass in frühen Phasen des Projektes jeweils die konkreten Qualitätsziele bestimmt wer- den sollen, die für das Softwareprodukt beziehungsweise für den Softwareprozess rele- vant sind. Nur wenn es dokumentierte Qualitätsziele gibt, können diese später von den Architekten und Entwicklern bei der Implementierung gezielt berücksichtigt und adres- siert werden.

In der industriellen Praxis ﬁndet eine gezielte A-priori-Bestimmung konkreter Quali- tätsziele oft jedoch nicht statt. Das liegt nicht zuletzt daran, dass die Kompetenzen zur methodischen Ermittlung und Speziﬁkation von Qualitätszielen auf Basis deﬁnierter Qualitätsmodelle häuﬁg fehlen. Da die potenzielle Menge an Vorgehensweisen und Techniken zur Qualitätssicherung sehr groß ist, kann eine zielgerichtete Umsetzung der geforderten Eigenschaften nur dann in Angriff genommen werden, wenn es formulierte Qualitätsziele gibt.

###### Prinzip der quantitativen Qualitätssicherung

Das Prinzip der quantitativen Qualitätssicherung bedeutet, dass Aussagen über die Qualität eines Systems oder Prozesses anhand von Kennzahlen (auch: Metriken) getrof- fen werden, zu denen es festgelegte Sollwerte gibt. Die tatsächlich gemessenen Ist- werte werden dazu mit den Sollwerten verglichen.

Beispiel

„Beim ersten Release der Anwendung müssen alle bekannten kritischen Fehler, aber nur 70 % aller bekannten nichtkritischen Fehler behoben sein.“

Zwar hat sich das Messen von Eigenschaften als praktisch herausgestellt, jedoch gibt es keine Standardmetriken, die zu jedem Softwareprozess eingesetzt werden. Welche Werte wirklich gemessen werden, ist abhängig von der Organisationstruktur des Pro- jekts, dem fachlichen Umfeld sowie den Anforderungen an das Produkt und an den Softwareprozess.

###### Prinzip der maximalen konstruktiven Qualitätssicherung

Mit dem Prinzip der maximalen konstruktiven Qualitätssicherung ist gemeint, dass beim Qualitätsmanagement alle sinnvollen Maßnahmen ergriffen werden sollen, die zur Vermeidung von Fehlern beitragen. Die Annahme der konstruktiven Qualitätssiche- rung ist, dass durch eine hohe Prozessqualität eine hohe Produktqualität erreicht wer- den kann.

###### Prinzip der frühzeitigen Fehlerentdeckung und -behebung

Mit dem Prinzip der frühzeitigen Fehlerentdeckung und -behebung soll das Erkennen von Fehlern bereits vor deren Manifestation im Programmcode gewährleistet werden. Bevor die eigentliche Implementierung beginnt, erfolgt das Requirements Engineering (RE), die Speziﬁkation und die Erstellung einer Architekturdeﬁnition. Fehler im RE set- zen sich über die Speziﬁkation und die Architekturdeﬁnition bis in die Implementierung fort. Je später ein Fehler identiﬁziert wird, desto höher sind die Kosten der Fehlerbehe- bung. Wird ein Fehler erst bei der Auslieferung eines Systems entdeckt, müssen unter Umständen alle Phasen bis zur Implementierung erneut durchlaufen werden, um ihn zu beheben.

Wird jedoch beispielsweise ein Fehler im Anforderungsdokument noch vor der Speziﬁ- kation entdeckt, müssen nur die betroffenen Stellen der Anforderungsdokumentation angepasst werden. Die Behebung von Fehlern in frühen Phasen ist daher deutlich kos- tengünstiger und schneller zu erledigen, als Fehler, die sich vom fachlichen Anforde- rungsdokument bis in die Implementierung durchziehen. In der Praxis wird ein großer Anteil der Fehler, die in frühen Phasen verursacht werden, jedoch erst beim Abnahme-

Einführung in die Softwarequalitätssicherung

test entdeckt. Um das zu verhindern, dürfen strukturierte QS-Maßnahmen nicht erst beim Testen der Software beginnen, sondern bereits mit der Erzeugung der ersten Artefakte in einem Softwareprozess.

###### Prinzip der entwicklungsbegleitenden, integrierten Qualitätssicherung

Dieses Prinzip ist die Konsequenz des Prinzips der frühzeitigen Fehlerentdeckung. Um Fehler möglichst frühzeitig zu erkennen, müssen Aktivitäten der Qualitätssicherung begleitend zu den Entwicklungsaktivitäten erfolgen und dabei fest in den Softwarepro- zess integriert sein. Eine konsequente Anwendung dieses Prinzips erfordert eine deﬁ- nierte Qualitätssicherung aller im Softwareprozess erzeugten Artefakte, sodass sicher- gestellt ist, dass sich Fehler nicht von einer Aktivität in die nächste fortpﬂanzen. Besonders prägnant wird dieses Prinzip vom Prozessparadigma evolutionäre Entwick- lung umgesetzt. Hierbei wird die Software in mehreren Iterationen entwickelt, deren Ergebnis jedes Mal auch das Testen der Software umfasst. So können Fehler rechtzeitig erkannt und deren Fortpﬂanzung in alle nachfolgenden Iterationen verhindert werden.

###### Prinzip der unabhängigen Qualitätssicherung

Das Prinzip der unabhängigen Qualitätssicherung besagt, dass die Überprüfung der erzeugten Artefakte möglichst objektiv und ohne Abhängigkeiten von individuellen Interessen erfolgen sollte. Soll beispielsweise der Programmcode von dem Team getes- tet werden, das den Code erstellt, wird häuﬁg die für die Qualitätssicherung vorgese- hene Zeit nicht in dem Maße für QS genutzt, wie es erforderlich ist, sondern es werden eher weitere Funktionen implementiert. Wird die QS allerdings vollständig aus der Ver- antwortung des Entwicklungsteams genommen, lässt sich häuﬁg eine nachlassende Sorgfalt bei der Entwicklung beobachten, da Andere für das Aufﬁnden von Fehlern zuständig sind.

In der Praxis wird häuﬁg ein Modell eingesetzt, das beide Aspekte berücksichtigt: Das Entwicklungsteam ist in der Pﬂicht, die Qualität seiner Ergebnisse bis zu einem quanti- tativ festgelegten Maß sicherzustellen. Die Qualitätssicherung prüft dann alle über die- ses Maß hinausgehenden Aspekte. Beispiel: Das Entwicklungsteam muss gewährleisten, dass zu jeder implementierten Funktion automatische Testfälle existieren, die alle erfolgreich durchlaufen werden. Darüber hinaus müssen besonders kritische Codeab- schnitte einen manuellen Review durchlaufen haben. Die Qualitätssicherung testet darauf aufbauend die fachlichen Anwendungsfälle, also das richtige Zusammenspiel aller Entwicklungsartefakte.

### Grundsätze im Softwaretest

Ergänzend zu den Prinzipien der Softwarequalitätssicherung, können gezielt grundsätz- liche Eigenschaften und Phänomene zu Softwaretests beschrieben werden. Bei Spill- ner/Linz (2019, S. 22f) werden dazu unter anderem die folgenden Grundsätze im Softwa- retest vorgestellt.

###### Testen zeigt die Anwesenheit von Fehlern

Mit der Durchführung von Tests können Fehler im System identiﬁziert werden, das bedeutet, anwesende Fehler können gefunden werden. Zur Abwesenheit von Fehlern kann jedoch keine Aussage getroffen werden, auch wenn alle Tests erfolgreich sind. Durch ausführliches Testen wird zwar die Wahrscheinlichkeit reduziert, dass es noch unentdeckte Fehler gibt, eine Fehlerfreiheit kann durch Softwaretests jedoch nicht nachgewiesen werden.

###### Vollständiges Testen ist nicht möglich

Das vollständige Testen, insbesondere von industriellen Informationssystemen, ist nicht möglich. Die Menge der möglichen Eingabewerte und die Menge möglicher Sys- temzustände sind bei industrieller Software so groß, dass ein vollständiger Test entwe- der technisch gar nicht möglich ist oder in der praktischen Durchführung so viele Res- sourcen erfordern würde, dass das zu testende System niemals wirtschaftlich betrieben werden könnte. Daher sind die tatsächlich durchgeführten Softwaretests immer nur Stichproben. Die Entscheidung, welche Funktionen mit welchem Aufwand und welchen Eingabeparametern getestet werden, muss daher bewusst und sehr sorgfältig getroffen werden.

###### Häufung von Fehlern

Die entdeckten Fehler in Softwaresystemen sind in der Regeln nicht über alle System- komponenten gleich verteilt. Häuﬁg konzentriert sich ein Großteil der Fehler auf nur wenige Komponenten. Wird also ein Fehler an einer Stelle X im System durch Tests

Einführung in die Softwarequalitätssicherung

nachgewiesen, so liegen mit hoher Wahrscheinlichkeit im Umfeld von X noch weitere Fehler vor. Das Phänomen der Fehlerhäufung ist daher bei der Durchführung von Soft- waretests zu berücksichtigen. Im Umfeld von bereits identiﬁzierten Fehlern muss besonders sorgfältig und gegebenenfalls mit mehr als den ursprünglich geplanten Test- fällen getestet werden.

###### Wiederholungen haben keine Wirksamkeit

Die mehrfache Ausführung der gleichen Testfälle mit den gleichen Eingabedaten im gleichen Systemzustand und auf Basis des gleichen Datenbestands sowie ohne Anpas- sungen am Programmcode erzielten keine Wirkung, die über eine einmalige Ausführung der Testfälle hinausgeht. Das mehrfache Ausführen des gleichen Tests unter identi- schen Bedingungen führt daher zu keinem Erkenntnisgewinn über eventuell vorlie- gende Fehler.

###### Testen ist abhängig vom Umfeld

Korrespondierend zum oben genannten Prinzip der produkt- und prozessabhängigen Qualitätssicherung werden Anzahl und Art der Testfälle für jedes System neu bestimmt und formuliert. Alle Aktivitäten – von der Konzeption über die Erstellung bis hin zur Durchführung und Auswertung von Testfällen – sind für jedes Projekt einmalig. Sie wer- den auf Grundlage der Anforderungen an das System, den Randbedingungen des Pro- jektes und den formulierten Qualitätszielen bestimmt. Streng genommen handelt es sich, insbesondere beim Einsatz von automatischen Softwaretests, um ein „Software- projekt im Softwareprojekt“, das von der Anforderungsanalyse bis zur Implementierung alle Phasen durchläuft.

###### Trugschluss: Keine Fehler bedeuten nicht automatisch ein brauchbares System.

Die Qualität von Software und damit auch die Formulierung von Testfällen erfolgt auf Basis der Speziﬁkation des Systems. Aspekte, die vielleicht für die Nutzer wichtig sind, jedoch bei der Speziﬁkation nicht berücksichtigt wurden, werden in der Regel nicht getestet. Daher kann selbst ein sehr gründlich getestetes System, in dem alle erkannten Fehler beseitigt wurden, aus Nutzersicht unbrauchbar sein. Wenn ein System beispiels- weise in seiner Betriebsumgebung als Teil der Anwendungslandschaft bei einer hohen Anzahl von gleichzeitigen Nutzern sehr langsam wird oder nur eingeschränkt verfügbar ist, liegt zwar kein Softwarefehler im Sinne einer fehlerhaften Systemfunktion vor, jedoch werden erwartete Qualitätseigenschaften nicht erfüllt. Dies wiederum beein- ﬂusst die Nutzbarkeit und die wahrgenommene Systemqualität. Zusammengefasst wer- den kann dieser Grundsatz mit folgendem Zitat des Informatikpioniers Edsger Dijkstra

„Testen kann nur die Anwesenheit von Fehlern aufzeigen, aber niemals deren Abwesen- heit beweisen.“

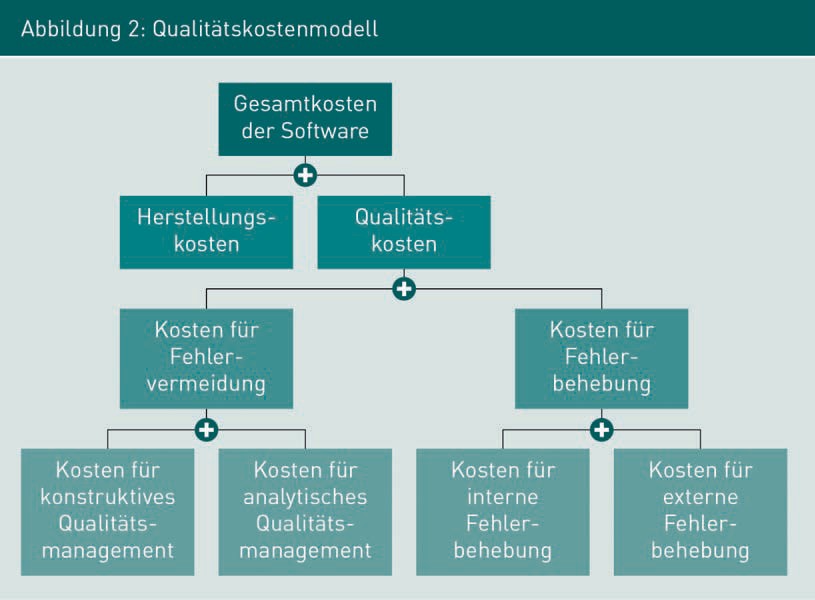
Phänomen der Feh- lerhäufung

Fehler sind in der Regel nicht gleich- verteilt. Wird ein Fehler an einer Stelle X im System durch Tests nachge- wiesen, so liegen mit hoher Wahrschein- lichkeit im Umfeld von X noch weitere Fehler vor.

### Kosten von Qualität

Wie oben bereits beschrieben, können Softwaretests weder vollständig durchgeführt werden, noch kann durch Tests die Fehlerfreiheit von Software nachgewiesen werden. Zu jedem Softwareprojekt muss individuell entschieden werden, wie viele Ressourcen für das Qualitätsmanagement aufgewendet werden können. Zur Abwägung und zur Ent- scheidungsﬁndung dient dabei das in Abbildung 2 dargestellte Qualitätskostenmodell, das sich an dem von Grechenig (2010, S. 468–470) vorgestellten Modell orientiert.

Die Gesamtkosten eines Softwaresystems werden durch die Herstellungs- und die Qua- litätskosten bestimmt. Als Herstellungskosten werden die Kosten bezeichnet, die zur Herstellung der gewünschten Funktionen und der gewünschten Qualitätseigenschaften aufgewendet werden. Die Qualitätskosten hingegen umfassen die Aufwände, die für die Fehlervermeidung, -erkennung, -lokalisierung und -behebung aufgebracht werden.

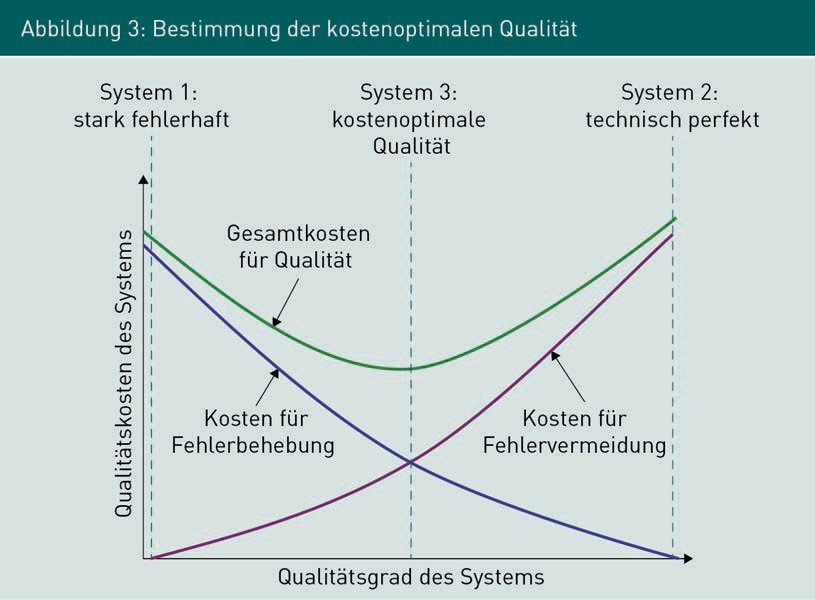


Konkret setzen sich die Qualitätskosten aus den Kosten für die Fehlervermeidung und den Kosten für die Fehlerbehebung zusammen. Zur Fehlervermeidung zählen alle Akti- vitäten, die innerhalb des Softwareprozesses vor der Auslieferung des Systems mit dem Ziel durchgeführt werden, dass nach der Auslieferung des Systems möglichst keine Fehler mehr entdeckt werden. Die Kosten der Fehlervermeidung umfassen damit alle Aktivitäten zum konstruktiven und analytischen Qualitätsmanagement innerhalb des Softwareprojekts.

Einführung in die Softwarequalitätssicherung

Kosten für die Fehlerbehebung entstehen dann, wenn nach Auslieferung des Systems Fehler entdeckt werden. Mit den Aufwänden zur Lokalisierung und Behebung von Feh- lern kommen interne Kosten für die Fehlerbehebung zum Tragen. Dazu zählen insbe- sondere auch die Aufwände für Softwaretests, die nach Anpassungen am Quellcode durchgeführt werden müssen. Allerdings entstehen auch externe Kosten, beispielsweise durch Vertragsstrafen. Ferner muss häuﬁg mit nicht genau zu beziffernden Schäden gerechnet werden, beispielsweise durch Vertrauens- und Imageverlust des Software- herstellers auf Seiten des Kunden.

Die Entscheidung, wie Ressourcen für die Fehlervermeidung zur Verfügung gestellt wer- den können, muss in der industriellen Praxis immer unter wirtschaftlichen Gesichts- punkten getroffen werden. Zur Veranschaulichung dieses Spannungsfeldes dient die in Abbildung 3 gezeigte Graﬁk, in der die Qualitätskosten eines Systems in Abhängigkeit zum Qualitätsgrad eines Systems dargestellt sind.



Die violette Linie veranschaulicht die Kosten der Fehlervermeidung – diese sind nicht linear abhängig zum erzielten Qualitätsgrad. Bis zu einem bestimmten Punkt im Soft- wareprozess tragen Maßnahmen zur Fehlervermeidung verhältnismäßig schnell zu einem höheren Qualitätsgrad bei. Je mehr und je intensiver jedoch Qualitätssicherung betrieben wird, desto schwerer wird es, weitere Fehler zu ﬁnden. Die Aufwände zur Qua- litätssicherung steigen überproportional hoch an im Verhältnis zur tatsächlichen Erhö- hung des Qualitätsgrades. Die blaue Linie veranschaulicht die Kosten zur Fehlerbehe- bung. Ein weitgehend ungetestetes System verursacht sehr hohe Behebungskosten, jedoch sinken die Behebungskosten mit einem höheren Qualitätsgrad immer langsa- mer.

In Abbildung 3 sind exemplarisch drei verschiedene Systemarten mit gestrichelten Linien dargestellt. System 1 ist ein stark fehlerhaftes System. Die Kosten für die Fehler- vermeidung sind dort sehr gering, da fast kein Qualitätsmanagement erfolgte. Die Kos- ten der Fehlerbehebung sind jedoch sehr hoch, da Fehler im System erst nach dessen Auslieferung erkannt werden. Im Gegensatz dazu steht ein technisch perfektes System 2, das rechts dargestellt ist. Bei diesem System entstehen sehr hohe Aufwände durch die Maßnahmen zur Fehlervermeidung, allerdings fast keine Aufwände durch die Feh- lerbehebung. In diesem Fall wurden nahezu alle Fehler bereits vor der Auslieferung erkannt und behoben. System 3 ist ein System, das bezogen auf die Qualitätskosten optimal erstellt wurde. Die wichtigsten Fehler wurden vor der Auslieferung identiﬁziert und beseitigt, aber einige Fehler werden auch erst nach der Auslieferung identiﬁziert und behoben. Die Kosten für Fehlervermeidung und Fehlerbehebung sind bei einer kostenoptimalen Qualität gleich.

Bei der Planung im Rahmen des Qualitätsmanagements ist der Punkt der optimalen Qualitätskosten in einem Projekt anzustreben. Einerseits sind alle Fehler zu vermeiden, die im Nachhinein sehr teuer zu beheben sind. Andererseits sollten die Kosten zur Ver- meidung von Fehlern nicht größer sein als die Kosten, die zur Behebung dieser Fehler entstehen würden. Dazu müssen Risiken mit ihren Eintrittswahrscheinlichkeiten und Folgekosten abgeschätzt werden. Entsprechend den Ergebnissen der Risikoanalyse und dem verfügbaren Projektbudget sind die Maßnahmen zur Qualitätssicherung so zu pla- nen, dass sie möglichst kostenoptimal durchgeführt werden können.

Zusammenfassung

Unter dem Begriff Qualitätsmanagement werden alle organisierten Maßnahmen zusammengefasst, die der Verbesserung der Qualität von Produkten, Prozessen oder Leistungen jeglicher Art dienen. Beim konstruktiven Qualitätsmanagement werden vor der Erstellung alle Qualitätseigenschaften für Produkte oder Prozesse deﬁniert, um Fehler während der Systemerstellung zu vermeiden. Beim analyti- schen Qualitätsmanagement werden im Softwareprozess erstellte Artefakte geprüft. Zu den Aktivitäten der Softwarequalitätssicherung können verschiedene Prinzipien und Grundsätze beschrieben werden, die bei der konkreten Qualitätsplanung und Qualitätssicherung berücksichtigt werden müssen.

Die Entscheidung, welches Budget für die Fehlervermeidung zur Verfügung steht, wird in der industriellen Praxis immer unter wirtschaftlichen Gesichtspunkten getroffen. Bei der Planung im Rahmen des Qualitätsmanagements ist der Punkt der optimalen Qualitätskosten in einem Projekt anzustreben. Die Kosten zur Vermei- dung von Fehlern sollten dabei nicht größer sein als die Kosten, die zur Behebung dieser Fehler entstehen würden. Die tatsächlichen optimalen Aufwände können nur bestimmt werden, wenn die Kosten für die Fehlerbehebung bekannt sind.



# Lektion 2

## Organisation und Planung von Softwarequalität

#### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie wissen, …

… wodurch sich produktorientiertes und prozessorientiertes Qualitätsmanagement unterscheiden.

… was Qualitätsplanung bedeutet und welche Rolle Qualitätsziele dabei spielen.

… welche typischen Aktivitäten zur Qualitätssicherung in einem Softwareprozess geplant werden können.

… wie Total Quality Management (TQM) zur Qualitätslenkung eingesetzt werden kann.

DL-D-IQSS01-L02

1. Organisation und Planung von Software- qualität

### Einführung

Die Qualität einer Software entsteht nicht zufällig, sondern muss systematisch geplant und organisiert werden. Ein aktives Qualitätsmanagement ist insbesondere erforder- lich, da sich die Aktivitäten der Qualitätssicherung zum einen über den gesamten Soft- wareprozess erstrecken und zum anderen für jedes Projekt individuelle Qualitätsziele gelten. Diese Lektion stellt die wichtigsten Aktivitäten zum Thema Qualitätsmanage- ment vor, bevor einzelne Aspekte in den folgenden Lektionen vertieft werden.

### Überblick über den Qualitätsmanagementprozess

###### Kernaktivitäten im Qualitätsmanagement

Der Begriff Qualitätsmanagement ist im Industriestandard ISO 9000 wie folgt deﬁniert:

„Aufeinander abgestimmte Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität, die üblicherweise das Festlegen der Qualitätspolitik und der Quali- tätsziele, die Qualitätsplanung, die Qualitätslenkung, die Qualitätssicherung und die Qualitätsverbesserung umfasst.“ Die für das Qualitätsmanagement erforderlichen Akti- vitäten können – und müssen – im Rahmen der Softwareentwicklung auch für jeden einzelnen Softwareprozess durchgeführt werden. Denn für jeden einzelnen Software- prozess sind die Qualitätsziele jeweils neu zu bestimmen. Daher steht beim Software- qualitätsmanagement nicht die Gestaltung des Qualitätsmanagements einer ganzen Organisation im Fokus, sondern gezielt die Aktivitäten und Verantwortlichkeiten inner- halb eines ganz konkreten Softwareprozesses.

Die Qualitätsplanung legt für einen konkreten Softwareprozess genau fest, welche Per- sonen für welche QS-Aktivitäten zuständig sind, zu welchem Zeitpunkt im Softwarepro- zess welche QS-Aktivitäten erforderlich sind und mit welchen Mitteln, Methoden und Werkzeugen die Qualitätssicherung durchgeführt wird.

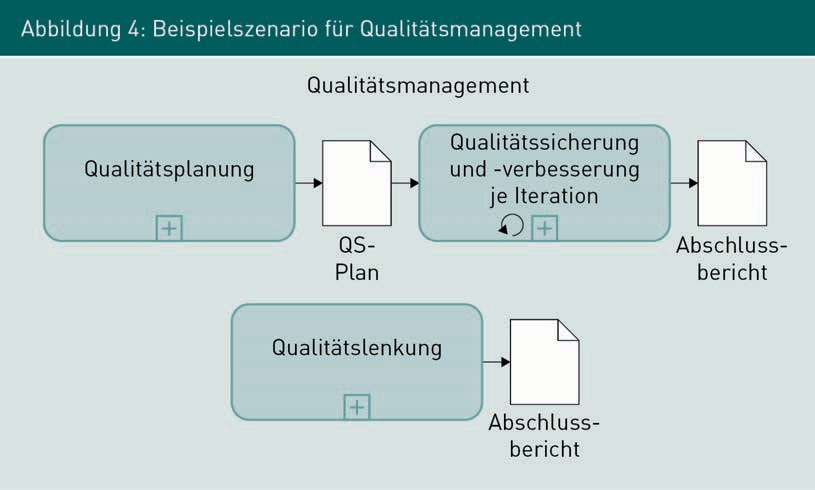
In den Aktivitäten zur Qualitätssicherung werden die im Qualitätsplan festgelegten Maßnahmen durchführt. Der Qualitätsplan gibt somit den konkreten Handlungsrahmen für ein Projekt vor, sodass jeder Mitarbeiter im Projekt jederzeit weiß, wann welche QS- Aktivitäten anstehen, wie die Ergebnisse dokumentiert werden und wer für die Durch- führung der Aktivitäten verantwortlich ist.

Unter der Bezeichnung Qualitätslenkung werden alle Aktivitäten zusammengefasst, die für die Überwachung und Steuerung der QS-Aktivitäten erforderlich sind. Hierbei steht nicht das Softwaresystem im Fokus der Betrachtung, sondern die Aktivitäten, die zur Qualitätssicherung durchgeführt werden. Ziel der Qualitätslenkung ist eine angemes- sene Güte der Qualitätssicherung.

Organisation und Planung von Softwarequalität

In Softwareprojekten werden die Aktivitäten der Qualitätsverbesserung häuﬁg zusam- men mit den Aktivitäten zur Qualitätssicherung durchgeführt. Bezogen auf die typi- schen Ergebnisartefakte im Softwareprozess bedeutet Qualitätsverbesserung das Über- arbeiten von Text, Modellen und Programmcode. Daher werden die Aktivitäten zur Qualitätsverbesserung im weiteren Verlauf des Kurses nicht weiter vertieft.

Abbildung 4 zeigt ein konkretes Beispielszenario für Softwarequalitätsmanagement. Das Ergebnis der Aktivitäten zur Qualitätsplanung ist ein Qualitätssicherungsplan, der kon- krete QS-Maßnahmen enthält. Diese werden im Rahmen der Qualitätssicherung für jedes Softwarerelease durchgeführt und am Ende des Projektes in Form eines Abschlussberichtes dokumentiert. Die Qualitätsverbesserung erfolgt begleitend zur Qualitätssicherung ebenfalls pro Release. Begleitend dazu werden die Aktivitäten zur Qualitätslenkung durchgeführt, die am Ende des Projekts ebenfalls mit einem Abschlussbericht dokumentiert werden.



Wie bereits erwähnt, können nicht nur die in einem Softwareprozess erzeugten Artef- akte qualitätsgesichert werden, sondern auch der Softwareprozess selbst.

###### Produktorientiertes Softwarequalitätsmanagement

Das produktorientierte Qualitätsmanagement überprüft sowohl Softwareprodukte auf festgelegte Qualitätsmerkmale als auch die Zwischenergebnisse, die bei der Erstellung, Wartung und Weiterentwicklung der Software erzeugt werden. Neben den projektspezi- ﬁschen Qualitätsmerkmalen wird für Anwendungssoftware gelegentlich zusätzlich die Erfüllung von Gütebedingungen gefordert, die von externen Zertiﬁzierungsstellen geprüft und bestätigt werden. Beispiele hierfür sind unter anderem die Prüfung von Anwendungen, die im iTunes Appstore durch die Firma Apple veröffentlicht werden, oder der TÜV-AppCheck-Service zur Zertiﬁzierung von mobilen Anwendungen auf Ein-

haltung von Datenschutzkriterien. Maßnahmen für produktorientiertes Softwarequali- tätsmanagement umfassen sowohl konstruktive Maßnahmen, wie das Vorgeben von bestimmten Dokumententemplates oder das Vorgeben von Modellierungskonventio- nen, als auch analytische Maßnahmen, wie das statische Begutachten von Quellcode oder das Ausführen von lauffähigen Komponenten im Rahmen von Softwaretests.

###### Prozessorientiertes Qualitätsmanagement

Das prozessorientierte Qualitätsmanagement dient zur Steuerung der Qualität des Soft- wareprozesses. Hierbei stehen also nicht die erzeugten Artefakte im Fokus der Betrach- tung, sondern die Durchführungsqualität der im Softwareprozess vorgesehenen Aktivi- täten und der Einsatz der vorgesehenen Techniken und Methoden. Das prozessorientierte Qualitätsmanagement erfolgt begleitend zum Softwareprozess und ist charakterisiert durch eine kontinuierliche Bewertung und gegebenenfalls Verbesse- rung der Prozessqualität. Eine wichtige Voraussetzung zu einem effektiven prozessori- entierten Qualitätsmanagement ist das Qualitätsbewusstsein der Teammitglieder. Nur wenn die Qualität der Zusammenarbeit regelmäßig kritisch reﬂektiert und geprüft wird, ist eine kontinuierliche Prozessverbesserung möglich. Beispiele für Maßnahmen zum prozessorientierten Qualitätsmanagement sind der Einsatz eines standardisierten Ent- wicklungsprozesses, der Rollen und Aktivitäten in einem angemessenen Maß vorgibt, der verbindliche Einsatz von Werkzeugen zum Konﬁgurationsmanagement oder die Vor- gabe von Eskalationspfaden im Fall einer Projekthavarie.

### Qualitätsplanung und Qualitätsziele

Mit den Aktivitäten zur Qualitätsplanung werden neben den konkreten Qualitätszielen des zu erstellenden Systems auch alle Vorgaben zur Organisation und Durchführung von Maßnahmen zur Qualitätssicherung festgelegt. Entsprechend der Risikoanalyse und dem verfügbaren Projektbudget, sind die Maßnahmen zur Qualitätssicherung so zu pla- nen, dass sie möglichst kostenoptimal durchgeführt werden können.

Die Qualitätsplanung erfolgt in frühen Phasen des Projekts. Gemeinsam mit dem Auf- traggeber und anderen gegebenenfalls relevanten Stakeholdern werden Qualitätsziele und Qualitätsindikatoren für das zu erstellende System festgelegt. Dabei werden die konkreten Qualitätsziele mit Hilfe eines geeigneten Qualitätsmodells bestimmt und zu jedem Qualitätsziel entsprechende Qualitätsindikatoren abgeleitet.

Beispiel

Für einen Onlineshop wurde die Benutzbarkeit als wichtiges Qualitätsmerkmal festge- legt. Als messbarer Qualitätsindikator wurde die Anzahl der dargestellten Dialogmasken vom Warenkorb bis hin zum Abschluss der Bestellung identiﬁziert. Eine konkrete Quali- tätsanforderung an den Onlineshop wurde dazu wie folgt formuliert: „Die Anzahl der dargestellten Dialogmasken in der GUI für die Bestellung von Warenkorb bis Abschluss der Bestellung darf für Bestandskunden den Wert 2 und bei Neukunden den Wert 4 nicht überschreiten.“

Organisation und Planung von Softwarequalität

###### Planung der organisatorischen Erreichung von Qualität

Neben der Bestimmung von Qualitätszielen für das erzeugte Softwaresystem wird im Zuge der Qualitätsplanung auch die organisatorische Erreichung von Qualität durch deﬁnierte Aktivitäten, Methoden und Techniken im Softwareprozess festgelegt.

Folgende Aspekte müssen durch die Ergebnisse der organisatorischen Qualitätsplanung berücksichtigt werden:

* Wer soll die Qualität sichern? Es muss zu jedem Zeitpunkt im Projekt klar sein, wel- chem Teammitglied welche Rollen in Bezug auf das Qualitätsmanagement und wel- che Aufgaben dem Zuständigkeitsbereich der einzelnen Rollen zugeordnet sind.
* Was soll qualitätsgesichert werden? Es muss zu jedem Ergebnistyp, der im Verlauf des Softwareprozesses erzeugt wird, festgelegt werden, welche Kriterien zu dem erzeugten Artefakt erfüllt sein müssen, damit der geforderte Qualitätsgrad vorliegt. Zu diesem Zweck müssen messbare Qualitätsindikatoren auf Grundlage der als rele- vant festgelegten Qualitätsziele bestimmt und dokumentiert werden.
* Wann soll die Qualität gesichert werden? Die verschiedenen QS-Aktivitäten müssen in einen zeitlichen Bezug zu den konstruktiven Aktivitäten im Softwareprozess gesetzt werden. Neben der gezielten Festlegung von Aktivitäten werden häuﬁg auch sogenannte Quality Gates deﬁniert. Ein Quality Gate ist ein bestimmter Zeitpunkt im Softwareprozess, zu dem deﬁnierte Eigenschaften der vorliegenden Artefakte explizit geprüft werden müssen. Sind eine oder mehrere geforderte Eigenschaften nicht erfüllt, können die Folgeaktivitäten nicht begonnen werden.
* Wie soll die Qualität gesichert werden? Bei der Qualitätsplanung werden konkrete Techniken und Methoden zur Qualitätssicherung bzw. Testfallerstellung bestimmt. Aus dem Portfolio der geeigneten Techniken werden diejenigen ausgewählt, mit denen nach aktuellem Kenntnisstand eine kostenoptimale Systemqualität erzielt werden kann. Im Verlauf des Softwareprozesses müssen die im Qualitätsplan vorge- sehenen Techniken und Methoden unter Umständen der aktuellen Projektsituation angepasst werden.

Als Ergebnis der Planungsaktivitäten gibt es einen projektspeziﬁschen Qualitätssiche- rungsplan (auch: Qualitätsmanagementplan), in dem alle Zuständigkeiten, Ergebnisse, Aktivitäten und Methoden dokumentiert sind. Abbildung 5 zeigt typische Aktivitäten, die im Rahmen einer umfassenden Qualitätsplanung für ein Softwareprojekt durchgeführt werden. Die konkreten Planungsaktivitäten sind immer abhängig von Art und Größe des Projektes sowie der konkreten Projektorganisation. Sie werden in der Regel nicht streng sequenziell wie in Abbildung 5 veranschaulicht abgearbeitet. Diese Darstellung dient vielmehr als Übersicht und Checkliste, ob bei der Qualitätsplanung alle wichtigen Punkte berücksichtigt wurden.



Alle in Abbildung 5 dargestellten Aktivitäten werden in Tabelle 1 beschrieben.

|  |  |
| --- | --- |
| Tabelle 1: Aktivitäten zur Qualitätsplanung | |
| Aktivität | Beschreibung |
| Zuständigkeiten festle- gen und Rollen beset- zen | Nennung und Beschreibung aller Rollen, die bei QS- Aktivitäten aktiv werden, sowie Zuordnung der Rollen zu konkreten Personen; ggf. Darstellung der Aufbau- organisation des Projekts. |
| Gremien und Regelmee- tings festlegen | Gremien benennen und deren Zuständigkeit beschrei- ben; regelmäßige Meetings dieser Gremien einrichten. |
| Qualitätsziele festlegen und priorisieren | Auf Basis eines Qualitätsmodells (z. B. ISO 9126) für das Projekt relevante Qualitätsziele bestimmen und anschließend priorisieren. Dieses Ergebnis dient für alle weiteren Aktivitäten als eine der wichtigsten Richt- linien. Eine Abstimmung mit dem Auftraggeber ist hier- bei in der Regel erforderlich. |

Organisation und Planung von Softwarequalität

|  |  |
| --- | --- |
| Aktivität | Beschreibung |
| QS-Aktivitäten und Qua- lity Gates im Software- prozess festlegen | Konkrete QS-Aktivitäten wie Reviews oder Software- tests werden festgelegt und in Bezug zu den konstruk- tiven Aktivitäten im SW-Prozess verortet. Darüber hinaus werden mit Quality Gates ganz wichtige Ent- scheidungspunkte in den Softwareprozess eingebaut, an denen auf Grundlage der gemessenen Qualität über den weiteren Verlauf des Projekts entschieden wird. |
| Abnahmekriterien und - gremien wichtiger Ergebnistypen festlegen | Um sicherzustellen, dass alle im Projekt erzeugten Artefakte im angestrebten Qualitätsgrad vorliegen, wer- den für wichtige Artefakte (wie fachliche Anforderun- gen, Speziﬁkation, Quellcode, Testfälle, Planungsdoku- mente, etc.) Abnahmekriterien festgelegt und es wird bestimmt, welcher Personenkreis bei der Vorbereitung der Abnahme und der ofﬁziellen Abnahme beteiligt ist. |
| Konstruktive QS-Maß- nahmen festlegen | Zu den konstruktiven Maßnahmen zählen alle Vorga- ben und Richtlinien, die zu den Aktivitäten im SW-Pro- zess festgelegt werden, um die Entstehung von Fehlern zu verhindern. Beispiele hierfür sind Checklisten, Tem- plates, Programmierrichtlinien, Vorgabe bestimmter Werkzeuge, Methoden, Techniken und Verfahren. |
| Analytische QS-Maßnah- men festlegen: statische Verfahren | Beschreibung der Abläufe und Ergebnisse von stati- schen Testverfahren wie Reviews oder statische Code- analyse. Da grundsätzlich alle erzeugten Artefakte sta- tisch geprüft werden können, ist ggf. eine Unterscheidung nach Artefakttyp erforderlich. |
| Analytische QS-Maßnah- men festlegen: dynami- sche Verfahren | Beschreibung der Abläufe und Ergebnisse von dynami- schen Testverfahren, typischerweise jeweils für jede relevante Teststufe. Hierbei wird auch festgelegt, wie Testfälle und Testdaten zu erzeugen sind. |

|  |  |
| --- | --- |
| Aktivität | Beschreibung |
| Testinfrastruktur festle- gen | Da industrielle Informationssysteme in der Regel in einer komplexen Anwendungslandschaft betrieben werden, über technische Schnittstellen mit anderen Systemen verbunden sind und häuﬁg viele hundert- tausende Datensätze verarbeiten und speichern, muss das System je nach Teststufe in einer möglichst pro- duktionsnahen Umgebung laufen. Daher ist die Bereit- stellung von Testinfrastrukturen häuﬁg sehr aufwendig, jedoch für Integrationstests und Systemtests unver- zichtbar. |
| Build-Prozesse festlegen | Die von den einzelnen Entwicklern erstellten Code- fragmente werden durch einen sogenannten Build-Pro- zess zu einem lauffähigen System zusammengestellt.  Bei der Zusammenstellung können automatische Soft- waretests durchgeführt werden. Wann ein Build-Pro- zess angestoßen wird, welche Art von Tests dabei durchgeführt werden und wie im Fehlerfall zu reagie- ren ist, wird im Qualitätsplan festgelegt. |

###### Qualitätsziele

Mit dem Festlegen und Priorisieren von Qualitätszielen wird projektweit eine Richtlinie erstellt, an der sich alle QS-Aktivitäten orientieren müssen. Im Grunde verhalten sich die Qualitätsziele zu den einzelnen QS-Aktivitäten, wie die Projektvision zu den kon- struktiven Aktivitäten: Qualitätsziele bilden den Orientierungspunkt, an dem sich die einzelnen QS-Aktivitäten orientieren müssen. Um den abstrakten Begriff Qualität greif- bar zu machen, können Qualitätsmodelle helfen. Für die Konkretisierung von Software- qualität werden insbesondere Softwarequalitätsmodelle eingesetzt. Die Grundidee eines Qualitätsmodells ist die Zergliederung eines abstrakten Oberbegriffs „Qualität“ in verschiedene Unterbegriffe. Diese Unterbegriffe werden wiederum so lange zergliedert, bis eine konkrete messbare Qualitätseigenschaft formuliert werden kann. Ein Qualitäts- modell besteht daher aus einer Hierarchie von Begriffen, die bei der Speziﬁkation kon- kreter Qualitätseigenschaften helfen können. Abbildung 6 zeigt als Beispiel ein Quali- tätsmodell nach ISO 9126. Der Begriff Softwarequalität wird in sechs Teilmerkmale aufgeteilt, die selbst wiederum in Untermerkmale aufgeteilt sind.

Organisation und Planung von Softwarequalität



Als Hilfsmittel zur Überprüfung, ob Qualitätseigenschaften geeignet beschrieben sind, kann das SMART-Modell aus dem Bereich des Projektmanagements bzw. aus der Perso- nalentwicklung verwendet werden. Beides sind Bereiche, in denen Ziele vereinbart und deren Erreichung geprüft werden müssen. Daher sollten auch die formulierten Quali- tätsziele SMART beschrieben werden. SMART ist ein Akronym, das nach Schiersmann (2000, S. 169ff.) wie folgt aufgelöst werden kann:

* S= speziﬁsch: Qualitätsziele sollen deutlich beschrieben und deﬁniert sein.
* M= messbar (qualitativ, quantitativ): Die Umsetzung der Qualitätsziele muss geprüft werden können.
* A= akzeptabel: Die Qualitätsziele müssen von den Stakeholdern des Projekts akzep- tiert werden; dazu zählt auch die Akzeptanz der Prüfmethode.
* R= realistisch: Qualitätsziele müssen unter Berücksichtigung der fachlichen und organisatorischen Rahmenbedingungen erreicht werden können.
* T= terminiert: Qualitätsziele müssen innerhalb eines vorgesehenen Zeitrahmens,

z. B. in einem bestimmten Release, umgesetzt werden.

In der Praxis werden die konkreten Qualitätsziele für ein Projekt aus verschiedenen Quellen abgeleitet. So haben große Unternehmen häuﬁg ein eigenes Qualitätsmodell, das sich an den speziﬁschen Bedürfnissen der Organisation orientiert. Weiterhin gibt es auch organisationsspeziﬁsche QM-Handbücher oder eine kommunizierte IT-Strate-

gie, die als Leitplanken für die Qualitätszielbestimmung genutzt werden können. Darü- ber hinaus sind die konkreten Projektziele bzw. die Projektvision eine Quelle für Quali- tätsziele, genauso wie vertragliche Vorgaben des Auftraggebers. Die persönlichen Erfahrungen von Stakeholdern mit vergleichbaren Projekten oder Systemen, insbeson- dere die des Qualitätsmanagers, sind ebenfalls zu berücksichtigende Quellen für Quali- tätsziele.

In der Regel werden in frühen Projektphasen sehr viele relevante Qualitätsziele identi- ﬁziert. Daher müssen diese priorisiert werden, damit im weiteren Projektverlauf die ein- zelnen QS-Aktivitäten diese Priorität berücksichtigen können. Außerdem ist darauf zu achten, dass die Menge der ermittelten Qualitätsziele ebenfalls qualitätsgesichert wer- den muss. Werden wichtige Ziele nicht erkannt und damit auch nicht dokumentiert, können sie auch nicht erreicht werden. Im Gegenteil: überﬂüssige oder falsch formu- lierte Qualitätsziele bergen das Risiko, dass Aufwände für die Erreichung dieser Ziele entstehen, die keinen Nutzen für die Kunden des Systems bringen.

### Qualitätssicherung und Qualitätsverbesserung

Die einzelnen Aktivitäten für die Qualitätssicherung und -verbesserung werden in der Regel zwar nach organisationsspeziﬁschen Vorgaben, jedoch für jedes einzelne Projekt konkret durch die Projektleitung zusammen mit dem Qualitätsmanager festgelegt. Am Beispiel des in Abbildung 7 dargestellten Qualitätsplans werden im Folgenden typische Beispiele für konkrete Aktivitäten der Qualitätssicherung und -verbesserung in einem Softwareprozess genannt und erläutert. Links in Abbildung 7 sind die schwerpunktmä- ßig konstruktiven Aktivitäten des Softwareprozesses dargestellt, rechts die schwer- punktmäßigen Aktivitäten zur Qualitätssicherung. Mit einem roten Stern (\*) sind die Elemente gekennzeichnet, die explizit zur Qualitätssicherung beitragen. Neben den ver- schiedenen Aktivitäten und deren Ergebnissen enthält Abbildung 7 auch sogenannte Quality Gates. Diese sind zwischen den beiden Pools SW-Entwicklung und Qualitätssi- cherung dargestellt. Überarbeitungszyklen sind dort nicht dargestellt, jedoch im Fall des Nichtbestehens von Tests oder Quality Gates erforderlich.

\*K: Konstruktives Qualitätsmanagement

Zu allen mit \*K gekennzeichneten Elementen – in Abbildung 7 sind das alle Ergebnisty- pen – gibt es festgelegte Templates, die zu verwenden sind. Darüber hinaus gibt es für jedes Dokument eine verantwortliche Rolle und einen deﬁnierten Abnahmeprozess. Zu Aktivitäten, die mit \*K markiert sind, gibt es Vorgaben: Im Requirements Engineering und der Architektur gibt es Templates und Checklisten. Zur Implementierung gibt es vorgegebene Werkzeuge und Programmiersprachen, zur Aktivität Testfallerstellung gibt es Richtlinien zur Testdatenermittlung und zu Aktivitäten der Testphase Richtlinien zum Einsatz von Werkzeugen zur Testautomatisierung und zur Regressionsfähigkeit.

\*1: Prüfen der fachlichen Anforderungen

Die Prüfung der fachlichen Anforderungen stellt sicher, dass die dokumentierten Anfor- derungen hinsichtlich der Kriterien Inhalt, Dokumentation und Abgestimmtheit eine ausreichende Qualität haben.

Organisation und Planung von Softwarequalität

\*2: Quality Gates

Der in Abbildung 7 dargestellte Qualitätsplan sieht insgesamt sechs Quality Gates vor, von denen fünf in jeder Iteration passiert werden müssen. Ziel von Quality Gates ist es, zu einem bestimmten Zeitpunkt im Softwareprozess sicherzustellen, dass die erzeugten Artefakte vorher festgelegte Eigenschaften erfüllen. Häuﬁg werden Quality Gates auch aus Gründen der Transparenz und Nachvollziehbarkeit der Qualitätsentwicklung einge- setzt. Ein Quality Gate kann passiert werden, wenn die vorher, z. B. in Form einer Check- liste, dazu festgelegten Kriterien erfüllt wurden. Neben der Qualität der erzeugten Artef- akte wird mit Quality Gates häuﬁg auch die Einhaltung von Vorgaben und Richtlinien an den Softwareprozess geprüft. In Tabelle 2 werden die im Beispiel dargestellten Quality Gates beschrieben.

|  |  |
| --- | --- |
| Tabelle 2: Beispiele für Quality Gates im Softwareprozess | |
| Quality Gate (QG) | Beschreibung |
| QG 1: Geprüfte Anfor- derungen | Eine technische Speziﬁkation darf erst erfolgen, wenn geprüfte Anforderungen vorliegen. Die ofﬁzielle Freigabe der Anforderungen ist erfolgt. |
| QG 2: Geprüfte Spezi- ﬁkation | Die technische Speziﬁkation muss erst geprüft und freige- geben werden, bevor mit der Erstellung der Architektur- beschreibung und der Testfallerstellung begonnen wer- den darf. |
| QG 3: Geeignete Architektur | Die erstellte Architekturbeschreibung und die dabei getroffenen Entscheidungen wurden auf Eignung zur Erfüllung der Anforderungen positiv bewertet. |
| QG 4: Getestete Kom- ponenten | Die vom Entwicklungsteam erstellten (oder geänderten) Komponenten sind implementiert und auf Ebene der Modul- und Komponententests getestet. |
| QG 5: Getestetes Sys- tem | Der Systemtest ist abgeschlossen und alle für die aktu- elle Iteration relevanten Fehler behoben. |
| QG 6: Abgenommenes System | Das System wurde vom Kunden abgenommen und kann in Betrieb genommen werden. |



Organisation und Planung von Softwarequalität

\*3: Prüfen der technischen Speziﬁkation

Die Prüfung der technischen Speziﬁkation stellt sicher, dass die zu implementierenden Systemkomponenten in einem für die Architektur und Implementierung geeigneten Umfang und Detaillierungsgrad beschrieben sind.

\*4: Bewertung der geplanten Architektur

Die erstellte Architekturbeschreibung und die dazu getroffenen Entscheidungen werden beispielsweise mit der szenariobasierten Architekturanalyse (siehe Lektion 7) geprüft. Dazu wird anhand von konkreten, aus der Speziﬁkation abgeleiteten fachlichen Anwen- dungsszenarien analysiert, ob die konzipierte Architektur die Anforderungen erfüllen kann.

\*5: Testfallerstellung

Bereits parallel zur Architekturerstellung kann auf Grundlage der fachlichen Anforde- rungen und der technischen Speziﬁkation mit der Testfallerstellung für die Integrati- ons- und Systemtests begonnen werden.

\*6: Komponententests

Begleitend und eng verzahnt mit der Implementierungsphase werden bereits Software- tests durchgeführt. Hierbei handelt es sich zum einen um Komponententests, bei denen von einem Entwickler erstellte Komponenten und Funktionen isoliert von ande- ren Komponenten getestet werden. Zudem werden auch bereits solche Tests durchge- führt, bei denen die Zusammenschaltung aller in der aktuellen Iteration erstellten Komponenten überprüft werden.

\*7: Integrationstest

Nach Abschluss der Implementierung müssen neu erstellte und geänderte Komponen- ten in das Gesamtsystem integriert werden. Der Integrationstest ist ein fachlicher Ende- zu-Ende-Test, bei dem das Zusammenspiel aller im System beteiligten Komponenten geprüft wird. Dabei wird mit automatischen GUI-Regressionstests sichergestellt, dass bereits erfolgreiche Tests aus früheren Iterationen ebenfalls wieder erfolgreich durch- laufen werden.

\*8: Systemtest

Der Systemtest wird in einer Umgebung durchgeführt, die möglichst der späteren Betriebsumgebung entspricht oder ihr zumindest sehr nahe kommt. Der Systemtest prüft insbesondere das Systemverhalten in Extremsituation, wie beispielsweise den gleichzeitigen Zugriff vieler Nutzer, das Verhalten bei Verarbeitung großer Datenmen- gen, die Reaktion beim Ausfall einzelner Komponenten und das Verhalten im Zusam- menspiel mit anderen Produktivsystemen.

\*9: Abnahmetest

Der Abnahmetest wird vom Auftraggeber durchgeführt und entspricht in der Regel einer Teilmenge der in der Testphase durch den Softwarehersteller durchgeführten Sys- temtests. Auf Grundlage des Ergebnisses des Abnahmetests entscheidet der Auftragge- ber, ob das System den Anforderungen entspricht oder ob Nacharbeiten erforderlich sind.

### Qualitätslenkung

Die Aktivitäten zu Qualitätslenkung (auch: Qualitätssteuerung, Qualitätsregelung) die- nen der Verbesserung der Qualität der QS-Aktivitäten. Die Notwendigkeit von Qualitäts- lenkung ergibt sich aus dem Zusammenhang zwischen der Durchführungsqualität der QS-Aktivitäten und der Produktqualität. Unter der Annahme, dass der Qualitätsgrad von Softwaresystemen von der Qualität der Qualitätssicherung abhängt, müssen beim industriellen Software Engineering die QS-Aktivitäten aktiv gesteuert werden. Steuern meint in diesem Zusammenhang die kontinuierliche Beobachtung der Prozess- und Produktqualität, die Reaktion auf Änderungen des Qualitätsgrades in Form von Anpas- sungen der QS-Vorgaben sowie die Prüfung der tatsächlichen QS-Qualität. Die Erkennt- nisse aus den Aktivitäten zur Qualitätslenkung beeinﬂussen die Vorgaben und Richtli- nien der Qualitätsplanung, die wiederum konkrete QS-Aktivitäten, -Techniken und - Methoden für den Softwareprozess festlegt.

Exemplarisch für eine grundsätzliche Herangehensweise an die Qualitätslenkung wird im Folgenden das Konzept des Total Quality Managements (TQM) vorgestellt (siehe Balzert 1998, S. 340–355).

###### Total Quality Management (TQM)

Im Vergleich zu klassischen Ansätzen des Qualitätsmanagements, bei dem die Verant- wortung für Qualitätssteuerung ganz bestimmten Rollen zugeordnet wurde, bezieht das TQM explizit alle Mitarbeiter einer Organisation bei der Qualitätslenkung mit ein. Die Erreichung einer hohen Produktqualität ist beim TQM das höchste Ziel aller Mitarbeiter einer Organisation. Allein der Kunde entscheidet über den erreichten Qualitätsgrad. Dabei basiert TQM auf der Annahme, dass eine vollständige und kontinuierliche Kon- zentration auf eine hohe Qualität einen langfristigen Geschäftserfolg durch die Zufrie- denheit der Kunden sicherstellt.

Zum Total Quality Management gibt es keinen Standardprozess und auch keine Stan- dardaktivitäten. Es handelt sich um ein Managementprinzip zur Gestaltung von Organi- sationen und Abläufen. Zum TQM lassen sich folgende Prinzipien beschreiben, die sich als Grundlage zur Gestaltung von eigenen Prozessen und Richtlinien eignen:

* + Prinzip der Kundenorientierung
  + Prinzip des Primats der Qualität
  + Prinzip der Zuständigkeit aller Mitarbeiter
  + Prinzip der ständigen Verbesserung
  + Prinzip des internen Kunden-Lieferanten-Verhältnisses
  + Prinzip der Prozessorientierung

Organisation und Planung von Softwarequalität

Prinzip der Kundenorientierung

Zufriedene Kunden sind das oberste Ziel, denn die Wahrnehmung des Kunden ent- scheidet über die Qualität des Systems. Daher müssen dem Entwicklungsteam die tat- sächlichen Anforderungen des Kunden und der Nutzen des Systems für den Kunden bekannt sein. Das erfordert in der Regel eine Unterstützung des Kunden bei der Formu- lierung der tatsächlichen Anforderungen.

Prinzip des Primats der Qualität

Alle Prozesse werden genauso durchgeführt, wie sie deﬁniert sind. Jede Person, die an der Ausführung des Prozesses beteiligt ist, erledigt ihre Aufgaben von Beginn an richtig

– auch bei jeder Wiederholung. Nacharbeiten und Verschwendung von Ressourcen sol- len vermieden werden. Werden Fehler identiﬁziert, muss deren Ursache ermittelt und beseitigt werden. Qualitätsverbesserungen werden durch die Optimierung des festge- legten Softwareprozesses erreicht.

Prinzip der Zuständigkeit aller Mitarbeiter

Jeder am Projekt beteiligte Mitarbeiter, ist für die Erreichung eines angemessenen Qua- litätsgrades mitverantwortlich. Die Führungskräfte müssen den Mitarbeitern ein mög- lichst fehlerfreies Arbeiten ermöglichen und alle Prozesse unter dem Gesichtspunkt der Prozess- und Produktqualität managen. Dafür kann die Verantwortung nicht beispiels- weise an eine QS-Abteilung delegiert werden.

Prinzip der ständigen Verbesserung

Eine Organisation im Allgemeinen und damit auch Softwareprozesse laufen niemals perfekt. Das Prinzip der ständigen Verbesserung bedeutet, dass durch kleine aber kon- tinuierliche Schritte die Prozessqualität stetig verbessert werden soll. Dazu können und müssen prozessbegleitend Schwachstellen und Verbesserungsvorschläge von allen Mit- arbeitern identiﬁziert und kommuniziert werden.

Prinzip des internen Kunden-Lieferanten-Verhältnisses

Die interne Leistungserbringung wird vergleichbar organisiert wie ein externes Kunden- Lieferanten-Verhältnis. Erbrachte Leistungen werden abgenommen und übergeben, wie es für einen externen Kunden der Fall wäre. Jedes Team ist für die Qualität seiner Leis- tungen selbst verantwortlich. Die Leistung eines Teams wird anhand der Zufriedenheit der internen Kunden gemessen.

Prinzip der Prozessorientierung

Qualitätsmängel und Fehler werden in erster Linie als Schwachstellen im Entwicklungs- prozess gesehen, die ermittelt und behoben werden sollen. Die Überprüfung des Pro- duktes dient der Überprüfung der Prozessqualität.

Zusammenfassung

Grundsätzlich kann das produktorientierte Qualitätsmanagement vom prozessori- entierten Qualitätsmanagement unterschieden werden. Das produktorientierte Qualitätsmanagement überprüft sowohl Softwareprodukte auf festgelegte Quali- tätsmerkmale als auch die Zwischenergebnisse, die bei der Erstellung, Wartung und Weiterentwicklung der Software erzeugt werden. Das prozessorientierte Qualitäts- management dient hingegen der Steuerung der Qualität des Softwareprozesses.

Mit den Aktivitäten zur Qualitätsplanung werden mit Hilfe von Qualitätsmodellen die konkreten Qualitätsziele des zu erstellenden Systems sowie alle Vorgaben zur Organisation und Durchführung von Maßnahmen zur Qualitätssicherung festgelegt. Ziel dabei ist die Erreichung einer möglichst kostenoptimalen Qualität. Die konkre- ten Aktivitäten für die Qualitätssicherung und -verbesserung werden in der Regel für jedes einzelne Projekt durch die Projektleitung und den Qualitätsmanager fest- gelegt.

Im Rahmen der Qualitätslenkung wird die Prozess- und Produktqualität kontinuier- lich beobachtet und auf Änderungen des Qualitätsgrades in Form von Anpassungen der QS-Vorgaben reagiert. Das Konzept des Total Quality Managements (TQM) stellt dabei exemplarisch eine grundsätzliche Herangehensweise an die Qualitätslenkung dar.



# Lektion 3

## Konstruktives Qualitätsmanagement

#### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie wissen, …

… welche Aktivitäten und Maßnahmen dem konstruktiven Qualitätsmanagement zugeordnet werden können.

… wie die 5-Why-Methode zur Root Cause-Analyse eingesetzt wird.

… was Timeboxing bedeutet und wie Checklisten im konstruktiven Qualitätsmanagement eingesetzt werden können.

DL-D-IQSS01-L03

1. Konstruktives Qualitätsmanagement

### Einführung

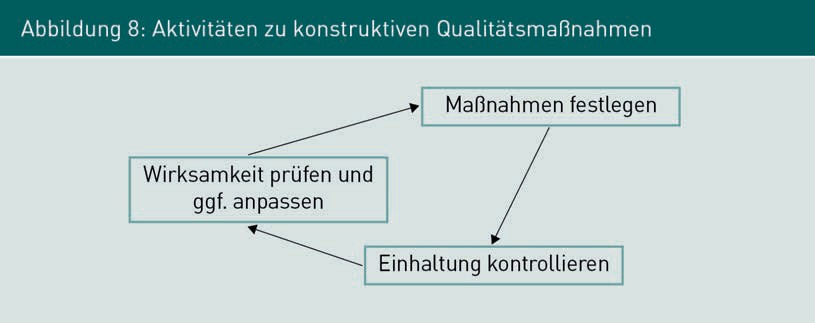
Maßnahmen zum konstruktiven Qualitätsmanagement werden ergriffen um eine hohe Prozessqualität sicherzustellen und Fehler zu vermeiden, bevor sie überhaupt erst ent- stehen. Dabei werden Vorgaben festgelegt, die bei der Bearbeitung konkreter Aktivitä- ten im Softwareprozess zu berücksichtigen sind. Die Grundidee des konstruktiven Qua- litätsmanagements ist, dass die Qualität des Produktes von der Qualität des Erstellungsprozesses abhängig ist.

### Überblick über konstruktive Qualitätssicherung

Die Maßnahmen des konstruktiven Qualitätsmanagements dienen der Sicherstellung der Qualität des Konstruktionsprozesses. Durch verbindliche Vorgaben und Richtlinien soll die Entstehung von Fehlern verhindert werden. So zählt beispielsweise auch ein deﬁnierter Softwareprozess bereits als Maßnahme zur konstruktiven Qualitätssiche- rung. Im konstruktiven QM werden im Vergleich zum analytischen QM keine erzeugten Artefakte betrachtet.

###### Aktivitäten im konstruktiven Qualitätsmanagement

Die Festlegung der konstruktiven QS-Maßnahmen sind, wie in Abbildung 5 gezeigt, Teil der Qualitätsplanung. Abbildung 8 stellt die Aktivitäten zu konstruktiven Qualitätsmaß- nahmen im Überblick dar. Aus der Menge der zur Verfügung stehenden Maßnahmen werden diejenigen ausgewählt, die für ein konkretes Projekt zur Erreichung der kosten- optimalen Qualität als hilfreich erachtet werden. Dabei sollte beachtet werden, dass mit den Vorgaben an das Projekt nicht nur die Fehlersymptome beseitigt werden, son- dern die tatsächliche Fehlerursache. Eine oft geeignete Technik dazu ist die sogenannte Urgrundanalyse (engl.: root cause analysis).

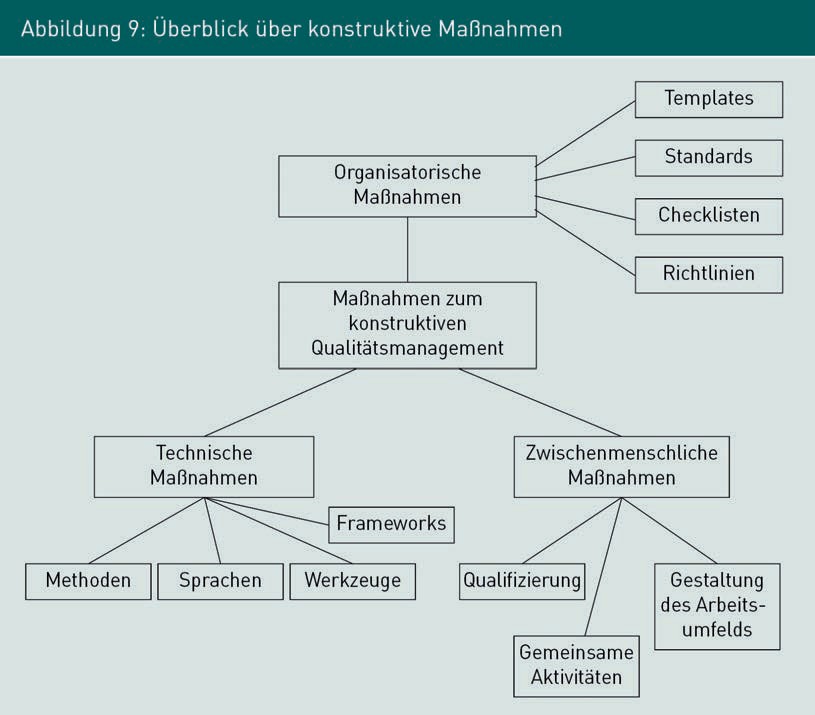


Konstruktives Qualitätsmanagement

Neben der Festlegung der Maßnahmen muss später im Verlauf des Softwareprozesses deren Einhaltung kontrolliert und gegebenenfalls explizit eingefordert werden. Die Ein- haltung der Vorgaben wird entweder in den Prüfungsaktivitäten zu jedem Artefakt geprüft oder bei der Entscheidung, ob ein Quality Gate passiert werden kann. Darüber hinaus muss die Wirksamkeit und die Qualität der Vorgaben regelmäßig geprüft und die Vorgaben gegebenenfalls angepasst werden.

###### Maßnahmen zum konstruktiven Qualitätsmanagement

Maßnahmen zum konstruktiven Qualitätsmanagement im Software Engineering zielen auf die Fehlervermeidung bei Aktivitäten zur Konstruktion, zum Betrieb und zur Weiter- entwicklung von Softwaresystemen ab. Abbildung 9 gibt einen Überblick über Maßnah- men zum konstruktiven QM, die sich in organisatorische, technische und zwischen- menschliche Maßnahmen kategorisieren lassen.



Organisatorische Maßnahmen

Unter den organisatorischen Maßnahmen werden alle nichttechnischen Aspekte zusammengefasst, die Auswirkungen auf die konkrete Arbeitsorganisation haben. Sie umfassen die Vorgabe von einzuhaltenden Standards, Richtlinien und Checklisten. Standards können zum Beispiel Industrienormen (wie ISO 9000, ISO 9126) sein, weit

Organisatorische Maßnahmen

Das sind Maßnah- men, die Auswirkun- gen auf die konkrete Arbeitsorganisation haben und alle nichttechnischen Aspekte umfassen (beispielsweise Standards, Templa- tes, Richtlinien und

Checklisten).

verbreitete Vorgehensmodelle (wie RUP, V-Modell XT, Scrum) oder auch organisationsin- terne Standards (wie Auftragsvergabevorschriften). Templates sind in der Regel unter- nehmens- oder projektspeziﬁsche Vorlagen für Dokumentenstrukturen und -inhalte. Vorlagen können sowohl für Ergebnisartefakte (wie Anforderungen, Speziﬁkation, Doku- mentation, Quellcode) als auch für Managementartefakte (wie Statusbericht, Abschlussbericht, Fehlermeldung) erstellt und vorgeschrieben werden. Mit dem Einsatz von Richtlinien werden mögliche Entscheidungsspielräume vorgegeben. Entscheidun- gen, die bei der Durchführung des Projektes getroffen werden, sollen sich an die vorge- gebenen Richtlinien halten. Typische Beispiele für Richtlinien im Software Engineering sind Programmierrichtlinien, Verhaltensrichtlinien und Konventionen. Mit Timeboxing wird eine Richtlinie zur Organisation und Durchführung von Meetings vorgestellt. Checklisten unterstützen die Erreichung eines konstanten Qualitätsgrads von Aktivitä- ten, die nicht sinnvoll durch eine Prozessdeﬁnition beschrieben werden können. Eine Checkliste enthält beispielsweise alle zu erledigenden Aktivitäten für eine Aufgabe und/oder die benötigten Ergebnis- oder Managementartefakte und wird typischerweise zu wiederholten Standardaufgaben erstellt, wie die Einweisung neuer Mitarbeiter, die Durchführung von Dokumentenreviews oder die Prüfung des aktuellen Prozessschrittes an Quality Gates.

Technische Maßnahmen

Unter dem Begriff technische Maßnahmen werden alle Aspekte zusammengefasst, die sich auf konkrete (software-)technische Aspekte im Softwareprozess beziehen, also um die Vorgaben von Methoden, Sprachen, Werkzeugen und Frameworks aus dem indust- riellen Software Engineering.

Als Methoden im Software Engineering werden Techniken und Vorgehensweisen bezeichnet, mit denen konkrete Aufgaben strukturiert und systematisch erledigt wer- den können. Beispiele für Methoden sind das Ranking (für die Priorisierung), die funkti- onale Äquivalenzklassenbildung (für die Testfallerstellung) und vollständige User Sto- ries (für das Requirements Engineering). Das wichtigste Ergebnisartefakt eines Softwareprojekts ist das Softwaresystem, das in einer oder mehreren Programmierspra- chen programmiert wird.

Die Wahl der im Projekt einzusetzenden Sprachen wird daher in der Regel vorgegeben. Neben den Programmiersprachen für die Programmlogik (wie Java, C#, JavaScript oder Python) werden auch die Programmiersprachen für ganz bestimmte Komponenten (wie JavaScript für die GUI, SQL für Datenbanken) festgelegt. Darüber hinaus werden auch Modellierungssprachen vorgegeben (wie UML, E/R-Diagramme, BMPN, ARIS, Datenﬂuss- diagramm). Zur Festlegung der Sprache gehört auch die Bestimmung der konkreten Sprachversion oder möglicher Dialekte oder Besonderheiten, die beim Einsatz zu berücksichtigen sind. Neben den Methoden und Sprachen werden auch konkrete Vor- gaben an die einzusetzenden Werkzeuge festgelegt. Als Werkzeuge werden Softwaresys- teme bezeichnet, mit denen die Aktivitäten im Softwareprozess unterstützt werden. Dazu zählen Entwicklungsumgebungen (wie Netbeans, eclipse, Visual Studio), Testwerk- zeuge (wie JUnit, Selenium) und Systeme zur Unterstützung des Projektmanagements und der Kommunikation (wie Microsoft Project, Microsoft Teams, Github, Gitlab, Bitbu- cket, Jira).

Konstruktives Qualitätsmanagement

Ein großer Teil der Funktionen eines Softwaresystems wird nicht durch das Entwick- lungsteam direkt erstellt, sondern mit dem Einsatz von bereits bestehendem Pro- grammcode, den sogenannten Bibliotheken oder Frameworks, wiederverwendet. Je nach Reifegrad und Verbreitung einer Programmiersprache gibt es für Standardaufga- ben (wie Datenbankanbindungen), aber auch für Spezialfunktionen (wie Visualisierun- gen) häuﬁg ausgereifte Frameworks. Für zentrale Funktionen eines Systems werden die erlaubten Frameworks häuﬁg vorgegeben, damit innerhalb der gesamten Anwendungs- landschaft der Grad der Heterogenität gering gehalten wird.

Zwischenmenschliche Maßnahmen

Als zwischenmenschliche Maßnahmen werden Aktivitäten bezeichnet, die auf eine pro- fessionelle und gute Zusammenarbeit der am Prozess beteiligten Personen abzielt. Typische Vertreter dieser Maßnahmen sind Qualiﬁzierungen, gemeinsame Aktivitäten und die Gestaltung des Arbeitsumfelds. Qualiﬁzierungen umfassen dabei die Entwick- lung persönlicher und fachlicher Kompetenzen durch Trainings oder Coachings. Gemeinsame Aktivitäten sind Aktivitäten, die das Projektteam gemeinsam durchführt, die jedoch nicht direkt mit dem Projektziel zusammenhängen. Beispiele hierfür sind das gemeinsame Durchführen von fachlichen Trainings, regelmäßige gemeinsame Frei- zeitaktivitäten (wie Sport oder soziales Engagement) oder Teamevents (wie Grillabend, Outdoor-Event). Die Gestaltung des Arbeitsumfelds ist ebenfalls eine konstruktive Maß- nahme. So kann beispielsweise durch die Schaffung von ausgewiesenen Ruhezonen ein konzentriertes Arbeiten ermöglicht, durch das Zusammensetzen von Teammitgliedern in Projektbüros die Kommunikation im Team gefördert und durch den Einsatz von Kan- ban-Boards das selbstorganisierte Arbeiten unterstützt werden.

###### Beispiele für konstruktives Qualitätsmanagement

Folgende Beispiele veranschaulichen die vielfältigen Möglichkeiten konkreter konstruk- tiver QS-Maßnahmen:

* Einsatz eines Vorgehensmodells, das in jeder Phase explizit eine Risikoabschätzung für den Entwicklungsprozess vorsieht;
* Verwendung eines Werkzeugs zur Konﬁgurationsverwaltung von Quell- und Objektc- ode;
* Dokumentenmuster für einen Qualitätsmanagementplan mit vorgegebenem Glieder- ungsschema und einer Anleitung für das Ausfüllen, das sicherstellt, dass alle wichti- gen Punkte behandelt werden;
* Aufstellen von Programmierrichtlinien, die unter anderem die sprechende Benen- nung von Variablen fordern und Unit-Tests für jede fachliche Funktion im Pro- grammcode verpﬂichtend machen.

Zwischenmenschli- che Maßnahmen Zielen auf eine gute Zusammenarbeit der am Prozess beteilig- ten Personen ab.

Beispiele hierfür sind gemeinsame Aktivitäten oder ein angemessen gestal- tetes Arbeitsumfeld.

### Ausgewählte Techniken

###### Root-Cause-Analyse mit 5-Why-Methode

Im Rahmen einer Root-Cause-Analyse (deutsch sinngemäß: Analyse der ursprünglichen Ursache) wird versucht, die eigentliche Fehlerursache von Fehlersymptomen zu identiﬁ- zieren. Als pragmatische Technik zur Fehleranalyse wird in Kleiner (2013, S. 229) die in Ohno (2006) vorgestellte „5-Why-Methode“ empfohlen. Die Kernidee dieser Analyse- technik kann im folgenden Satz auf den Punkt gebracht werden: „Ask ‘why’ ﬁve times about every matter“. Zu einem Fehler soll fünfmal hintereinander die Frage „Warum“ gestellt werden, dann hat man sehr wahrscheinlich die ursprüngliche Ursache gefun- den.

Folgendes Beispiel illustriert die Anwendung anhand eines typischen Fehlverhaltens:

„Zur Komponente Warenkorb werden überdurchschnittlich viele Fehler gemeldet.“

1. Warum-Frage: Warum werden Fehler gemeldet? Antwort: Weil die Berechnung Gesamtsumme häuﬁg etwas von der Summe der Einzelpositionen abweicht.
2. Warum-Frage: Warum weicht die Gesamtsumme ab? Antwort: Weil bei der Berech- nung von Rabatten für Premiumkunden Rundungsfehler auftreten.
3. Warum-Frage: Warum treten Rundungsfehler auf? Antwort: Weil bei der internen Berechnung von Zwischenergebnissen diese nicht in der erforderlichen Genauigkeit gespeichert werden.
4. Warum-Frage: Warum kann die Genauigkeit nicht erreicht werden? Antwort: Weil ein Datentyp zur Speicherung gewählt wurde, der alle Nachkommastellen ab der 3. Stelle nicht speichert.
5. Warum-Frage: Warum wurde dieser Datentyp ausgewählt? Antwort: Weil der Entwick- ler immer davon ausgegangen ist, dass zu Beträgen niemals mehr als 2 Nachkom- mastellen gespeichert werden müssen.

Mit fünf aufeinander aufbauenden Warum-Fragen kann in diesem Fall die ursprüngli- che Ursache für den Fehler lokalisiert werden. Zur nachhaltigen Behebung des Fehlers können gezielte Schulungsmaßnahmen durchgeführt werden und/oder eine Checkliste für den Codereview um die Prüfung auf die Gefahr von Rundungsfehlern erweitert wer- den. Bei der Verwendung der 5-Why-Methode werden bei einer Root-Cause-Analyse fol- gende Schritte durchlaufen (nach Kleiner 2013, S. 229):

1. Schritt: Sammlung aller verfügbaren Informationen zum Fehler und dem durch den Fehler ausgelösten fehlerhaften Verhalten.
2. Schritt: Anwenden der 5-Why-Methode.
3. Schritt: Identiﬁkation der Stellen im Programmcode, die verändert werden müssen, um den Fehler zu beseitigen.
4. Schritt: Identiﬁkation möglicher (konstruktiver und analytischer) QS-Maßnahmen, um den Fehler zukünftig zu vermeiden.

Konstruktives Qualitätsmanagement

1. Schritt: Einführung der QS-Maßnahmen.
2. Schritt: Evaluation der QS-Maßnahmen, ob durch sie die Fehler zuverlässig verhin- dert wurden.

###### Timeboxing

Mit Timeboxing wird eine Technik der Projektplanung bezeichnet, die für Aktivitäten und Vorgänge innerhalb eines Projektes einen festen und unveränderlichen Zeitrahmen vorsieht. Diese Technik ist in der agilen Softwareentwicklung weit verbreitet und wird häuﬁg als Hilfsmittel für eine efﬁziente Durchführung von Meetings verwendet. Timebo- xing hilft dabei, langwierige und ausufernde Diskussionen zu verhindern und durch die Vorgabe eines strengen Rahmens zu hoher Besprechungsqualität zu führen. Die Kern- idee von Timeboxing ist eine Zeitzuteilung zu den einzelnen Aktivitäten bzw. Agenda- punkten und die konsequente Überwachung und Einhaltung der vorgesehenen Zeit. Kann eine Aktivität bzw. ein Agendapunkt nicht in der vorgegebenen Zeit beendet wer- den, so wird beim erreichten Stand abgebrochen und mit der nächsten Aktivität bzw. mit dem nächsten Agendapunkt begonnen. Häuﬁg wird für Besprechungen ein fester Zeitrahmen gesetzt, der allen Teilnehmern bekannt gemacht wird. Nach Ablauf der Zeit wird die Besprechung in jedem Fall beendet.

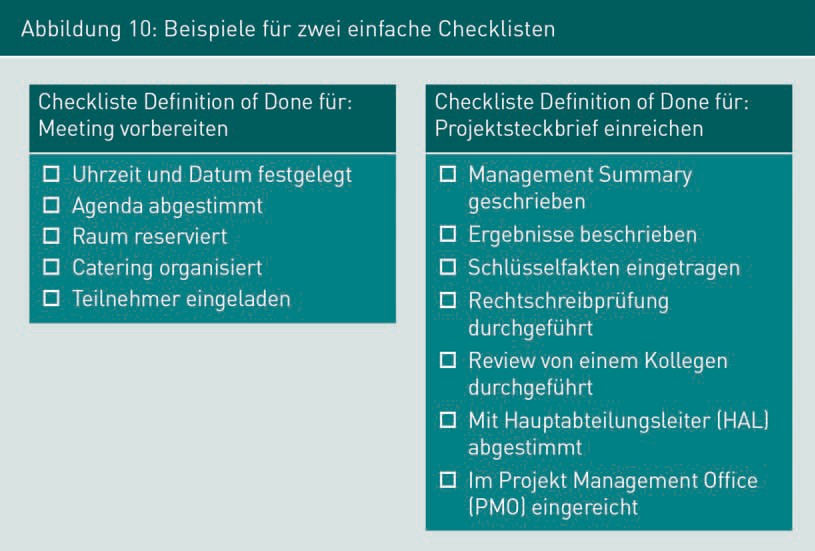
Wird das Timeboxing neu eingeführt, muss ein Moderator bestimmt werden, der die Einhaltung des Zeitplans überwacht. Er muss das Recht haben, einen Agendapunkt bzw. die Besprechung zu beenden. Die Beobachtung in der Praxis zeigt, dass nach einer kur- zen Phase der Eingewöhnung, die durch das Timeboxing eingeführte Zeitschranke von den Beteiligten verinnerlicht wird, sodass der Bedarf eines Moderators mit der Zeit nachlässt. Dazu ist insbesondere in der Einführungsphase die konsequente Einhaltung der Zeitbeschränkung notwendig, um klarzustellen, dass nicht alle geplanten Punkte besprochen werden können bzw. Diskussionen abgebrochen werden können.

###### Checklisten

Checklisten sind in vielen verschiedenen Bereichen des privaten Alltags aber auch bei industriellen Abläufen ein wichtiges Hilfsmittel. Sie helfen sowohl bei der Zusammen- stellung des privaten Reisegepäcks vor langen Reisen oder Bergexpeditionen als auch Flugzeugpiloten bei der Durchführung des Systemchecks vor dem Start des Flugzeugs.

Sie dienen dabei zur Sicherstellung, dass im Rahmen einer Aufgabe keine erforderliche Aktivität vergessen wird und/oder alle benötigten Ergebnis- oder Managementartefakte am Ende der Aufgabe vorliegen. Checklisten werden in der Regel zu sich wiederholen- den Standardaufgaben erstellt, wie die Einweisung neuer Mitarbeiter, die Einrichtung einer Entwicklungsumgebung für ein Projekt oder die Prüfung des aktuellen Prozess- schrittes an Quality Gates.

Abbildung 10 zeigt zwei einfache Checklisten aus der Praxis: links zur Vorbereitung von Besprechungen, rechts für die Einreichung eines Projektsteckbriefes, auf dessen Grund- lage entschieden wird, ob das Projekt genehmigt wird oder nicht.



Wie in Abbildung 10 dargestellt, bestehen Checklisten aus einer einfachen Liste von Einträgen, die nach Prüfung bzw. nach Erledigung einfach abgehakt werden können. Bei der Formulierung von Checklisten sollten folgende Punkte beachtet werden:

* + Ein Punkt auf der Checkliste entspricht genau einem Ziel; zusammengesetzte Ziele sollten vermieden werden und auf mehrere Punkte aufgeteilt werden.
  + Eine klare und einfache Formulierung der Checkpunkte erhöht die Lesbarkeit und die Verständlichkeit. Eine Checkliste sollte nicht größer als ein DIN A4-Blatt sein, sodass sie mit jedem Drucker einfach ausgedruckt werden kann und alle Check- punkte auf einen Blick erfasst werden können.
  + Die zur Erfüllung eines Checkpunktes erforderlichen Kriterien müssen transparent sein, bei Bedarf werden zu einzelnen Checkpunkten Untercheckpunkte oder eigene Checklisten erstellt.

Auch Checklisten müssen einer kontinuierlichen Beobachtung unterliegen. Bei Bedarf müssen sie angepasst werden, um auf geänderte Rahmenbedingungen oder Qualitäts- anforderungen zu reagieren.

Zusammenfassung

Maßnahmen zum konstruktiven Qualitätsmanagement dienen der Sicherstellung der Qualität des Konstruktionsprozesses. Sie lassen sich nach organisatorischen, technischen und zwischenmenschlichen Maßnahmen unterscheiden. In der Regel reicht es nicht, Maßnahmen zum konstruktiven Qualitätsmanagement nur festzule- gen. Im Verlauf des Softwareprozesses muss ihre Einhaltung kontrolliert und expli-

Konstruktives Qualitätsmanagement

zit eingefordert werden. Darüber hinaus müssen die Wirksamkeit und die Qualität der Vorgaben regelmäßig geprüft und die Vorgaben gegebenenfalls angepasst wer- den. Konkrete Beispiele für Maßnahmen im konstruktiven Qualitätsmanagement sind zum einen die Vorgabe der Root-Cause-Analyse, um die Fehlerursachen ausﬁn- dig zu machen. Zum anderen können Timeboxing für die Organisation und Durch- führung von Besprechungen sowie standardisierte Checklisten zur Sicherstellung der Prozessqualität eingesetzt werden.



# Lektion 4

## Statische Qualitätssicherung: Begutachten und Messen

#### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie wissen, …

… wo statische Verfahren zur Qualitätssicherung eingesetzt werden.

… welche Rollen und Aktivitäten es in den verschiedenen Review-Techniken gibt.

… wie Metriken für Softwaresysteme eingesetzt werden und was es für typische Softwaremetriken gibt.

… was die statische Codeanalyse ist und wozu sie eingesetzt wird.

DL-D-IQSS01-L04

1. Statische Qualitätssicherung: Begutachten und Messen

### Einführung

Im Gegensatz zum konstruktiven Qualitätsmanagement, das die Vorgaben zu Aktivitäten im Softwareprozess macht, werden bei Maßnahmen der statischen Qualitätssicherung die im Softwareprozess erzeugten Ergebnisse geprüft und bewertet. Nach einem Über- blick über statische Verfahren zur Qualitätssicherung werden in dieser Lektion sowohl verschiedene Review-Techniken als auch der Einsatz von Metriken vorgestellt und dis- kutiert. Darüber hinaus wird beispielhaft dargestellt, wozu Techniken zur statischen Codeanalyse eingesetzt werden können.

### Einsatz und Überblick über statische Verfahren

Review Eine Review-Technik ist ein statisches manuelles Prüfver- fahren zur Analyse eines Prüﬂings. Der Prüﬂing wird analy- siert, begutachtet und untersucht. Die dabei gewonnenen Informationen wer- den zusammengetra- gen, verdichtet und

ausgewertet.

Statische Codeana-

lyse Diese ist eine werk- zeuggestützte inhalt- liche Analyse und Auswertung von einem Programm- code, jedoch ohne ihn auszuführen.

Statische Verfahren der Qualitätssicherung sind analytische Verfahren, bei denen der Prüﬂing im Rahmen eines Reviews analysiert, begutachtet und untersucht. Die dabei gewonnenen Informationen werden zusammengetragen, gegebenenfalls in Metriken oder Kennzahlen verdichtet und schließlich ausgewertet. Wie bereits in Abbildung 1 dargestellt, wird bei statischen Verfahren im Gegensatz zu dynamischen Verfahren der zu prüfende Programmcode nicht ausgeführt. Alle Verfahren der statischen Qualitätssi- cherungen können grundsätzlich ohne den Einsatz von Software, also nur „mit Papier und Stift“ durchgeführt werden. Jedoch werden insbesondere das Messen und die Berechnung von Metriken sowie die statische Codeanalyse durch Werkzeuge gut unter- stützt. Mit statischen Verfahren können sowohl alle im Softwareprozess erzeugten Artefakte als auch Aktivitäten zur Qualitätssicherung geprüft werden. Darüber hinaus können organisatorische Vorgaben wie das Softwareprozessmodell oder der Qualitäts- plan im Rahmen von Aktivitäten zur Qualitätslenkung geprüft werden.

Abbildung 7 zeigt exemplarisch einen Softwareprozess mit Aktivitäten zum Qualitätsma- nagement. Im Rahmen der folgenden Aktivitäten können statische Verfahren eingesetzt werden:

\*1 = Prüfen der fachlichen Anforderungen: Mittels Review-Techniken werden dokumen- tierte Anforderungen hinsichtlich vorher bestimmter Qualitätskriterien geprüft.

\*2 = Quality Gates: Mithilfe von Checklisten wird die aktuelle Situation im Softwarepro- zess begutachtet und die Einhaltung geforderter Eigenschaften bewertet.

\*3 = Prüfen der technischen Speziﬁkation: Mit Review-Techniken wird die technische Speziﬁkation hinsichtlich vorher bestimmter Qualitätskriterien geprüft. Technische Datenschemas und Datenstrukturen werden bei Bedarf automatisch mit analysiert und bewertet.

Statische Qualitätssicherung: Begutachten und Messen

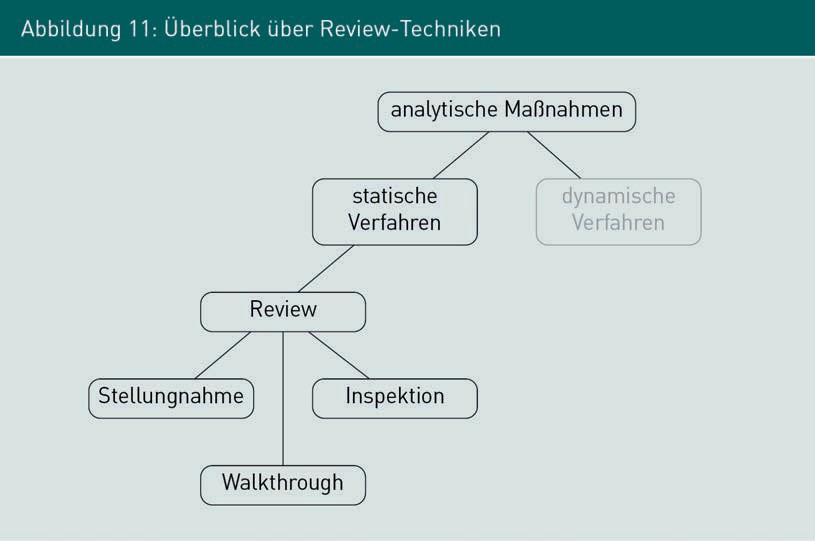
\*4 = Bewertung der geplanten Architektur: Mit Review-Techniken und der szenarioba- sierten Architekturanalyse wird die Architekturbeschreibung auf Eignung zur Erfüllung der Anforderungen geprüft. Detaillierte technische Entwürfe von Komponenten und Datenstrukturen können bei Bedarf automatisch mit analysiert und bewertet werden.

\*5 = Testfallerstellung: Die erstellten Testfälle werden mit Review-Techniken dahinge- hend überprüft, ob sie den für das aktuelle Projekt bestimmten Qualitätsgrad sicher- stellen können.

\*6 = Komponententests: Zwar sind Komponententests keine statischen Verfahren, jedoch werden im Umfeld dieser Tests Maßnahmen zur statischen Codeanalyse durch- geführt. Hierbei wird der erzeugte Programmcode auf die Einhaltung geforderter Eigen- schaften automatisch überprüft und Fehlerwahrscheinlichkeitsanalysen durchgeführt.

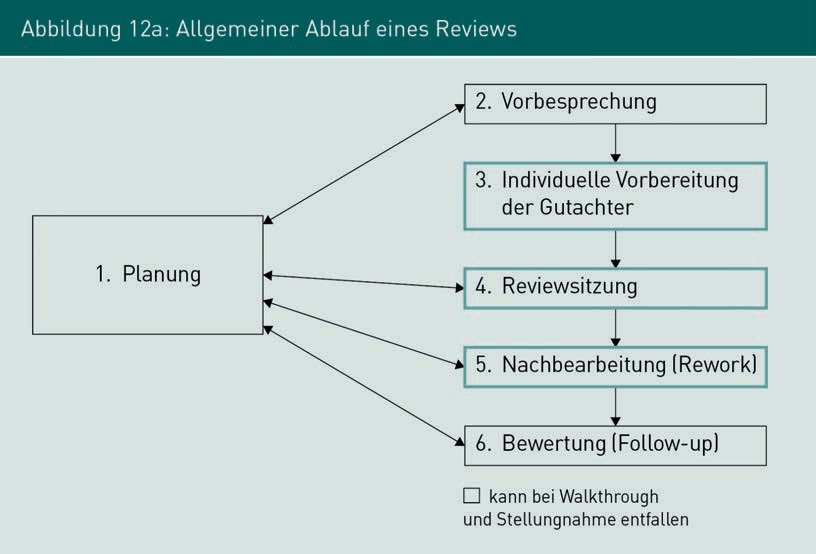
### Begutachten mit Review-Techniken

Eine Review-Technik ist ein statisches manuelles Prüfverfahren zur Analyse eines Prüf- lings. Abbildung 11 zeigt einen Überblick über typische Review-Techniken, die sich hin- sichtlich des Aufwandes und Grades der Strukturierung unterscheiden. So ist eine Stel- lungnahme eine sehr einfache und wenig aufwendige, eine Inspektion hingegen eine sehr stark strukturierte und aufwendige Begutachtungstechnik.



Walkthrough Dies ist ein Review mit mehreren Gut- achten, jedoch ohne die Einhaltung for- maler Kriterien.

In Abbildung 12a ist der allgemeine Ablauf eines Reviews dargestellt. Je nach Art, Ziel und Organisation des Reviews, werden einzelne Phasen weniger intensiv oder gar nicht durchgeführt. Eine Inspektion ist ein sehr formaler Review, der alle in Abbildung 12a gezeigten Aktivitäten umfasst. Bei einem Walkthrough werden häuﬁg nur die in Abbil- dung 12a dick umrandeten Aktivitäten durchgeführt, die jedoch formal organisiert wer- den. Die formale Organisation kann bei einer Stellungnahme entfallen, die somit die einfachste Review-Technik darstellt. Eine ausführliche Beschreibung der einzelnen Akti- vitäten erfolgt im Anschluss an die Beschreibung der Rollen.



###### Rollen im Review

Bei der Durchführung von Reviews sind in Abhängigkeit der gewählten Review-Form die Rollen Moderator, Gutachter, Autor und Protokollführer beteiligt. Deren konkreter Ein- satz variiert dabei wie in Tabelle 3 dargestellt mit der gewählten Variante des Reviews.

Der Moderator ist für die Planung, Vorbereitung und Leitung der Review-Sitzung verant- wortlich. Er wählt, oft gemeinsam mit dem Autor, die weiteren am Review beteiligten Personen aus. Darüber hinaus ist er für die Überwachung der Mängelbehebung verant- wortlich. Die Gutachter sind Sachverständige, die auf Basis ihres Wissens und ihrer Erfahrung in der Lage sind, die fachliche Qualität des Prüﬂings zu bewerten und gege- benenfalls Mängel zu identiﬁzieren. Je nach Komplexität und Kritikalität des Prüﬂings werden auch mehrere Gutachter beim Review miteinbezogen: typischerweise ein bis fünf Gutachter, bei Bedarf auch mehr. Mit jedem zusätzlichen Gutachter erhöhen sich jedoch auch die Aufwände zur Koordination und Abstimmung. Der Autor steht den Gut- achtern und dem Moderator für Fragen zum Prüfobjekt zur Verfügung und sorgt für die

Statische Qualitätssicherung: Begutachten und Messen

Aufklärung bei Unklarheiten. Im Zuge der Nacharbeit ist er für die Beseitigung von Mängeln im Prüﬂing verantwortlich. Der Protokollführer ist für die Anfertigung des Review-Protokolls während der Review-Sitzung zuständig.

|  |  |  |  |
| --- | --- | --- | --- |
| Tabelle 3: An Review-Techniken beteiligte Rollen | | | |
| Name der Rolle | Inspektion | Walkthrough | Stellungname |
| Moderator | X | X  (bei Bedarf vom Autor übernom- men) | - |
| Gutachter | X | X | X |
| Autor (des Prüf- lings) | X | X | X |
| Protokollführer | X | - | - |

###### Aktivitäten beim Review

Im Folgenden werden die in Tabelle 4 dargestellten Aktivitäten eines Reviews detailliert beschrieben.

|  |  |  |  |
| --- | --- | --- | --- |
| Tabelle 4: Review-Aktivitäten | | | |
| Aktivität | Ziele | Ergebnis | Verant- wortlich |
| Planung | * Organisation aller benötigten Ressour- cen (wie Personen, Termine, Räume) * Bestimmung der Prüfziele * Bestimmung der zu bestehenden Krite- rien * Benennung der Gut- achter | * Mit allen Beteilig- ten abgestimmte Terminplanung * Dokumentierte Prüfziele und Bestanden-Krite- rien | Moderator |
| Vorbespre- chung; bei Bedarf | * Alle Beteiligten erhal- ten einen Überblick über den Prüﬂing und die Prüfziele. * Ggf. Aufteilung der Prüfaspekte/Sichten der Prüfung auf Gut- achter * Verteilung aller benö- tigten Unterlagen * Klären von organisa- torischen und fachli- chen Fragen | * Jeder Gutachter kennt den Prüﬂing, das Ziel der Prü- fung und die Prüf- kriterien, für die er verantwortlich ist. | Moderator |
| Individuelle Vorbereitung der Gutach- ter | * Prüﬂing wird von Gut- achter individuell durchgearbeitet und anhand der fest- gelegten Kriterien geprüft. | * Dokumentierte Fragen des Gutach- ters sowie doku- mentierte inhaltli- che Mängel und formale Mängel (wie Schreibfehler) | Gutachter |

Statische Qualitätssicherung: Begutachten und Messen

|  |  |  |  |
| --- | --- | --- | --- |
| Aktivität | Ziele | Ergebnis | Verant- wortlich |
| Review-Sit- zung | * Gemeinsames Durch- gehen des Prüﬂings durch alle Beteiligten * Diskussion und Bewertung der durch Gutachter identiﬁzier- ten fachlichen Mängel | * Protokoll aller identiﬁzierten fachlichen Mängel * Sammlung aller formalen Fehler (werden nicht in Sitzung bespro- chen) * Dokumentiertes Ergebnis (keine Mängel, Nacharbeit nötig, Abbruch wegen schwer- wiegender Mängel) | Moderator |
| Nachberei- tung | * Korrigieren der erkannten Mängel * Bei Inspektion: Män- gelbericht und Erledi- gungsliste der Kor- rekturen erstellen | * Überarbeiteter Prüﬂing * Mängelbericht, Erledigungsliste | Autor |
| Bewertung | * Prüfen, ob alle Kor- rekturen umgesetzt wurden * Abschließende Bewertung des Prüf- lings | * Review-Bericht | Moderator |

###### Gegenüberstellung: Stellungnahme, Walkthrough, Inspektion

Die Review-Techniken Stellungnahme, Walkthrough und Inspektion lassen sich wie folgt beschreiben.

Stellungnahme

Die einfachste und schnellste Technik des Reviews ist eine informelle Stellungnahme. Dabei wird der Prüﬂing durch eine an der Erstellung unbeteiligten und unabhängigen Person gelesen und hinsichtlich der relevanten Prüfkriterien bewertet. Mängel werden

markiert und kurz begründet. Die Stellungnahme kann beispielsweise durch einen Kol- legen oder einen externen Dienstleister durchgeführt werden. Der Ablauf folgt dabei keinem bestimmten Schema oder Vorgaben. Folgendes Szenario skizziert ein Beispiel für eine Stellungnahme: Das Anforderungsdokument wird an einen Testingenieur über- geben. Dieser prüft das Dokument auf Testbarkeit der Anforderungen (Abbildung 12, 3.), markiert die Mängel und gibt dem Requirements Engineer Feedback zu Verbesserungen (Abbildung 12, 4.). Dieser führt anschließend Nachbesserungen durch (Abbildung 12, 5.).

Walkthrough

Im Unterschied zur Stellungnahme wird das Dokument bei einem Walkthrough von mehreren Personen gelesen und die identiﬁzierten Mängel werden in der Gruppe dis- kutiert und bewertet. Ein Walkthrough muss wie eine Stellungnahme zwar keinen for- malen Kriterien entsprechen, der Aufwand für die Organisation und Koordination ist jedoch durch die Beteiligung und Diskussion mehrerer Gutachter größer. Je nach Bedarf und Projektsituation kann die Rolle des Moderators durch den Autor oder durch eine neutrale Person ausgeführt werden. Durch den Einsatz mehrerer Gutachter kann mit dem Walkthrough eine detailliertere Prüfung erfolgen als mit der Stellungnahme. So können beispielsweise durch das perspektivenbasierte Lesen gezielt verschiedene Aspekte beim Walkthrough berücksichtigt werden. Darüber hinaus kann durch die Beteiligung mehrerer Personen als Gutachter ein gemeinsames, teaminternes Verständ- nis von Inhalten des Prüﬂings erlangt und sichergestellt werden. Wenn beispielsweise jeder Entwickler bei der Prüfung der Speziﬁkation als Gutachter mitarbeitet, wird in der gemeinsamen Diskussion während der Review-Sitzung über Inhalte und mögliche Män- gel gemeinsam mit dem Autor ein teamintern einheitlicher Wissensstand erreicht.

Inspektion

Eine Inspektion ist ein formaler Review mit in der Regel mehreren Gutachtern, das nach einem festgelegten Ablauf durchgeführt wird und dessen Durchführung und Ergebnisse ausführlich dokumentiert werden müssen. Die Organisations- und Durchführungskom- plexität ist im Vergleich zur Stellungnahme und zum Walkthrough größer, da bei einer Inspektion die Rollen eines Moderators und Protokollanten explizit gefordert sind. Eine Inspektion wird bei wichtigen Prüﬂingen durchgeführt, in denen nicht identiﬁzierte Mängel ein sehr hohes Schadenspotenzial haben. Typische Szenarien sind Meilensteine sehr großer Projekte und/oder Artefakte in stark verteilten Entwicklungsprojekten an denen mehrere Organisationen beteiligt sind. Durch einen Fokus auf die Dokumenta- tion und den formalen Rahmen werden Inspektionen auch dann eingesetzt, wenn durch Richtlinien oder Gesetze nachweisliche Maßnahmen zur Risikominimierung gefordert werden. Die Gegenüberstellung der Review-Techniken hinsichtlich Charakteri- sierung und Einsatz ist in Tabelle 5 dargestellt.

Statische Qualitätssicherung: Begutachten und Messen

|  |  |  |  |
| --- | --- | --- | --- |
| Tabelle 5: Gegenüberstellung von Review-Techniken | | | |
|  | Stellungnahme | Walkthrough | Inspektion |
| Charakteris- tik | * Schneller, infor- maler Review ohne explizite Organisation | * Informaler Review durch mehrere Gut- achter | * Formaler Review durch mehrere Gutachter nach deﬁniertem Schema |
| Anwen- dungsszena- rien | * Schnelles, infor- males Feedback zum Prüﬂing erhalten * Internes Feed- back unter Kol- legen ohne Pﬂicht zum Nachweis for- maler Review- Kriterien | * Detaillierte Prü- fung wichtiger Artefakte, in denen nicht identiﬁzierte Fehler einen deutlich merk- baren Schaden verursachen können * Schaffung eines einheitlichen Verständnisses von Sachver- halten und För- derung der Teamkommuni- kation | * Prüfung wichti- ger Artefakte, in denen nicht identiﬁzierte Fehler ein sehr hohes Scha- denspotenzial haben * Pﬂicht zum Nachweis von formalen Krite- rien der Prüfung |

###### Feedbacktechnik: Feedback Burger

Häuﬁg ist das Kommunizieren von Feedback zu einem Arbeitsergebnis schwierig, insbe- sondere wenn Fehler oder Mängel im Prüﬂing identiﬁziert wurden. Das Übermitteln von Kritik zusammen mit der Motivation, das Ergebnis anschließend noch zu überarbeiten, stellt oft eine Herausforderung dar. Insbesondere, da beispielsweise im Requirements Engineering die Aufrechterhaltung der Kooperationsbereitschaft der Stakeholder ein wichtiger Erfolgsfaktor ist. Die Feedbacktechnik Feedback Burger unterstützt den Review-Prozess dahingehend, dass sie verschiedene Prinzipien zur Kommunikation von Feedback auf eine anschauliche Art vereinigt.

Folgende Schritte sind zur Vorbereitung der Kommunikation des Feedbacks wichtig:

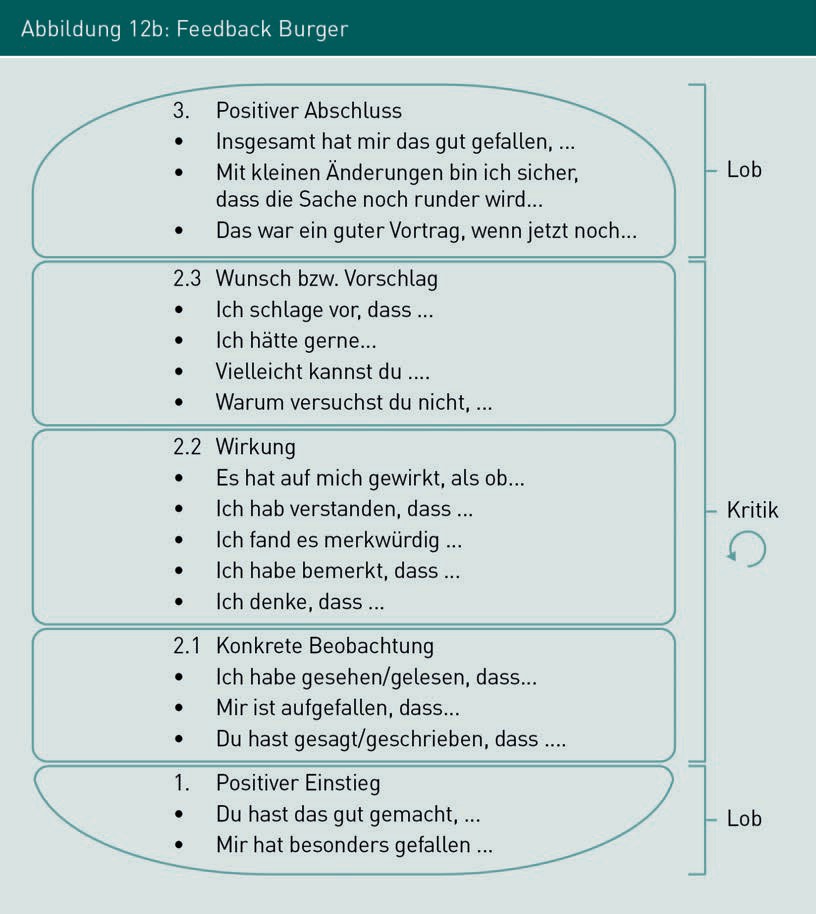
Feedback Burger Dies ist eine Technik zur Kommunikation von Feedback zu Arbeitsergebnissen und dient der Auf- rechterhaltung der Kooperationsbereit- schaft der beteilig- ten Stakeholder.

1. Zuerst möglichst unvoreingenommen Lesen (und/oder Zuhören). Das Ziel dabei ist, ein Verständnis für das Geschriebene/Gesagte zu entwickeln und zu bewerten.
2. Feedback vorbereiten: Inhalte des Prüﬂings werden mit Hilfe eines Bewertungs- schemas bewertet, beispielsweise anhand der vorher festgelegten Prüfkriterien. Alle Aussagen des Feedbacks sollten sich dabei immer auf konkrete Beobachtungen/ Stellen innerhalb des Prüﬂings beziehen. Zur Vorbereitung des Feedbacks hilft der Feedback Burger. Ziel eines Reviews ist es nicht, den Autor persönlich zu kritisieren, sondern zur Verbesserung des Ergebnisses beizutragen.
3. Feedback geben (in der Review-Sitzung): Bei einem Feedback sollte darauf geachtet werden, dass es sich um den ganz persönlichen Eindruck handelt („Ich“-Botschaf- ten) und bei der Formulierung keine Generalisierungen wie „immer“, „nie“ oder

„alles“ benutzt werden.

Abbildung 12b zeigt das Schema eines Feedback Burgers, das zur Vorbereitung und zur Kommunikation des Feedbacks eingesetzt werden kann. Der grundsätzliche Aufbau des Feedback Burgers ist die Kombination aus der Kommunikation von Fehlern und Män- geln sowie positiv wahrgenommenen Eigenschaften. Der Aufbau des Burgers erfolgt dabei von unten nach oben. Das Feedback sollte mit einem positiven Einstieg begon- nen werden, bei dem konkrete positive Details und Eindrücke genannt werden. Mögli- che Kritik oder Mängel werden dann anhand konkreter, individueller Beobachtungen vorgetragen. Das erfolgt mit dem Schema Beobachtung-Wahrnehmung-Wunsch: Zuerst wird die konkrete Beobachtung genannt, dann die individuelle Wirkung auf den Revie- wer und anschließend konkrete Wünsche und Verbesserungsvorschläge. Dieser Zyklus kann dabei auch mehrfach durchlaufen werden. Der Abschluss eines Feedbacks nach dem Feedback Burger ist eine kurze, positive Zusammenfassung, die den Autor moti- viert, die Änderungsvorschläge bei der Nachbesserung zu berücksichtigen.

Statische Qualitätssicherung: Begutachten und Messen



### Messen und Metriken

Eine weitere Technik zur Bestimmung der Qualität von Produkten und Prozessen mit statischen Verfahren ist die Messung. Grundsätzlich können Eigenschaften sowohl von im Softwareprozess erzeugten Artefakten als auch von Eigenschaften des Softwareproz- esses selbst gemessen werden. Die Funktion, die aus den Ergebnissen der Vermessung eines Prüﬂings einen Zahlenwert ermittelt, wird Metrik genannt.

Beispiel für Softwaremetrik

Zu einem Softwaresystem kann die Metrik Lines of Code (LOC) bestimmt werden. Dazu wird die Anzahl aller programmierten Zeilen im Programmcode ermittelt und als Zahl angegeben.

###### Einsatz vom Metriken

Das Ziel, das mit dem Einsatz von Metriken erreicht werden soll, ist die Qualitätskon- trolle und Verbesserung von Ergebnissen und Prozessen. Entsprechend den Leitsätzen

„To measure is to know“ sowie „You can’t control what you can’t measure!“ werden Werte von Kenngrößen mit Hilfe von Werkzeugen bestimmt und analysiert. Auf Grund- lage der gemessenen Werte werden Aussagen über den aktuellen Zustand des Produk- tes oder des Prozesses abgeleitet. Grundsätzlich lassen sich Metriken zur Bewertung von Prozessen von Metriken zur Bewertung von Artefakten unterscheiden. Beispiele für zu messende Aspekte in einem Softwareprozess sind verbrauchte Ressourcen (Zeit, Geld, Personal), Anzahl der identiﬁzierten Fehler im Projekt, Dauer einzelner Aktivitäten, mittlere Behebungszeit von identiﬁzierten Fehlern oder die Anzahl von Änderungen an einer Systemkomponente. Aspekte, die für Ergebnisse gemessen werden, sind bei- spielsweise der Umfang (Anzahl Seiten, LOC, Anzahl Modellelemente), Komplexität, Qua- lität (Anzahl identiﬁzierter fachlicher Fehler, Anzahl identiﬁzierter formaler Fehler), Sta- bilität (Grad der Änderungen in einem bestimmten Zeitraum) oder Stil (Einhaltung von Konventionen, Lesbarkeit, Redundanzfreiheit).

###### Beispiele für Softwaremetriken

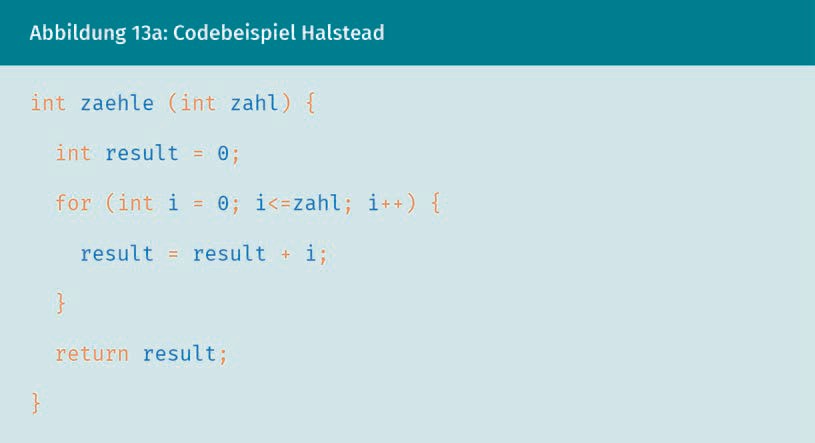
Neben relativ einfachen Metriken wie LOC gibt es auch komplexere Metriken, deren Wert sich aus mehreren Messwerten ermittelt.

Textuelle Komplexität mit Halstead-Metriken

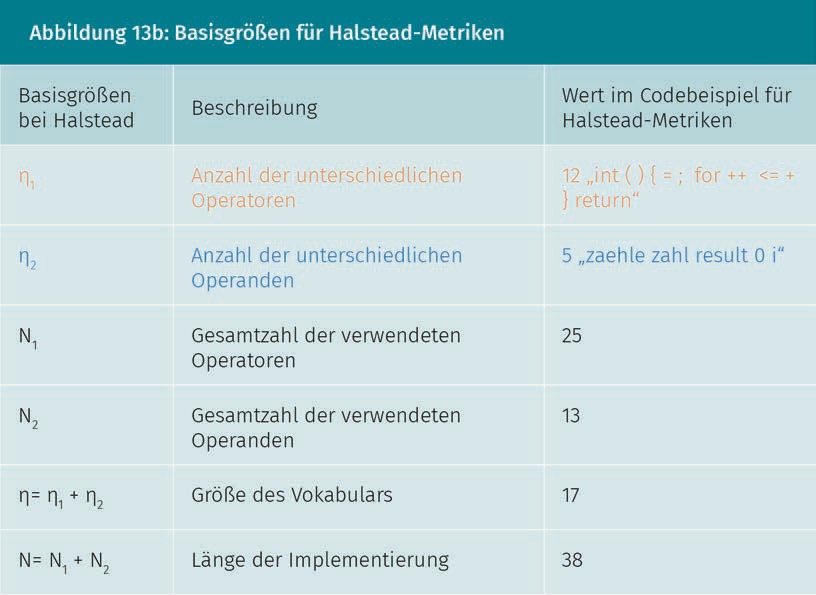
Halstead-Metriken (siehe Liggesmeyer 2009, S. 260-262) wurden zur Bestimmung der Komplexität von Softwarekomponenten entwickelt. Dabei werden zunächst konkrete Basiswerte im Programmcode gemessen, aus denen dann anschließend verschiedene Metriken berechnet werden.

Anhand des folgenden Codebeispiels werden Halstead-Metriken als Beispiele für Soft- waremetriken erläutert:

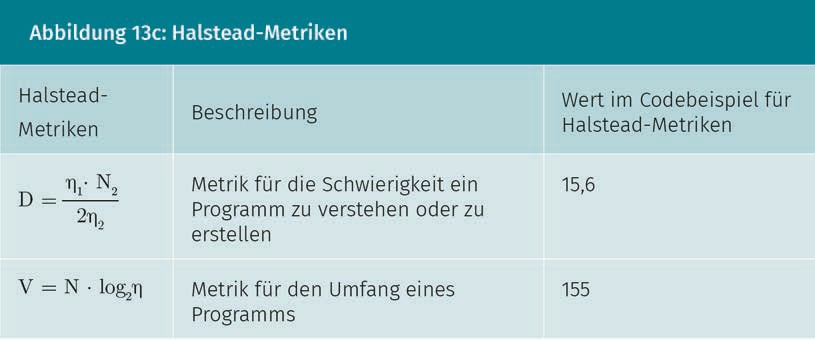
Statische Qualitätssicherung: Begutachten und Messen



Zu diesem Codebeispiel werden in Abbildung 13b die Basisgrößen der Halstead-Metri- ken bestimmt, deren Grundlage die Bestimmung von Art und Anzahl der Operatoren und Operanden im Programmcode ist. Hinweis: Wie konkret gezählt wird, unterscheidet sich je nach Programmiersprache und Implementierung der Metrik.



Auf Grundlage der Basisgrößen können weitere Komplexitätsmetriken bestimmt wer- den. Beispiele für weitere Halstead-Metriken sind in Abbildung 13c angegeben.

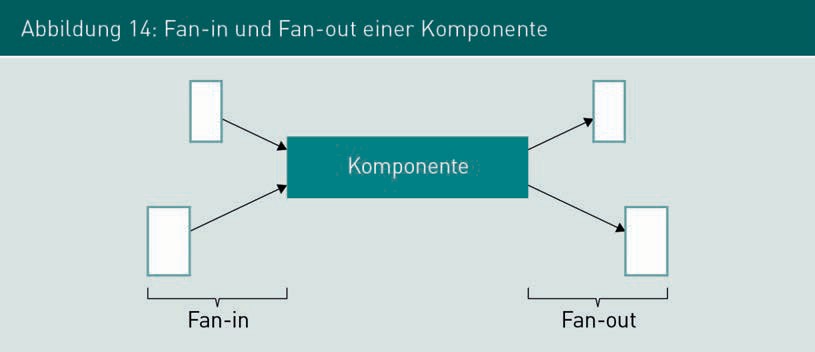


Die Halstead-Metriken wurden bereits 1977 entwickelt. Mittlerweile gibt es sehr viele verschiedene Metriken, die sich jeweils auf ganz bestimmte Aspekte des Programmco- des beziehen. Die Halstead-Metriken sind Beispiele für die Messung der textuellen Komplexität. Sie lassen sich auf jede Programmiersprache anwenden.

Metriken zur strukturellen Komplexität

Zur Messung der strukturellen Komplexität von Komponenten werden häuﬁg die in Abbildung 14 veranschaulichten Metriken Fan-in und Fan-out bestimmt:

* Fan-in: Metrik zu einer Komponente, welche die Anzahl der Funktionen außerhalb einer Komponente bestimmt, die die Funktionen der betrachteten Komponente auf- rufen;
* Fan-out: Metrik zu einer Komponente, welche die Anzahl der Funktionen zurückgibt, die von der betrachteten Komponente auf andere Komponenten aufgerufen werden.



Statische Qualitätssicherung: Begutachten und Messen

Metriken für objektorientierte Systeme

Beispiele für typische Metriken für objektorientierte Systeme sind bei Grechenig (2010, S. 476–479):

* + Grad von Vererbungsbäumen, d. h. Länge des maximalen Pfades eines Vererbungs- baums;
  + Tiefe einer Klasse (DIT), d. h. die Länge des Pfades von der Wurzel bis zu einer Klasse im Vererbungsbaum;
  + Anzahl der direkten Unterklassen einer Klasse (NOC);
  + Kopplung zwischen Objekten (CBO): Anzahl der Klassen, die Methoden der gemesse- nen Klasse benutzen. Je höher die Kopplung zwischen Objekten ausgeprägt ist, umso aufwendiger wird es, die Klassen zu testen oder wiederzuverwenden;
  + Response for a class (RFC): Anzahl aller möglichen ausführbaren Methoden einer Klasse, dazu zählen sowohl die Methoden, die in der Klasse implementiert werden, als auch die Methoden, welche (durch Kopplung) in anderen Klassen aufgerufen werden können.

###### Diskussion von Metriken

Metriken sind Kennzahlensysteme und haben damit auch deren Vor- und Nachteile.

Vorteile

Softwaremetriken sind relativ einfach zu ermitteln. Für gängige Entwicklungsumgebun- gen gibt es Plug-ins, mit denen die Werte per Knopfdruck ermittelt und angezeigt wer- den können. Gleichfalls können Metriken zum Softwareprozess einfach ermittelt wer- den, vorausgesetzt die zu Grunde liegende Datenbasis wird kontinuierlich aktualisiert. Weiterhin sind die meisten Metriken unabhängig von der konkreten Programmierspra- che oder der Projektorganisation und können daher in den meisten Projekten einfach eingesetzt werden. Somit bieten insbesondere Softwaremetriken ein einfaches Mittel, um vielfältige Eigenschaften von Ergebnisartefakten zu ermitteln.

Nachteile

Mit Metriken können nur messbare Eigenschaften von Systemen oder Prozessen ermit- telt werden. Jedoch sind Aussagen über die tatsächliche Konsequenz bestimmter Eigen- schaften oder Kombinationen von Eigenschaften insbesondere mit Softwaremetriken oft nicht möglich. Zwar zeigen Untersuchungen, dass beispielsweise die Messgrößen von Haltsteadt-Metriken mit der tatsächlichen Lese- und Verständnisdauer korrelieren. Jedoch lassen sich auf Basis von Softwarevermessungen in der Regel keine gesicherten tatsächlichen Konsequenzen ableiten. Das Messergebnis, dass es sich um ein einfach zu verstehendes System handelt, dessen strukturelle Komplexität nicht sehr hoch ist, sagt noch nichts darüber aus, ob das System die Anforderungen des Kunden erfüllt.

Werden Metriken darüber hinaus als Bestandteile in Zielvereinbarungen aufgenommen, besteht die Gefahr, dass die Erfüllung der Metrik stärker in den Vordergrund rückt als die Erfüllung von fachlichen Anforderungen. Folgendes Beispiel soll dieses Risiko ver- anschaulichen: Entwickler sollen anhand der von ihnen erstellten LOC bewertet wer- den. Mit der Metrik Lines of Code, die sowohl Anweisungen als auch Quellcodekom-

mentare misst, wird ein Anreiz geschaffen, möglichst viele Kommentare zu schreiben. Mit der Metrik Non-Commented Source Code, die nur die Anzahl von Codezeilen ohne Quellcodekommentare ermittelt, wird hingegen ein Anreiz geschaffen, auf Kommentare zu verzichten und möglichst wenig Programmcode in eine Zeile zu schreiben.

Daher eignen sich Metriken nur sehr bedingt, um aus ihnen konkrete Aussagen über die Qualität von Systemen oder Prozessen abzuleiten. Dennoch können sie als Indiz für mögliche Fehlerquellen genutzt werden. So kann beispielsweise bei einem manuellen Codereview der Fokus auf besonders große oder als komplex bewertete Codestellen gelegt werden.

### Statische Codeanalyse

Verfahren zur statischen Codeanalyse werden zur qualitativen Bewertung von Pro- grammcode eingesetzt. Im Unterschied zu Metriken werden bei der statischen Code- analyse keine Eigenschaften des Programmcodes gemessen, sondern der Code wird inhaltlich analysiert und ausgewertet. Typische Anwendungsgebiete für die statische Codeanalyse sind die automatische Stilüberprüfung und die Analyse nach typischen Fehlermustern. Für industriell weit verbreitete Programmiersprachen wie Java oder C++ sind eine Reihe von Werkzeugen für die statische Codeanalyse verfügbar und können sowohl als Plugin in Entwicklungsumgebungen als auch in einen automatischen Build- Prozess eingebunden werden. Auf diese Weise wird sichergestellt, dass die Analyse automatisch und wiederholt ausgeführt wird. Im Folgenden werden exemplarisch drei verschiedene Werkzeuge für die Programmiersprache Java vorgestellt, die alle frei ver- fügbar sind.

###### Automatische Stilanalyse mit Checkstyle

Das Programm Checkstyle ist ein Werkzeug für die automatische Stilanalyse. Es prüft den Programmcode auf die Einhaltung von Programmierkonventionen. Beispiele hierfür sind die richtige Verwendung von Annotationen, die Einhaltung von Namenskonventio- nen für Klassen, Attribute, Methoden und Variablen sowie die Prüfung auf das Vorliegen von Javadoc-Kommentaren. Eine vollständige Übersicht über verfügbare Checks sind auf der Internetseite Checkstyle (o. J.) dokumentiert. Die konkret zu prüfenden Konven- tionen können im Einzelnen konﬁguriert werden, sodass neben den Standardkonventi- onen für Javacode auch eigene Konventionen erstellt und dann automatisch geprüft werden können.

Die Einhaltung von Programmierkonvention führt allerdings noch nicht zu einem fehler- freien System. Insbesondere bei manuellen Code-Reviews oder der Entwicklung im Team, bei der mehrere Entwickler an einer Komponente arbeiten, können mit der auto- matischen Stilanalyse die Einarbeitungszeit und das Codeverständnis gefördert werden. Darüber hinaus erleichtert ein einheitlich formatierter Programmcode die Arbeit mit

Statische Qualitätssicherung: Begutachten und Messen

Versionskontrollsystemen. Bei einer einheitlichen Formatierung werden in der Ände- rungsverfolgung weniger Formatierungsänderungen, beispielsweise Zeileneinrückun- gen, angezeigt.

###### Quelltextanalyse mit PMD

Das Programm PMD analysiert wie Checkstyle den Quellcode von Programmen. Hierbei steht jedoch die Suche nach möglichen Fehlerquellen im Vordergrund. Typische Bei- spiele für solche Codestellen, die mit PMD gefunden werden können, sind:

* + mögliche Bugs durch leere try/catch- oder switch-Blöcke;
  + erstellte, aber nicht genutzte Variablen, Parameter und private Methoden;
  + überﬂüssiger Einsatz von String und StringBuffer-Methoden;
  + Identiﬁkation unnötiger oder zu komplizierter if-Statements sowie
  + Codestellen, die durch Copy&Paste dupliziert wurden.

Ein Überblick über alle von PMD unterstützten Regeln ist im Internet verfügbar (PMD

o. J.).

###### Bytecode-Analyse mit SpotBugs

Das Programm SpotBugs ist ein Werkzeug zur Analyse von Java-Bytecode auf typische Fehlermuster. Der Quelltext wird hierzu nicht benötigt, das Programm muss sich aber zur Analyse mit SpotBugs erfolgreich kompilieren lassen.

Eine aktuelle Übersicht der verfügbaren Fehlermuster ist im Internet verfügbar (Spot- Bugs o. J.). SpotBugs unterteilt die Fehlermuster in Kategorien. Beispiele für hierfür sind:

* + Bad practice (dt. sinngemäß: schlechte Angewohnheit): beispielsweise das Imple- mentieren der equals()-Methode ohne die hashCode()-Methode oder ein Stringver- gleich mit == statt mit equals();
  + Correctness (dt.: Korrektheit): beispielsweise die falsche Verwendung der Inkrement- funktion (i++) bei Return-Statements oder das Hinzufügen eines Collection-Objektes zu sich selbst;
  + Dodgy code (dt.: schlechter Code): beispielsweise das Speichern eines Wertes in einer lokalen Variable, die niemals gelesen wird;

sowie die Kategorien Malicious code vulnerability (dt.: Verletzbarkeit durch schädlichen Code), Multithreaded correctness (dt.: Mulitithread-Korrektheit) und Performance (dt.: Ausführungsgeschwindigkeit).

Sowohl die Quellcodeanalyse mit PMD als auch die Bytecode-Analyse mit SpotBugs sind Hilfsmittel zur Identiﬁkation von tatsächlichen Fehlern. Beide Analysewerkzeuge geben Hinweise, wo im Code gegen eine „gute Praxis“ verstoßen wird. Ob es sich dabei jedoch um tatsächliche Fehler handelt, muss durch die Entwickler manuell geprüft wer-

den. Mit der statischen Codeanalyse können jedoch typische Fehler bereits bei der Pro- grammierung identiﬁziert werden, sodass bei späteren Tests die Wahrscheinlichkeit für typische Flüchtigkeitsfehler reduziert werden können. Es gibt jedoch keine Garantie, dass der Programmcode, der keine Fehlermuster enthält, auch wirklich funktioniert.

Zusammenfassung

Statische Verfahren der Qualitätssicherung sind analytische Verfahren, bei denen der Prüﬂing im Rahmen eines Reviews analysiert, begutachtet, untersucht wird und die dabei gewonnenen Informationen zusammengetragen und gegebenenfalls in Metriken oder Kennzahlen verdichtet und ausgewertet werden. Im Gegensatz zu dynamischen Verfahren wird der Prüﬂing hierbei nicht ausgeführt.

Review-Techniken sind manuelle statische Prüfverfahren, die sich hinsichtlich des Aufwandes und Grades der Strukturierung unterscheiden. So ist eine Stellung- nahme eine sehr einfache und wenig aufwendige Technik, eine Inspektion hingegen eine sehr stark strukturierte und aufwendige Begutachtungstechnik. Ein Walkth- rough ist schwächer strukturiert als eine Inspektion jedoch stärker als eine Stel- lungnahme.

Eine weitere Technik zur Bestimmung der Qualität von Produkten und Prozessen mit statischen Verfahren ist die Bestimmung von Softwaremetriken durch Vermes- sen von Programmcodeeigenschaften des Systems. Beispiele hierfür sind Halstead- Metriken zur Messung der textuellen Komplexität.

Darüber hinaus können Verfahren zur statischen Codeanalyse für die qualitative Bewertung von Programmcode eingesetzt werden. Mit ihnen wird der Code inhalt- lich analysiert und ausgewertet. Typische Anwendungsgebiete für die statische Codeanalyse sind die automatische Stilüberprüfung und die Analyse nach typi- schen Fehlermustern im Code.



# Lektion 5

## Dynamische Qualitätssicherung: Testen

#### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie wissen, …

… was ein Testfall ist, welche Teststufen unterschieden werden und welche typischen Techniken zur Testfallerstellung es gibt.

… was die anwendungsfallbasierte Testfallerstellung ist und wie sie eingesetzt wird.

… wie Testfälle mit Äquivalenzklassenbildung erstellt werden.

… wie Testfälle mit der zustandsbasierten Testfallerstellung erstellt werden.

… wie die Erstellung von Zufallsdaten eingesetzt werden kann.

DL-D-IQSS01-L05

1. Dynamische Qualitätssicherung: Testen

### Einführung

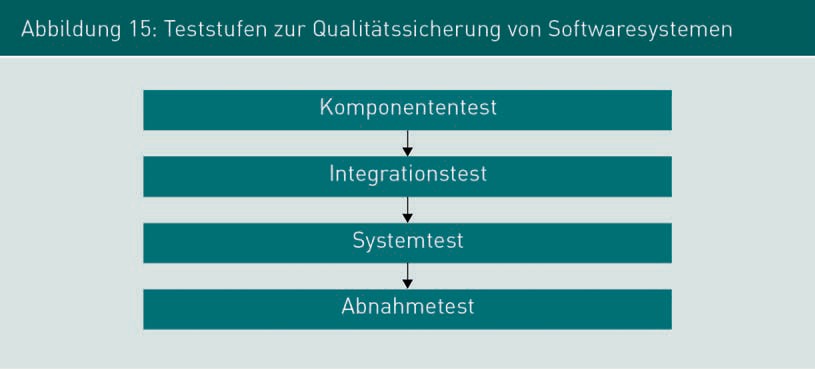
Im Vergleich zu den statischen Verfahren der analytischen Qualitätssicherung bezeich- net der Begriff „Testen“ die Durchführung von dynamischen Verfahren. Der Prüﬂing in der dynamischen Qualitätssicherung ist immer ein Stück Software. Beim Testen wird Software mit konkreten Eingabewerten ausgeführt und das Ergebnis der Ausführung bewertet.

### Einsatz und Überblick über dynamische Verfahren

Dynamische Verfahren zählen zu den analytischen QS-Maßnahmen. Sie werden auch testende Verfahren, Softwaretests oder auch nur Test genannt. Beim Testen von Soft- ware werden ein System oder Teile eines Systems zur Ausführung gebracht, konkrete Eingabewerte übergeben und das Systemverhalten bzw. das erzeugte Ergebnis bewer- tet. Tests sind daher diagnostische Maßnahmen zur Prüfung und Bewertung der Quali- tät von Softwareartefakten nach deren Erstellung.

###### Teststufen

Die Aktivitäten zum Testen erfolgen nicht erst nach Abschluss aller Entwicklungstätig- keiten, sondern entwicklungsbegleitend. Daher wird in der industriellen Softwareent- wicklung bereits mit dem Testen begonnen, sobald die ersten Softwareartefakte erzeugt wurden. Abbildung 15 zeigt eine schematische Übersicht über die verschiedenen Test- stufen im Software Engineering.



Welche konkreten Qualitätseigenschaften im Fokus der aktuellen Prüfung stehen, wird neben den für das Projekt festgelegten Qualitätszielen auch durch die aktuelle Test- stufe bestimmt. Daher unterscheiden sich die konkreten Testfälle beim Testen je nach Teststufe.

Dynamische Qualitätssicherung: Testen

###### Testfall

Ein Testfall dient als Handlungsanweisung zur Durchführung von Softwaretests und besteht mindestens aus den folgenden Elementen:

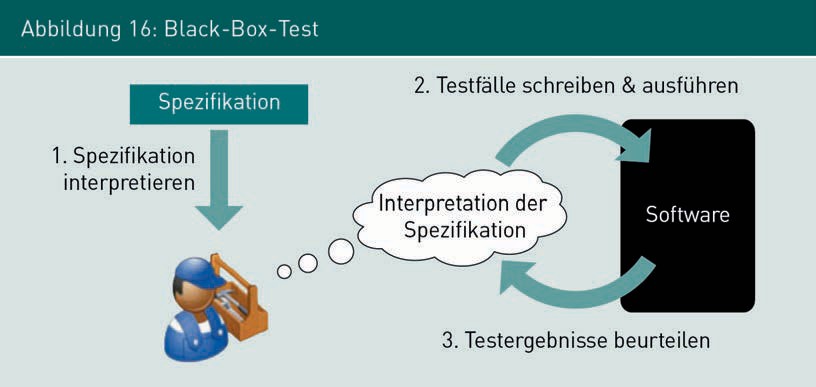
* Vorbedingungen, die vor dem Prüfschritt sichergestellt werden müssen;
* Testdaten bzw. Testaktionen, die während des Prüfschrittes eingegeben bzw. durch- geführt werden;
* Nachbedingungen, deren Erfüllung nach dem Prüfschritt geprüft wird sowie
* beschreibende Daten zum Testfall, wie Name, ID, betreffende Komponente, Funktion oder Eigenschaft, die geprüft wird, Anwendungsfall oder Name des Erstellers.

###### Testfallerstellung

In der Literatur zum Thema Softwaretest werden verschiedene Testtechniken anhand der Art und Weise unterschieden, wie die Testfälle erzeugt werden. Dabei können Black- Box-Tests und White-Box-Tests unterschieden werden.

Black-Box-Tests

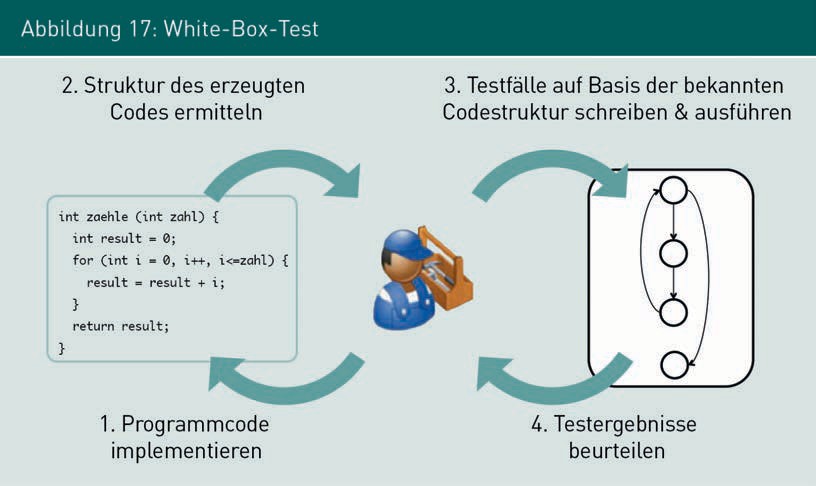
Ziel der Black-Box-Tests ist die Prüfung aller in der Speziﬁkation geforderten Funktio- nen und Eigenschaften zu einem System oder einer Systemkomponente. Bei der Test- fallerstellung steht eine möglichst große Abdeckung der speziﬁzierten Funktionen eines Systems im Vordergrund. Über die innere Struktur des Prüﬂings liegen keine Informati- onen vor, denn nur das nach außen hin sichtbare Verhalten ist relevant. Abbildung 16 stellt schematisch das Vorgehen bei Black-Box-Tests dar: Der Tester liest und interpre- tiert die Speziﬁkation. Dann erstellt er anhand seiner Interpretation der Speziﬁkation die Testfälle und führt diese aus. Anschließend beurteilt der Tester die Testergebnisse anhand seiner Interpretation der Speziﬁkation.



Mit Black-Box-Tests wird in der Regel das Zusammenspiel verschiedener Systemkompo- nenten geprüft. Sie sind dabei sehr anwendernah und werden in der Regel bei Integra- tions-, System- und Abnahmetests eingesetzt. Insbesondere für Black-Box-Tests erhält die Speziﬁkation eine zentrale Bedeutung: Auf Basis der Speziﬁkation wird der Grad der Testabdeckung bestimmt; sie wird als Basis für die Erstellung der Testfälle genutzt und anhand der Speziﬁkation werden die Testergebnisse beurteilt. Daher ist es umso wich- tiger, dass bei der Erstellung der Speziﬁkation sorgfältig darauf geachtet wird, dass in ihr alle tatsächlich von den Stakeholdern benötigten Funktionen und Eigenschaften dokumentiert werden.

White-Box-Tests

White-Box-Tests werden mit Kenntnis des Programmcodes erstellt, der im Gegensatz zu den Black-Box-Tests hier bekannt ist. Ziel bei den White-Box-Tests ist eine möglichst große Abdeckung der im Programmcode implementierten Anweisungen und Kontroll- strukturen (wie if, while, for, switch), um dabei möglichst alle implementierten Anwei- sungen zu testen. Abbildung 17 stellt schematisch das Vorgehen bei White-Box-Tests dar. Nachdem ein Teil des Programmcodes implementiert ist, wird die tatsächlich erzeugte Struktur des Codes analysiert. Auf dieser Basis werden dann Testfälle formu- liert, mit denen eine möglichst hohe Abdeckung des erzeugten Programmcodes erreicht wird. Nach der Ausführung werden die Ergebnisse ausgewertet und bei Bedarf der Pro- grammcode angepasst, bevor erneut getestet wird. Häuﬁg werden hierbei die Rollen Entwickler und Tester von einer Person eingenommen, daher werden diese Tests auch Entwicklertests genannt.



White-Box-Tests werden in der Regel begleitend zur Implementierung und bei Kompo- nenten- und Integrationstests durchgeführt.

Zusätzlich unterscheidet Liggesmeyer (2002, S. 50–51) auch noch zwischen funktions- orientierten und kontrollﬂussorientierten Tests.

Dynamische Qualitätssicherung: Testen

###### Testabdeckung

Grundsätzlich ist es in der Praxis nicht möglich, industrielle Informationssysteme voll- ständig zu testen. Ziel der QS-Maßnahmen ist daher keine vollständige, sondern eine kostenoptimale Qualitätssicherung. Bei der Durchführung von Tests müssen sich die Tester daher auf eine gezielte Auswahl von Testfällen und Testdaten beschränken. Ein Maß für die Vollständigkeit von Testfällen ist die sogenannte Testabdeckung. Je nach der eingesetzten Methode kann anhand der Testfallerstellung bestimmt werden, wie

„vollständig“ die Menge der formulierten Testfälle das System prüft. In der Praxis wird die Testabdeckung eingesetzt, um die für das Projekt erforderlichen Testaktivitäten zu beschränken.

###### Auswahl von Testfällen

Sowohl bei Black-Box-Tests als auch bei White-Box-Tests wird ein System oder ein Teil eines Systems mit konkreten Eingabedaten ausgeführt. Da nicht alle theoretisch mögli- chen Kombinationen aus Eingabedaten praktisch getestet werden können, ist das Tes- ten von Informationssystemen immer ein Stichprobenverfahren. Die Korrektheit von Systemen kann mit Tests nicht nachgewiesen werden. Eine der wesentlichen Herausfor- derungen beim Testen ist somit die Auswahl geeigneter Testfälle und Testdaten, damit mit möglichst wenig Aufwand eine möglichst hohe Testabdeckung erreicht werden kann. Anders ausgedrückt müssen aus der Grundgesamtheit aller möglichen Eingabe- werte genau die ausgesucht werden, mit denen möglichst viele potenzielle Fehler iden- tiﬁziert werden können.

In Tabelle 8 werden verschiedene Techniken zur Auswahl von Testdaten mit ihren cha- rakteristischen Eigenschaften vorgestellt und beschrieben.

Testabdeckung

Dies ist das Maß für die Vollständigkeit von Testfällen. Der konkrete Messwert ist abhängig von der Art des Tests und der gewählten Tech- nik der Testfaller- stellung.

|  |  |  |
| --- | --- | --- |
| Tabelle 8: Techniken zur Testdatenermittlung | | |
| Name der Technik | Beschreibung | Einsatz |
| Anwendungsfall- basierte Testfall- erstellung | Ableitung von Testdaten aus dokumentierten Anwendungs- fällen (Use Cases), zu denen häuﬁg fachliche Vorbedingun- gen, das zu erreichende Ziel und Nachbedingungen doku- mentiert sind. | Test, ob die festgelegten fachlichen Ziele inner- halb des Use Cases erreicht werden können; in der Regel nur als Black-Box-Test anwend- bar. |

|  |  |  |
| --- | --- | --- |
| Name der Technik | Beschreibung | Einsatz |
| Äquivalenzklas- senbildung | Zusammenfassen von mögli- chen Eingabewerten, mit denen das gleiche fachliche Ergebnis in Äquivalenzklassen erzielt wird. Jede Äquivalenz- klasse wird durch einen Ver- treter getestet. | Ermittlung von Eingabe- daten für Fachfunktionen, gezieltes Testen des Sys- temverhaltens mit gülti- gen und ungültigen Wer- ten. |
| Grenzwertanalyse | Erstellen von Testdaten gezielt an den Grenzen von gültigen und ungültigen Werteberei- chen, oft in Kombination mit Äquivalenzklassenbildung. | Testen des Systemverhal- tens gezielt an gültigen und ungültigen Werten bzw. an Grenzen von Äquivalenzklassen, die jeweils ein anderes fach- liches bzw. technisches Verhalten erfordern. |
| Zustandsbasierte Testfallerstellung | Erstellen von fachlichen Test- daten auf Grundlage des Lebenszyklus oder fachlichen Zuständen von Geschäftsob- jekten (Black-Box-Test); Erstel- len von technischen Testdaten zur Überprüfung von techni- schen Zuständen (White-Box- Test) des Systems, einer Kom- ponente oder einer Klasse sowie deren Zustandsüber- gänge. | Fachlicher oder techni- scher Zustandstest, mit dem auch explizit die Nicht-Durchführung von verbotenen Zustands- übergängen getestet wer- den kann; insbesondere relevant, wenn bei der Speziﬁkation von System- verhalten Zustandsdiagr- amme oder Zustandsta- bellen eingesetzt werden. |
| Erstellung von Zufallstestdaten | Automatische, nicht-determi- nistische Erstellung von Test- daten; Daten werden in der Regel nicht völlig zufällig erzeugt; häuﬁg wird von den Eingabedaten gefordert, dass deren Struktur bestimmte Anforderungen erfüllt. | Ergänzung von struktu- rierten Methoden zur Testdatenerzeugung; eig- net sich beispielsweise in Kombination mit der Äquivalenzklassenbil- dung. |

Dynamische Qualitätssicherung: Testen

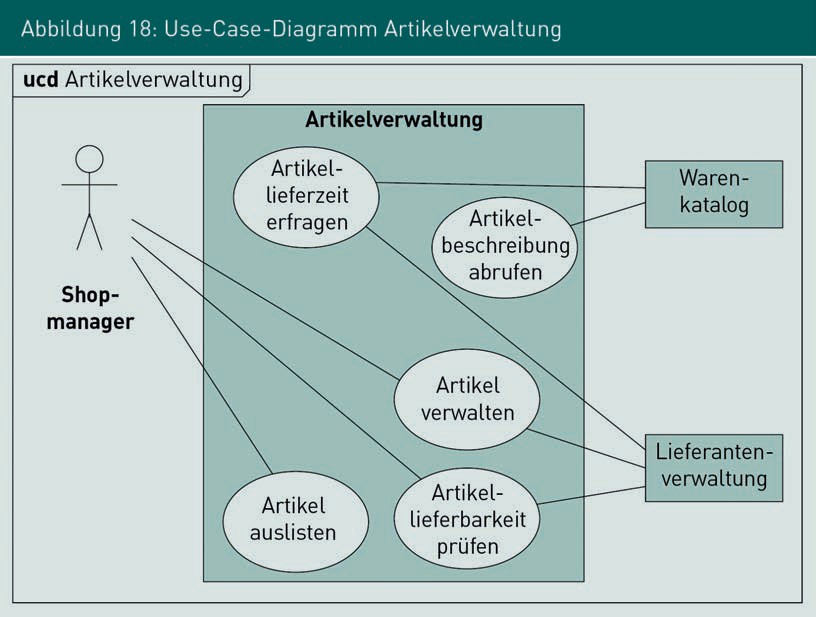
|  |  |  |
| --- | --- | --- |
| Name der Technik | Beschreibung | Einsatz |
| Ursache-Wirkungs- Analyse (siehe Lig- gesmeyer 2002,  S. 66–76) | Prüfung von Beziehungen und Wechselwirkungen von Äquiva- lenzklassen; Ergebnis dieser Prüfung sind Kombinationen verschiedener Äquivalenzklas- sen, die jeweils in ein ganz bestimmtes Systemverhalten resultieren. | Prüfung von Anwen- dungsverhalten (Wir- kung), das die Kombina- tion verschiedener Eingabewerte (Ursachen) bestimmt. |
| Anweisungsüber- deckung (siehe Liggesmeyer 2002,  S. 85–88) | Erstellung von Testdaten, mit denen jede Anweisung im Code (White-Box-Test) mindes- tens 1 x ausgeführt wird; Erstellung von Testdaten, mit denen jede Fachfunktion (Black-Box-Test) des unter- stützten Geschäftsprozesses mindestens 1 x ausgeführt wird. | Sicherstellen, dass jede relevante Anweisung/ Funktion beim Test 1 x aufgerufen wird. |
| Zweigüberdeckung (siehe Ligges- meyer 2002, S. 88–  93) | Erstellung von Testdaten, mit denen jeder Zweig im Kontroll- ﬂuss mindestens 1 x durchlau- fen wird; d. h., Entscheidungen im Kontrollﬂuss müssen so getroffen werden, dass jeder ausgehende Zweig mind. 1 x durchlaufen wird. | Sicherstellen, dass jeder mögliche Zweig im Quell- code (White-Box-Test) bzw. dass jeder mögliche Zweig im Geschäftspro- zess (Black-Box-Test) erreicht und durchlaufen werden kann. |
| Bedingungsüber- deckung (siehe Liggesmeyer 2002,  S. 93–117) | Erstellung von Testdaten, die jede einzelne logische Ent- scheidung im Programmcode (White-Box-Test) oder an Ent- scheidungsknoten im Geschäftsprozess (Black-Box- Test) jeweils mind. 1 x zu WAHR und mind. 1 x zu FALSCH aus- werten lassen. | Prüfen von zusammen- gesetzten Entscheidun- gen und dem aus ihrem Verhalten resultierenden Systemverhalten. |

|  |  |  |
| --- | --- | --- |
| Name der Technik | Beschreibung | Einsatz |
| Pfadtest (siehe Liggesmeyer 2002,  S. 118–132) | Erstellung von Testdaten, sodass der gesamte Kontroll- ﬂusspfad innerhalb einer Komponente im Programm- code (White-Box-Test) oder ein Pfad durch den Geschäftspro- zess (Black-Box-Test) von Anfang bis Ende durchlaufen werden kann. Dabei muss sichergestellt sein, dass dabei auch der mehrfache Durchlauf von Schleifen getestet wird.  Häuﬁg wird gefordert, dass eine Schleife 0 x, 1 x und 2 x durchlaufen werden soll. | Fachlicher Ende-Zu-Ende- Test durch den gesamten fachlichen Prozess (Black- Box-Test) bzw. durch eine Methode oder System- komponente (White-Box- Test), Prüfen des System- verhaltens gezielt beim Schleifendurchlauf. |
| Entscheidungsta- belle (siehe Lig- gesmeyer 2002, S. 79f.) | Erstellung von fachlichen Test- daten, mit denen die Auswer- tung aller relevanter Bedin- gungen im Programmcode (White-Box-Test) bzw. in spezi- ﬁzierten Geschäftsregeln (Black-Box-Test) geprüft wer- den kann. | Prüfen des Systemver- haltens bei der Auswer- tung und Umsetzung von Geschäftsregeln (Black- Box-Test, White-Box-Test) als auch von komplexen Kontrollﬂussentschei- dungen; insbesondere relevant, wenn bei der Speziﬁkation Entschei- dungstabellen eingesetzt wurden. |
| Exploratives Tes- ten | Manuelle, nach Gefühl und Erfahrung erzeugte Testdaten, von denen der Tester glaubt, mögliche Fehler identiﬁzieren zu können. Können bei einem großen Erfahrungsschatz im Testen der relevanten System- klasse bzw. des konkreten Sys- tems nach Weiterentwicklung sehr efﬁzient sein. Ist metho- disch die schwächste Technik und nicht geeignet zur Erfül- lung formaler Testkriterien. | Wird häuﬁg von Kunden oder Fachanwendern ohne Kenntnisse von Methoden zur Testdaten- erzeugung durchgeführt. Einsatz bei großem Erfah- rungsschatz mit dem Sys- tem, bei dem es bei bestimmten Funktionen oder Eingabedaten immer zu Problemen kommt. |

Dynamische Qualitätssicherung: Testen

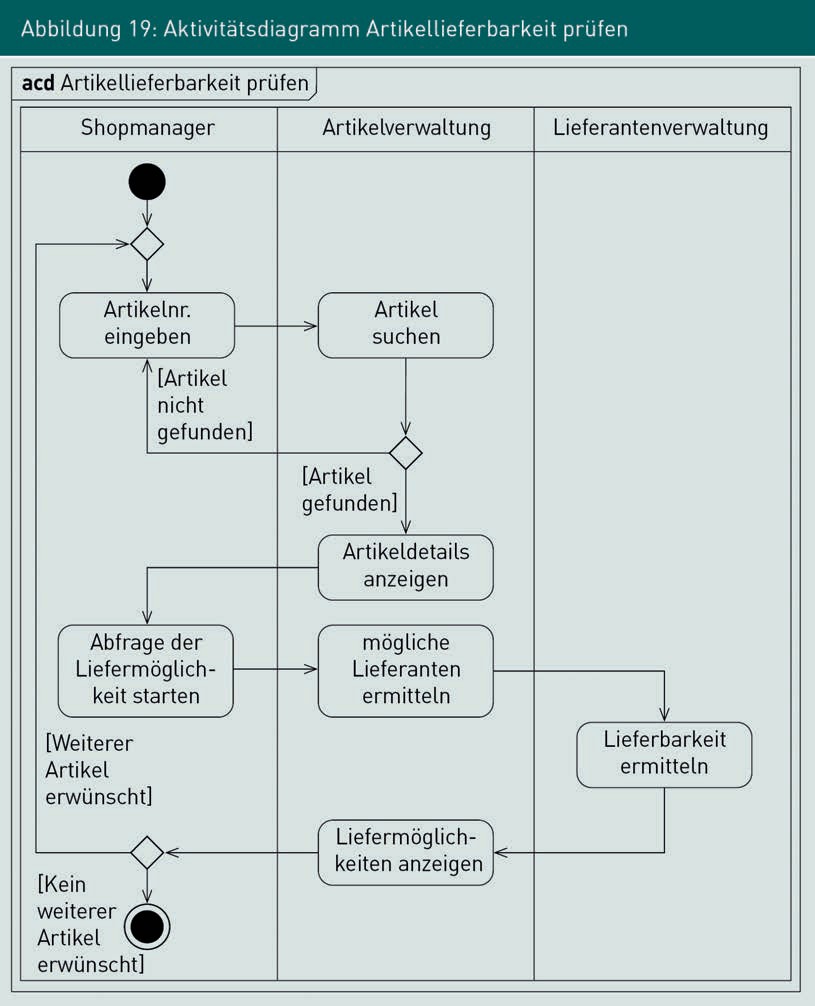
### Anwendungsfallbasierte Testfallerstellung

Bei der anwendungsfallbasierten Testfallerstellung werden die Testfälle aus den fachli- chen Anwendungsfällen, die beispielsweise mit Use-Cases-Diagrammen dokumentiert sind, abgeleitet. Dabei sollte mindestens zu jedem Anwendungsfall ein Testfall erstellt werden. Aus dem Use-Case-Diagramm in Abbildung 18 würden sich demnach mindes- tens fünf Testfälle ergeben.



Häuﬁg umfassen Use Cases mehrere komplexe fachliche Abläufe, die beispielsweise mit Aktivitätsdiagrammen wie in Abbildung 19 dargestellt, speziﬁziert werden. In diesem Fall können durch Analyse des Aktivitätsdiagramms verschiedene Testfälle für den Use Case

„Artikellieferbarkeit prüfen“ abgeleitet werden. Für den Fall, dass keine UML-Diagramme erstellt wurden, muss der beschreibende Text zu dem Use Case analysiert werden.



###### Vorgehen bei der Testfallerstellung

Da häuﬁg nicht alle Anwendungsfälle eines Systems detailliert getestet werden können, müssen besonders relevante Anwendungsfälle identiﬁziert werden. Kriterien für die Auswahl von detailliert zu testenden Use Cases sind unter anderem:

* + Wertbeitrag: Wie hoch ist der Beitrag dieser Funktion zur Wertschöpfungskette?
  + Nutzungshäuﬁgkeit: Wie viele Nutzer greifen wie häuﬁg auf die Funktion zu?

Dynamische Qualitätssicherung: Testen

* Schadenspotential: Welcher Schaden kann durch nicht identiﬁzierte Fehler in der Funktion entstehen?
* Typische Fehler: Welche Funktionen bergen häuﬁg Fehler in der Umsetzung? Wo sind die typischen Schwachstellen bei vergleichbaren Alt- oder Konkurrenzsystemen?
* Geforderte Testabdeckung: Welcher Grad der Testabdeckung ist für die fachliche Ebene tatsächlich gefordert?

Anhand dieser Kriterien können auch die Funktionen komplexer Use Cases analysiert werden: Fachlich wichtige Funktionen werden intensiv getestet, zu weniger wichtigen Funktionen wird nur ein Mindestmaß an Testabdeckung gefordert.

Beispiel

Für ein Projekt wird entschieden, dass Anwendungsfälle nach ihrer Relevanz in zwei Kategorien eingeteilt werden: wichtige Anwendungsfälle, deren Funktionen möglichst intensiv getestet werden, und weniger wichtige Anwendungsfälle, zu denen nur gefor- dert wird, dass alle ihre speziﬁzierten fachlichen Funktionen während des Tests min- destens 1 x ausgeführt werden.

###### Testabdeckung

Als Kriterien für die geforderte Testabdeckung von Funktionen eines Use Cases können die oben bereits erwähnten folgenden Überdeckungskriterien dienen:

* Anweisungsüberdeckung: Jede Funktion wird mindestens 1 x aufgerufen.
* Zweigüberdeckung: Jeder Kontrollﬂusspfeil wird mindestens 1 x durchlaufen.
* Bedingungsüberdeckung: Jede Bedingung wird mindestens 1 x zu WAHR und 1 x zu FALSCH ausgewertet.
* Pfadtest: Jeder mögliche Pfad durch den Use Case wird vollständig („am Stück“) durchlaufen, dabei werden mögliche Schleifen berücksichtigt.

Liegt zu einem Anwendungsfall ein Aktivitätsdiagramm, wie in Abbildung 18 dargestellt, oder eine vergleichbare Darstellung der Abläufe eines Anwendungsfalls vor, wird die Ermittlung der benötigten Testfälle gemäß der geforderten Testabdeckung deutlich erleichtert. Welche Funktionen in welcher Reihenfolge in einem Testfall ausgeführt wer- den müssen, kann direkt aus dem Ablaufdiagramm abgelesen werden.

### Äquivalenzklassenbildung und Grenzwertanalyse

###### Äquivalenzklassenbildung

Die Äquivalenzklassenbildung ist eine Technik zur Auswahl konkreter Testdaten bei der Testfallerstellung auf der Ebene von Funktionen. Dabei werden alle möglichen Eingabe- werte, die zu einer Funktion gemacht werden können, in sogenannte Äquivalenzklassen eingeteilt. Der Begriff Äquivalenzklasse kommt aus der Algebra und bedeutet sinnge-

mäß: Menge gleichartiger Elemente. Bezogen auf die Erstellung von Testfällen bedeutet dies, dass alle Eingabewerte, die in ein fachlich identisches Systemverhalten resultie- ren, in einer Äquivalenzklasse zusammengefasst werden. Bei der Durchführung der Tests müssen dann nicht alle möglichen Eingabewerte getestet werden, sondern nur jeweils ein beliebig ausgewählter Vertreter aus jeder Äquivalenzklasse. Auf diese Weise lassen sich die Anzahl der benötigten Testdaten pro Testfall sehr deutlich reduzieren. Die Äquivalenzklassenbildung kann grundsätzlich für alle Tests eingesetzt werden, in denen dem System oder den Systemkomponenten Daten übergeben (beispielsweise bei GUI-Tests oder Tests von technischen Schnittstellen) oder Funktionen aufgerufen (in White-Box-Test) werden.

###### Beispiele für den Einsatz von Äquivalenzklassen

Eine oder mehrere Bedingungen, die erfüllt sein müssen

Die Erreichbarkeit bzw. die Ausführung von Funktionen ist abhängig von Bedingungen, die zu Geschäftsobjekten erfüllt sein müssen. Beispiel: Die Buchung von Konferenzräu- men einer Firma ist nur durch fest angestellte Mitarbeiter erlaubt.

Mögliche Äquivalenzklassen:

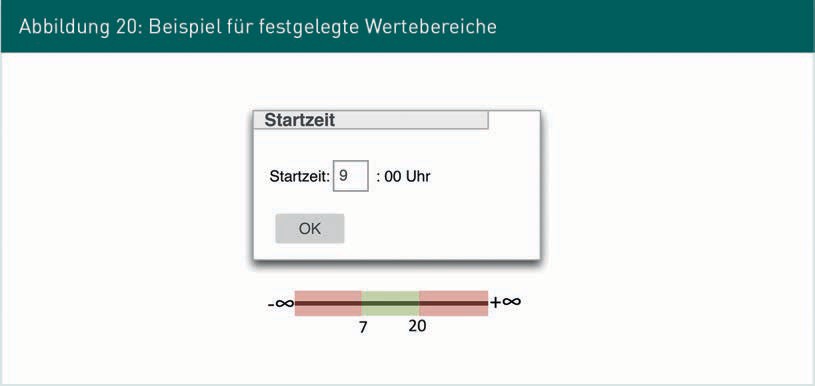
* + Gültige Äquivalenzklasse A1: alle fest angestellten Mitarbeiter
  + Ungültige Äquivalenzklasse A2: alle freien und alle externen Mitarbeiter

Der Test, ob der Zugriff nur für fest angestellte Mitarbeiter möglich ist, wird nun nicht mit allen möglichen Datensätzen zu den Festangestellten durchgeführt, sondern mit genau einem Datensatz. Für den Test, ob für Nicht-Festangestellte keine Buchung mög- lich ist, wird genau ein Datensatz aus der Menge der möglichen Nicht-Festangestellten ausgewählt. Auf diese Weise wird durch die Bildung von Äquivalenzklassen die Menge der erforderlichen Testdaten deutlich eingeschränkt.

Festgelegter Wertebereich

Eine Funktion erwartet Eingabewerte in bestimmten Wertebereichen. Abbildung 20 zeigt dazu einen beispielhaften (vereinfachten) Eingabedialog, bei dem für eine Startzeit zur vollen Stunde ein Zahlenwert zwischen 7 und 20 erwartet wird.

Dynamische Qualitätssicherung: Testen

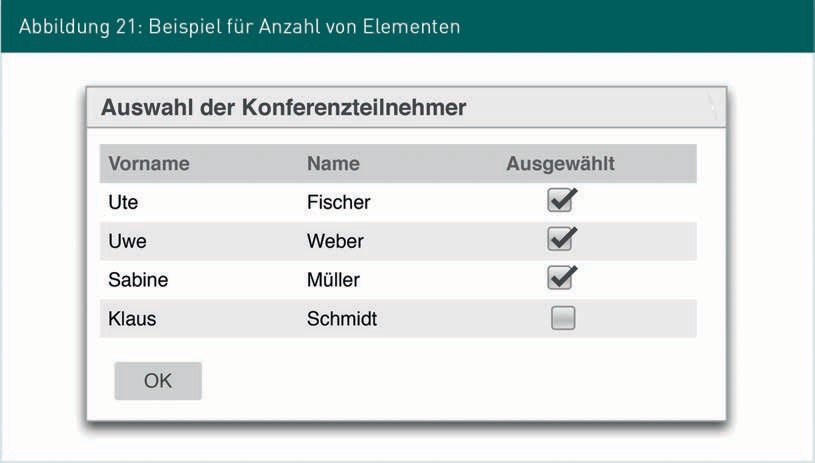


Für die Konzeption der Testfälle ergeben sich drei Äquivalenzklassen:

* Gültige Äquivalenzklasse B1: alle Zahlen zwischen 7 und 20: {x|7 ≤ x ≤ 20 und x ist ganze Zahl}
* Ungültige Äquivalenzklasse B2: alle Zahlen kleiner als 7: {x|x < 7}
* Ungültige Äquivalenzklasse B3: alle Zahlen größer als 20: {x|20 < x}

Anzahl von Elementen einer Menge

In einer Funktion ist nur eine ganz bestimmte Anzahl von Elementen einer bestimmten Eingabemenge erlaubt. In Abbildung 21 wird exemplarisch eine Dialogmaske gezeigt, in der die Teilnehmer einer Konferenz zur Buchung eines Konferenzraums festgelegt wer- den. Hierbei müssen mindestens drei und maximal 25 Teilnehmer ausgewählt werden.



Äquivalenzklassen für gültige Werte:

* + Äquivalenzklasse C1: 3 ≤ |Teilnehmer| ≤ 25; alle Eingaben bei der eine beliebige Zahl zwischen 3 und 25 ausgewählt wurde.

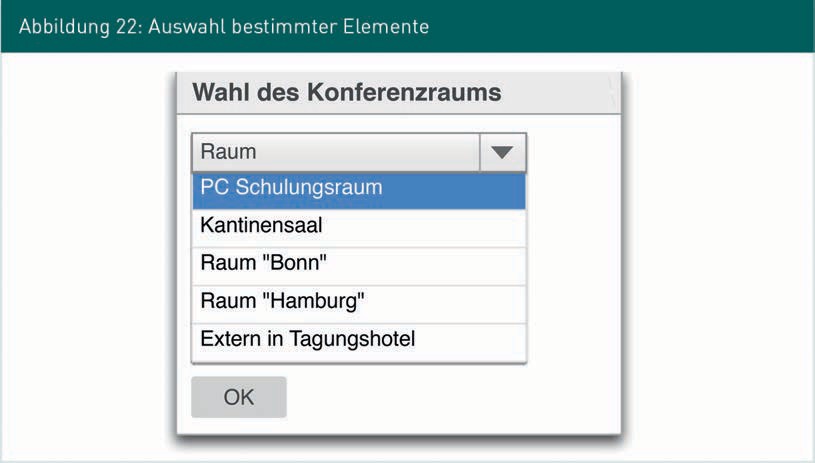
Äquivalenzklassen für ungültige Werte:

* + Äquivalenzklasse C2: |Teilnehmer| < 3; alle Eingaben, bei denen weniger als 3 Teil- nehmer ausgewählt wurden.
  + Äquivalenzklasse C3: 25 < |Teilnehmer|; alle Eingaben, bei denen mehr als 25 Teil-

nehmer ausgewählt wurden

Auswahl Elemente einer Menge, die jeweils unterschiedlich behandelt werden

In einer Funktion können Werte ausgewählt werden, die jeweils ein unterschiedliches Systemverhalten bzw. unterschiedliche Fachfunktionen auslösen. Abbildung 22 zeigt einen GUI-Dialog, in dem ein konkreter Konferenzraum für eine Konferenz ausgewählt werden kann. Die anschließenden Schritte zur Vorbereitung und Reservierung von Standardkonferenzräumen verhalten sich gegenüber dem System gleich, alle besonde- ren Räume erfordern jedoch jeweils verschiedene Abläufe. Insbesondere darf der Kan- tinensaal nur für bestimmte Anlässe gebucht werden, die jedoch noch nicht mit dem System unterstützt werden.



Für dieses Beispiel ergeben sich folgende Äquivalenzklassen:

* + Gültige Äquivalenzklasse D1: {Raum „Bonn“, Raum „Hamburg“}
  + Gültige Äquivalenzklasse D2: {PC Schulungsraum}
  + Gültige Äquivalenzklasse D3: {Extern in Tagungshotel}
  + Ungültige Äquivalenzklasse D4: {Kantinensaal}

Dynamische Qualitätssicherung: Testen

Hinweis

Die Auswahl ungültiger Elemente dient hier zur Veranschaulichung und kann bei der Implementierung der GUI in der Regel verhindert werden.

###### Unterschied zu Äquivalenzklassen in der Mathematik

In Abweichung zum Begriff der Äquivalenzklasse in der Mathematik müssen Testdaten als Elemente von Äquivalenzklassen nicht disjunkt sein. Beispiel (siehe auch Abbildung 23): Äquivalenzklasse A = {0,1,2,3,4,5,6,7,8,9 | Menge aller Ziffern}, Äquivalenzklasse B

= {0,2,4,6,8 | Menge aller geraden Ziffern}, Äquivalenzklasse C = {1,3,5,7,9 | Menge

aller ungeraden Ziffern}. Äquivalenzklasse A und Äquivalenzklasse B haben gemeinsame Elemente, sie sind daher nicht disjunkt.



###### Vorgehen bei der Testfallerstellung

Das Vorgehen zur Erstellung von Testfällen mit Einsatz der Äquivalenzklassenbildung erfolgt in den hier folgenden Schritten:

Schritt 1

Zunächst werden die Äquivalenzklassen für jeden Eingabeparameter der Funktion bzw. jedes GUI-Element der Dialogmaske bestimmt und eindeutig gekennzeichnet. Das Ergebnis von Schritt 1 kann bei Bedarf aus Gründen der Übersichtlichkeit in einer Tabelle dokumentiert werden. Tabelle 9 zeigt das Ergebnis von Schritt 1 aus dem Bei- spiel zur Buchung von Konferenzräumen.

|  |  |  |
| --- | --- | --- |
| Tabelle 9: Menge aller Äquivalenzklassen zur Konferenzraumbuchung – Beispiel | | |
| Eingabeparameter | Gültige Äquivalenzklassen | Ungültige Äquivalenzk- lassen |
| Festanstellung | A1 | A2 |
| Startuhrzeit | B1 | B2, B3 |
| Auswahl Teilnehmer | C1 | C2, C3 |
| Auswahl Konferenzraum | D1, D2, D3 | D4 |

Schritt 2

Es werden Testfälle erstellt, mit denen alle gültigen Äquivalenzklassen abgedeckt wer- den. Die Anzahl der erforderlichen Testfälle ist dabei so gering wie möglich zu halten. Daher sollte die Kombination der Testdaten so gewählt werden, dass in möglichst wenig Testfällen möglichst viele Äquivalenzklassen getestet werden. Tabelle 10 zeigt eine mögliche Kombination aus drei Testfällen, die alle gültigen Äquivalenzklassen für das Beispiel abdecken.

|  |  |  |  |
| --- | --- | --- | --- |
| Tabelle 10: Beispiel – Testfälle für gültige Äquivalenzklassen | | | |
| Eingabewert | Testfall 1 | Testfall 2 | Testfall 3 |
| Festanstellung | A1 | A1 | A1 |
| Startuhrzeit | B1 | B1 | B1 |
| Auswahl Teilneh- mer | C1 | C1 | C1 |
| Auswahl Konfe- renzraum | D1 | D2 | D3 |
| Getestete gültige Äquivalenzklas- sen | A1, B1, C1, D1 | A1, B1, C1, D2 | A1, B1, C1, D3 |

Dynamische Qualitätssicherung: Testen

Grundsätzlich sollte jede Äquivalenzklasse nur einmal getestet werden, denn eine Wie- derholung von Tests mit den gleichen Daten bringt keinen zusätzlichen Erkenntnisge- winn. In Testfall 2 und Testfall 3 werden die bereits getesteten Äquivalenzklassen A1, B1 und C1 jedoch wiederholt verwendet, da diese für die erstmalige Ausführung von D2 und D3 benötigt werden.

Schritt 3

Es werden Testfälle erstellt, mit denen alle ungültigen Äquivalenzklassen abgedeckt werden. Im Unterschied zu den Testfällen für gültige Äquivalenzklassen werden bei den Testfällen für ungültige Äquivalenzklassen pro Testfall nur eine ungültige Äquivalenz- klasse getestet. Alle anderen Testdaten werden aus gültigen Äquivalenzklassen genom- men. So kann im Fall eines fehlgeschlagenen Tests die betreffende ungültige Äquiva- lenzklasse einfacher identiﬁziert werden. Würden mehrere ungültige Äquivalenzklassen in einem Testfall getestet, kann im Fall eines Test-Fehlschlags nicht sicher gesagt wer- den, welche der ungültigen Äquivalenzklassen die Ursache dafür ist. Tabelle 11 zeigt exemplarisch für das Konferenzraumbeispiel die Testfälle für ungültige Äquivalenz- klassen.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tabelle 11: Beispiel – Testfälle für ungültige Äquivalenzklassen | | | | | | |
| Testfälle | # 4 | # 5 | # 6 | # 7 | # 8 | # 9 |
| Festanstellung | A2 | A1 | A1 | A1 | A1 | A1 |
| Startuhrzeit | B1 | B2 | B3 | B1 | B1 | B1 |
| Auswahl Teilneh- mer | C1 | C1 | C1 | C2 | C3 | C1 |
| Auswahl Konfe- renzraum | D1 | D1 | D1 | D1 | D1 | D4 |
| Getestete ungül- tige Äquivalenz- klassen | A2 | B2 | B3 | C2 | C3 | D4 |

Schritt 4

Abschließend werden konkrete Eingabedaten auf Basis der ausgewählten Äquivalenzk- lassen für jeden Testfall erstellt. Zum Testen der Beispielfunktion „Buchung des Konfe- renzraumes“ wurden insgesamt neun Testfälle identiﬁziert. In Tabelle 12 ist beispielhaft für jeden Testfall ein konkreter Testdatensatz erstellt worden, mit denen die Funktion bezüglich der identiﬁzierten Äquivalenzklassen vollständig getestet wurde.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tabelle 12: Beispiel – Konkrete Testdaten (1) | | | | | |
| Testfälle | # 1 | # 2 | # 3 | # 4 | # 5 |
| Festanstellung | Fester MA | Fester MA | Fester MA | Externer MA | Fester MA |
| Startuhrzeit | 8 | 8 | 8 | 8 | 3 |
| Auswahl Teil- nehmer | 10 | 10 | 10 | 10 | 10 |
| Auswahl Konfe- renzraum | Raum  „Bonn“ | PC Schu- lungs- raum | Extern in Tagungs- hotel | Raum  „Bonn“ | Raum  „Bonn“ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tabelle 12: Beispiel – Konkrete Testdaten (2) | | | | |
| Testfälle | # 6 | # 7 | # 8 | # 9 |
| Festanstellung | Fester MA | Fester MA | Fester MA | Fester MA |
| Startuhrzeit | 24 | 8 | 8 | 8 |
| Auswahl Teilneh- mer | 10 | 1 | 30 | 10 |
| Auswahl Konfe- renzraum | Raum  „Bonn“ | Raum „Bonn“ | Raum  „Bonn“ | Kantinen- saal |

Bewertung der Äquivalenzklassenbildung

Vorteile

Die Äquivalenzklassenbildung ist eine einfache Technik zur Ermittlung von Testdaten und Testfällen. Sie kann für jede technische oder fachliche Funktion eingesetzt werden, die Eingabedaten erwartet. Auf Basis der tabellarischen Darstellung ist sowohl eine einfache Erstellung der Testfälle als auch eine intuitive Dokumentation möglich.

Dynamische Qualitätssicherung: Testen

Nachteile

Fachliche Abhängigkeiten zwischen einzelnen Äquivalenzklassen werden dabei jedoch nicht mit berücksichtigt; jede Äquivalenzklasse ist grundsätzlich unabhängig. Darüber hinaus ist die Äquivalenzklassenbildung ungeeignet für Komponenten oder Systeme, deren Verhalten von deﬁnierten internen Zuständen abhängig ist, beispielsweise Zustände von Geschäftsobjekten oder Systemkomponenten.

###### Grenzwertanalyse

Die Grenzwertanalyse ist wie die Äquivalenzklassenbildung eine Technik zur Auswahl von konkreten Eingabedaten. Die der Grenzwertanalyse zu Grunde liegende Annahme ist die Beobachtung, dass Softwarefehler häuﬁg in dem Grenzbereich liegen, der den Übergang von gültigen zu ungültigen Eingabewerten darstellt, d. h. an den Grenzen von Äquivalenzklassen. Daher wird die Grenzwertanalyse häuﬁg in Kombination mit der Äquivalenzklassenbildung eingesetzt. Bei der Erstellung von Testfällen mit der Grenz- wertanalyse werden gezielt gültige und ungültige Werte berücksichtigt, die an den Grenzen von Äquivalenzklassen liegen. Konkret werden bei der Grenzwertanalyse der am dichtesten an der Grenze liegende gültige Wert sowie der am dichtesten an der Grenze liegende ungültige Wert ermittelt und als Eingabewert für die Testfälle verwen- det.

Beispiel

In der oben gezeigten Abbildung 20 wurden folgende Äquivalenzklassen identiﬁziert:

* Gültige Äquivalenzklasse B1, B: alle Zahlen zwischen 7 und 20: {x|7 ≤ x ≤ 20 und x ist ganze Zahl}
* Ungültige Äquivalenzklasse B2: alle Zahlen kleiner als 7: {x|x < 7}
* Ungültige Äquivalenzklasse B3: alle Zahlen größer als 20: {x|20 < x}

Es gibt hierbei zwei Grenzen von gültigen zu ungültigen Äquivalenzklassen: die Grenze von B1 zu B2 und die Grenze von B1 zu B3. In Tabelle 13 sind alle relevanten Grenzwerte aufgeführt, die im Rahmen der Grenzwertanalyse identiﬁziert wurden.

|  |  |  |
| --- | --- | --- |
| Tabelle 13: Beispiel – Grenzwerte | | |
|  | Gültige Grenzwerte | Ungültige Grenzwerte |
| Grenze B1 zu B2 | 7 | 6 |

|  |  |  |
| --- | --- | --- |
|  | Gültige Grenzwerte | Ungültige Grenzwerte |
| Grenze B1 zu B3 | 20 | 21 |

Bei der Kombination von Äquivalenzklassenbildung und Grenzwertanalyse werden häu- ﬁg als Testdaten neben den Grenzwerten auch noch jeweils ein Vertreter der Äquiva- lenzklasse in die Menge der Testdaten aufgenommen, die im mittleren Wertebereich und nicht in der Nähe der Grenze liegen. Im Vergleich zur Äquivalenzklassenbildung erweitert die Grenzwertanalyse die Menge der zu berücksichtigenden Testdaten und damit auch die Menge der durchzuführenden Testfälle. Müssen jedoch zur Erreichung der erforderlichen Testabdeckung Werte einer Äquivalenzklasse in mehreren Tests ein- gesetzt werden, können hierbei gezielt verschiedene Grenzwerte eingesetzt werden, um die Gesamtmenge der notwendigen Testfälle möglichst gering zu halten.

Beispiel

So werden beispielsweise bei den oben gezeigten Testfällen in Tabelle 11 und Tabelle 12 insgesamt sieben Mal ein Wert der Äquivalenzklasse B1 benötigt. In diesem Fall können sowohl die Grenzwerte 7 und 20 aber auch Werte aus dem mittleren Wertebereich, wie 12 oder 14, als Vertreter von B1 eingesetzt werden. Auf diese Weise können die gültigen Grenzen ohne Mehraufwand getestet werden.

### Zustandsbasierte Testfallerstellung

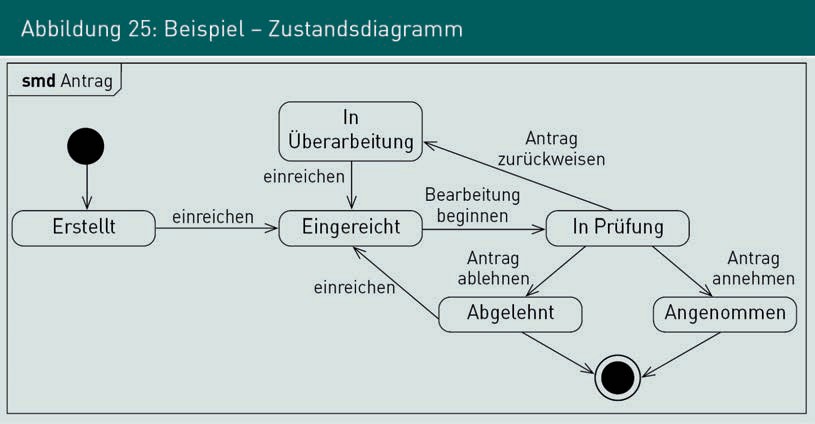
Die zustandsbasierte Testfallerstellung wird bei Systemen oder Systemkomponenten eingesetzt, deren Verhalten maßgeblich durch einen internen fachlichen oder techni- schen Zustand gesteuert wird. Beispiele für fachliche Zustände sind die Phasen in Lebenszyklen von Geschäftsobjekten (wie: beantragt, angenommen, abgelehnt) oder Prozessen (wie: eröffnet, Gutachten beauftragt, Schaden bewertet, Schaden reguliert). Beispiele für technische Zustände sind interne Zustände von Systemkomponenten oder Protokollen an technischen Schnittstellen. Daher kann die zustandsbasierte Testfaller- stellung sowohl für White-Box-Tests als auch für Black-Box-Tests eingesetzt werden. Insbesondere wenn Zustandsdiagramme als Teil der Speziﬁkation eingesetzt werden, können auf dieser Grundlage relativ einfach Testfälle abgeleitet werden. Grundsätzlich werden mit der zustandsbasierten Testfallerstellung solche Testfälle erstellt, mit denen die Erreichung von deﬁnierten Zuständen über die dafür vorgesehenen Zustandsüber- gänge (auch: Transitionen) geprüft wird. Dabei wird explizit auch die Nicht-Unterstüt- zung von nicht-vorgesehenen Zustandsübergängen durch das System geprüft.

###### Vorgehen bei der Testfallerstellung

Die Testfälle werden auf Grundlage der speziﬁzierten Zustände und Zustandsübergänge erstellt. Diese können mit Zustandsdiagrammen, Zustandstabellen, Text oder einer Mischform speziﬁziert werden. Abbildung 25 zeigt beispielhaft ein Zustandsdiagramm,

Dynamische Qualitätssicherung: Testen

das Teil der Speziﬁkation ist, und auf dessen Grundlage Testfälle erstellt werden sollen. Wurde bei der Speziﬁkation kein Zustandsdiagramm oder keine Zustandstabelle einge- setzt, müssen diese bei der Testfallerstellung gegebenenfalls aus der Speziﬁkation abgeleitet werden.



Schritt 1: Erstellen einer Zustandsübergangstabelle

Schwerpunkt der Darstellung eines Zustandsdiagramms ist die graﬁsche Darstellung des gewünschten Systemverhaltens. Nicht gewünschtes Verhalten wird im Zustandsdia- gramm nicht explizit speziﬁziert, muss jedoch bei der Durchführung von Softwaretests explizit ausgeschlossen werden. Daher muss zur Testfallerstellung auch nicht erwünschtes Verhalten explizit beschrieben werden. So wird beispielsweise durch Tests sichergestellt, dass im System durch keine einzige Funktion ein Zustandsübergang von

„abgelehnt“ zu „angenommen“ möglich ist. Zur einfachen Identiﬁkation von erlaubten und nicht erlaubten Transitionen wird das Zustandsdiagramm in eine Zustandsüber- gangstabelle überführt. Abbildung 26 veranschaulicht die Elemente einer typischen Zustandsübergangstabelle: Pro deﬁnierten Zustand im Diagramm wird jeweils eine Spalte und eine Zeile angelegt.

Schritt 2: Ausfüllen der Tabelle

Gültige Kombinationen aus Zuständen und Transitionen im Diagramm führen zu einem Folgezustand. Der Name einer Transition wird jeweils genau in die Zelle notiert, deren Ausgangszustand durch die Spalte und deren Zielzustand durch die Zeile vorgegeben werden.

Alle Zellen, die nach der Übertragung der im Diagramm deﬁnierten Transitionen leer bleiben, sind Zustandsübergänge, die das System nicht ausführen darf. Sie werden bei- spielsweise mit „-“ in der Tabelle notiert. In Abbildung 26 ist im Zustand „Eingereicht“ keine Transition in den Zustand „Erstellt“ möglich. Wohingegen die Transition „einrei- chen“ im Zustand „Erstellt“ in den Folgezustand „Eingereicht“ führt.

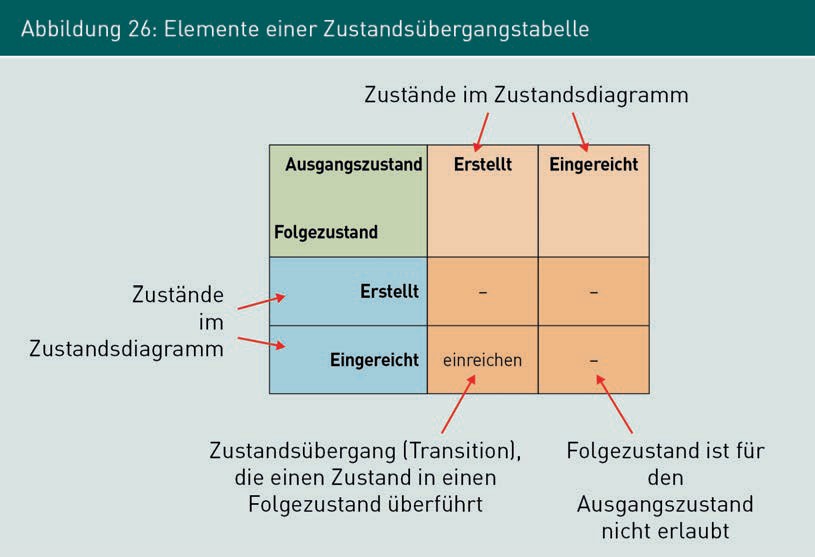
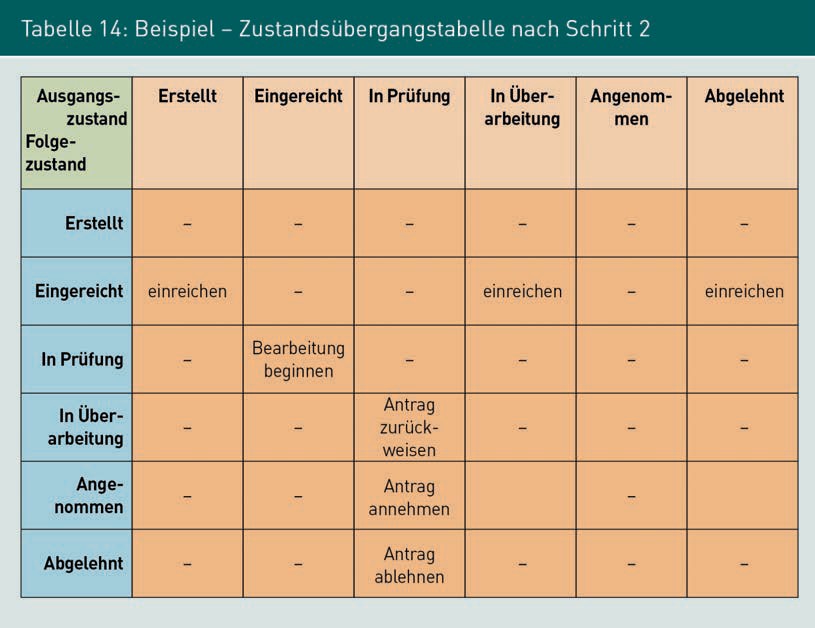


Tabelle 14 enthält eine vollständige Zustandsübergangstabelle zu dem in Abbildung 26 dargestellten Zustandsdiagramm.



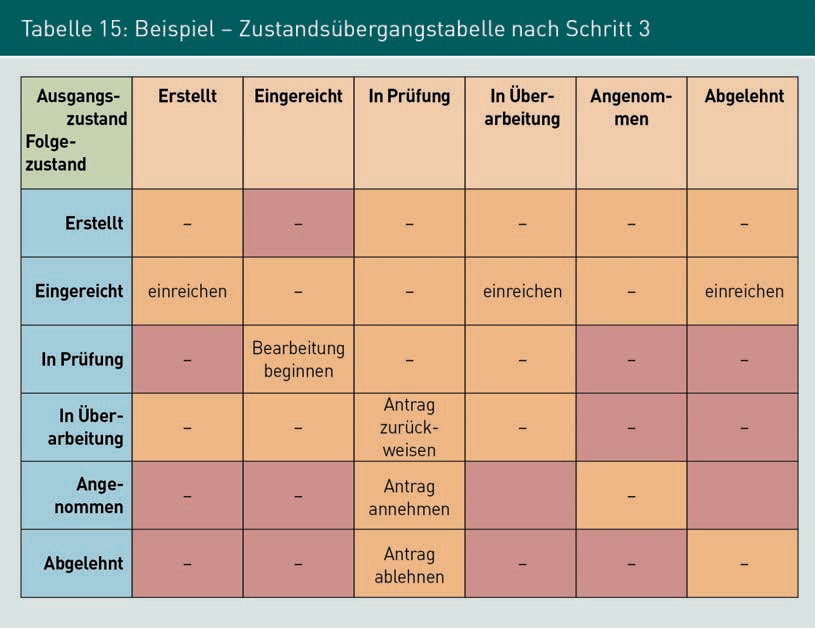
Dynamische Qualitätssicherung: Testen

Schritt 3: Fehlerzustände in Tabelle eintragen

Je nach Größe der Zustandsübergangstabelle und dem Einsatzgebiet des zu testenden Systems können die nicht erlaubten Transitionen (mit „-“ in der Tabelle markiert) in zwei Kategorien eingeteilt werden:

* Kategorie 1: Fehlersituationen, deren Auftreten zu fachlichen Fehlern führt und daher sicher zu vermeiden sind. Beispiel: Eine Transition vom Zustand „Erstellt“ in den Zustand „Angenommen“, da ein Antrag in jedem Fall vor einer Entscheidung geprüft werden muss.
* Kategorie 2: Nicht erlaubte, jedoch triviale Transitionen, die, falls sie doch durchge- führt werden, nicht zu einem fachlichen Fehlverhalten führen. Beispiel: Eine Transi- tion vom Zustand „Eingereicht“ in den Zustand „In Überarbeitung“, da hierbei keine fachlich wichtige Entscheidung übergangen wird.

Die Kategorisierung ist für die Ableitung von Testfällen relevant. So kann sich bei der Testfallerstellung auf die echten Fehlersituationen konzentriert werden. Nicht erlaubte Transitionen, die zu keinem fachlichen Fehler führen, werden nur mit niedriger Priorität getestet. Die Zuordnung der nicht erlaubten Transitionen in die Kategorien erfordert Fachwissen und ist gegebenenfalls mit den fachlichen Stakeholdern abzustimmen. Tabelle 15 zeigt die Zustandsübergangstabelle mit rot gekennzeichneten Fehlersituatio- nen, deren Nicht-Umsetzung bei der Erstellung der Testfälle in jedem Fall mit berück- sichtigt werden muss.



So muss sichergestellt werden, dass die Endzustände „Angenommen“ und „Abgelehnt“ nur aus dem Zustand „In Prüfung“ erreichbar sind. Ist jedoch einer der beiden Zustände bereits erreicht, führt eine Transition mit dem gleichen Folgezustand wie dem Ausgangszustand zu keinem fachlichen Fehler.

Schritt 4: Testfälle aus Tabelle ableiten

Aus der nach Schritt 3 vorliegenden Zustandsübergangstabelle mit gekennzeichneten Fehlerzuständen können nun Testfälle abgeleitet werden. Ergebnis der Testfallerstel- lung ist eine Menge aus Folgen von Transitionen, mit denen die vom System zu imple- mentierenden Funktionen als auch die vom System nicht zu unterstützenden Transitio- nen getestet werden können. Je nach Größe der Tabelle und den Anforderungen an die Testfälle können dabei folgende verschiedene Stufen der Testabdeckung unterschieden werden:

* + Alle Zustände: In Bezug auf das Zustandsdiagramm ist diese Testabdeckung das Minimalziel. Hierzu werden Folgen von Transitionen bestimmt, mit denen jeder deﬁ- nierte Zustand mindestens 1 x erreicht wird.
  + Alle erlaubten Zustandsübergänge: Bei dieser Stufe der Testabdeckung sind alle in der Zustandsübergangstabelle deﬁnierten Transitionen mindestens 1 x ausgeführt worden.
  + Alle Fehlersituationen: Hierbei werden zusätzlich zu allen erlaubten Zustandsüber- gängen auch alle Fehlersituationen auf Nicht-Umsetzung getestet.
  + Alle nicht erlaubten Zustandsübergänge: Ergänzend zu den Fehlersituationen wer- den hierbei auch alle nicht erlaubten Zustandsübergänge, die keine Fehlersituatio- nen sind, auf Nicht-Umsetzung geprüft.

Abbildung 27 stellt die Abhängigkeiten der Abdeckungsgrade für zustandsbasierte Tests dar: So ist der Test aller Zustände beispielsweise im Test aller erlaubten Zustandsüber- gänge enthalten. Der Test aller nicht erlaubten Zustandsübergänge enthält neben dem Test aller Fehlersituationen auch den Test aller erlaubten Zustandsübergänge:

Alle Zustände ⊂ Alle erlaubten Zustandsübergänge ⊂ Alle Fehlersituationen ⊆ Alle nicht erlaubten Zustandsübergänge.

Dynamische Qualitätssicherung: Testen



###### Bewertung der zustandsbasierten Tests

Vorteile

Die zustandsbasierte Testfallerstellung lässt sich auf Grundlage speziﬁzierter Zustands- diagramme sehr leicht anwenden, daher ist sie in der Praxis weit verbreitet. Mithilfe der Zustandsübergangstabelle lassen sich sowohl die Testfälle leicht ableiten, die Testab- deckung veranschaulichen und beispielsweise durch Einfärben von Tabellenzellen der aktuelle Fortschritt der Testdurchführung sowie das Testergebnis darstellen. Wurde der Dialogﬂuss von GUIs mit Zustandsdiagrammen speziﬁziert, kann die zustandsbasierte Testfallerstellung auch für die Erstellung von GUI-Tests eingesetzt werden.

Nachteile

Werden sehr viele zu testende Zustände identiﬁziert, wird die Zustandsübergangsta- belle schnell sehr groß und unübersichtlich. Bei fachlichen Zustandsdiagrammen wer- den häuﬁg nur die Funktionen im Zustandsdiagramm notiert, die einen echten Zustandsübergang darstellen. In der Regel gibt es auch fachliche Funktionen, die den fachlichen Zustand nicht verändern. Unter Umständen werden diese Funktionen bei der Testfallerstellung vergessen. Beispiele hierfür sind Funktionen für Drucken oder Bear- beiten von im System gespeicherten Daten.

### Erstellung von Zufallstestdaten

Bei der Erstellung von Zufallstestdaten werden Testdaten erzeugt, die als Eingabewerte für zu testende Funktionen genutzt werden. Diese Technik wird beispielsweise für den Test von Benutzeroberﬂächen mit vielen Eingabeelementen oder für das Testen des Datenaustausches über technische Schnittstellen eingesetzt. Das Grundprinzip der Testdatenerstellung basiert darauf, dass im Rahmen eines gegebenen Wertebereichs von Eingabeparametern gültige Werte zufällig erzeugt werden. So können beispiels-

weise Grenzen von Wertebereichen mit der Äquivalenzklassenbildung ermittelt werden und die Festlegung der konkreten Eingabewerte über Zufallstestdaten erfolgen. In Abhängigkeit der Anforderungen an die Testfälle können aber auch komplexe Zufalls- testdaten auf Grundlage bereits bestehender Testdatensätze erfolgen. Dabei wird eine vorgegebene, nicht-zufällig erzeugte Menge von Testdaten nach einem festgelegten Prinzip zufällig kombiniert. Auf diese Weise wird eine möglichst reale Grundstruktur der Testdaten sichergestellt, ohne jedoch in jedem Testfall exakt die gleichen Daten zu ver- wenden.

Beispiel

Zu testen ist ein komplexer Eingabedialog einer Webanwendung, der unter anderem die Eingabe von Anschriften erfordert. Dazu wird eine Datenbank mit je 50 gültigen Postleitzahlen, Ortsnamen, Straßennamen und Hausnummern aufgebaut. Durch die zufällige Kombination dieser Daten können 504 verschiedene Testanschriften generiert werden, was einer Gesamtmenge von 6.250.000 Varianten entspricht.

###### Bewertung der Zufallstestdaten

Vorteile

In Kombination mit der Äquivalenzklassenbildung ist die Erstellung von Zufallstestda- ten eine Technik, bei der mit vergleichbar geringem Aufwand viele verschiedene Test- datensätze generiert werden können. Sobald ein geeigneter Testdatengenerator exis- tiert, können in der Regel beliebig viele Testdatensätze erzeugt werden. Durch die automatische Erzeugung wird sichergestellt, dass jeder Testdatensatz die geforderten Eigenschaften hat und menschliche Fehler durch eine manuelle Testdatenerzeugung ausgeschlossen werden können.

Nachteile

Die zufällige Erstellung von Testdaten erzeugt zwar viele verschiedene, jedoch nicht in jedem Fall wirklich realistische Daten. Sollen beispielsweise die Adressdaten aus dem oben genannten Beispiel automatisch mit einer Adressvalidierung geprüft werden, hel- fen zufällig erzeugte, jedoch unrealistische Testdaten nicht weiter. Daher müssen vor der Erstellung des Testdatengenerators die Anforderungen an die Struktur und den Inhalt der Testdaten detailliert speziﬁziert werden. Ob die dazu erforderlichen Auf- wände in einem sinnvollen Verhältnis zum Nutzen stehen, muss für jedes Projekt indi- viduell entschieden werden.

Die Erstellung von Zufallstestdaten ist daher eine häuﬁg sinnvolle Ergänzung zu ande- ren Techniken, als einzige Technik jedoch in der Regel nicht ausreichend.

Dynamische Qualitätssicherung: Testen

Zusammenfassung

Dynamische Testverfahren (auch: Softwaretests) sind analytische QS-Maßnahmen bei denen ein System oder Teile eines Systems ausgeführt werden. Die verschiede- nen Testtechniken werden anhand der Art und Weise unterschieden, wie die Test- fälle erzeugt werden. Dabei können grundsätzlich Black-Box-Tests und White-Box- Tests unterschieden werden. Da jedoch nicht alle möglichen Kombinationen aus Eingabedaten praktisch getestet werden können, muss eine Menge möglichst geeig- neter Testfälle und Testdaten identiﬁziert werden, die mit möglichst wenig Aufwand eine möglichst hohe Testabdeckung erreichen.

Bei anwendungsfallbasierten Testfallerstellungen werden die Testfälle aus den fachlichen Anwendungsfällen abgeleitet, die beispielsweise mit Use-Cases-Dia- grammen dokumentiert sind.

Die Äquivalenzklassenbildung ist eine Technik zur Auswahl konkreter Testdaten auf der Ebene von Funktionen. Dabei werden alle Eingabewerte, die in ein fachlich identisches Systemverhalten resultieren in einer Äquivalenzklasse zusammenge- fasst.

Bei der zustandsbasierten Testfallerstellung werden Testfälle auf Basis eines Zustandsdiagramms oder einer Zustandsübergangstabelle erstellt. Dabei wird expli- zit auch die Nicht-Unterstützung von nicht-vorgesehenen Zustandsübergängen durch das System geprüft.



# Lektion 6

## Systematisches Testen von Software

#### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie wissen, …

… welche Aktivitäten zum methodischen Testen allgemein für jede Teststufe beschrieben werden können.

… was Komponententests sind und wie sie durch automatische Unit-Tests und Test Driven Development unterstützt werden können.

… was Integrationstests sind und welche Integrationsstrategien es gibt.

… was Systemtests und Abnahmetests sind und welche verschiedenen Arten von Testschwerpunkten es dort jeweils gibt.

DL-D-IQSS01-L06

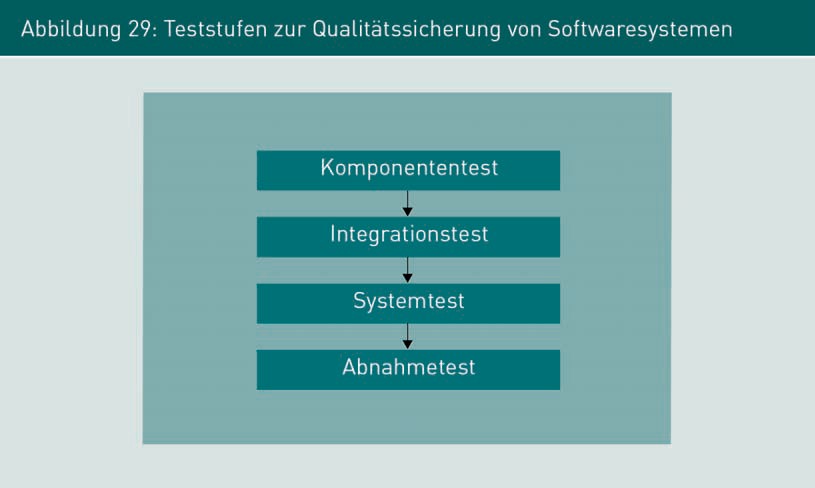
1. Systematisches Testen von Software

### Einführung

Das Testen von Software erfolgt je nach aktuellem Projektfortschritt in den unter- schiedlichen Teststufen Komponententest, Integrationstest, Systemtest und Abnahme- test. In dieser Lektion werden zunächst die Aktivitäten vorgestellt, die zum methodi- schen Testen auf allen Teststufen durchgeführt werden müssen. Anschließend werden die einzelnen Teststufen im Detail beschrieben und dabei konkrete Techniken und Vor- gehensweisen vorgestellt.

### Aktivitäten zum methodischen Testen

Beim Testen des erzeugten Softwaresystems werden sogenannte Teststufen unterschie- den. Wie bereits erwähnt, ist es zu risikoreich mit der Qualitätssicherung von Software bis zum Abschluss der Implementierung zu warten. Daher wird in der industriellen Soft- wareentwicklung mit Testaktivitäten bereits begonnen, sobald die ersten Softwareartef- akte erzeugt wurden. Abbildung 29 zeigt eine schematische Übersicht über die ver- schiedenen Teststufen im Software Engineering.

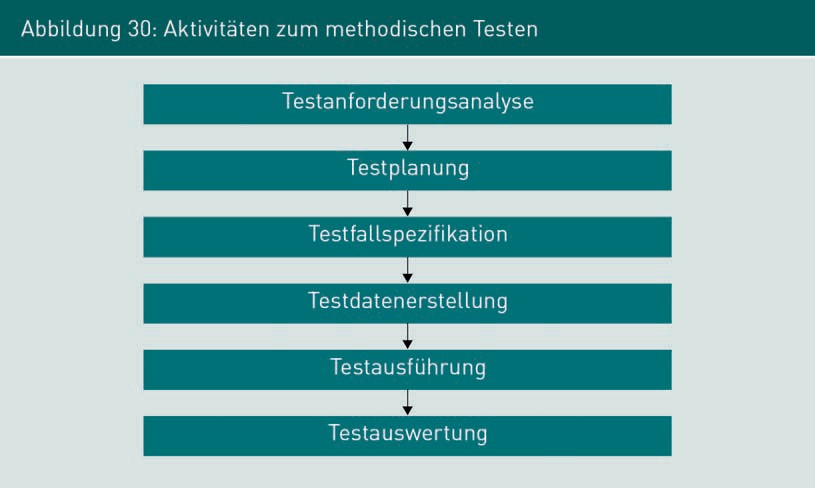


Komponententests werden häuﬁg von den Entwicklern selbst durchgeführt, um die von ihnen erstellten Funktionen und Systemkomponenten isoliert vom Gesamtsystem zu testen. Bei den Integrationstests wird gezielt das Zusammenspiel mehrerer Komponen- ten getestet, die über technische Schnittstellen miteinander verbunden sind. Das ganze System wird im Rahmen des Systemtests getestet. Hierbei wird neben dem Testen von fachlichen Abläufen durch das gesamte System auch das Systemverhalten unter Ext-

Systematisches Testen von Software

rembedingungen hinsichtlich der Einhaltung von geforderten Qualitätseigenschaften getestet. Die erläuterten Abnahmetests werden seitens des Auftraggebers durchgeführt, der auf Grundlage des Ergebnisses das erstellte Softwaresystem abnimmt.

Zur methodischen Durchführung von Softwaretests lassen sich die in Abbildung 30 gezeigten Aktivitäten identiﬁzieren, die unabhängig von der Teststufe zu jedem Test durchgeführt werden müssen. Je nach Komplexität des durchzuführenden Tests sind diese Aktivitäten jedoch unterschiedlich umfangreich. So wird die Testplanung von Komponententests in der Regel nur sehr kurz durchgeführt, während die Planungen für einen umfassenden Systemtest mehrere Tage oder Wochen dauern können.



###### Testanforderungsanalyse

Wie oben bereits beschrieben, können industrielle Informationssysteme nicht vollstän- dig getestet werden. Der Aufwand, der zur Fehlervermeidung nach der Auslieferung des Systems betrieben wird, sollte jedoch möglichst kostenoptimal sein. Daher muss für jedes System individuell bestimmt werden, welche Risiken es durch nicht erkannte Feh- ler gibt. Die so identiﬁzierten Risiken müssen durch die Gestaltung der konkreten QS- Maßnahmen adressiert werden. Die Testanforderungsanalyse ist in etwa vergleichbar mit der Anforderungsanalyse des Softwaresystems: Hier werden die zu erreichenden Qualitätsziele bzw. der zu erreichende Qualitätsgrad bestimmt und es wird beschrie- ben, welche Systemfunktionen und welche Systemeigenschaften getestet werden sol- len. Das Ergebnis der Testanforderungsanalyse sind noch recht grobe Beschreibungen der Testfälle oder des Testgegenstandes, ohne dabei auf Details zum Ablauf oder zu den Testdaten einzugehen.

Beispiel

„Teste, ob der Konferenzraum nur von berechtigten Personen reserviert werden kann“ oder „Teste die Antragseinreichung, wenn sie zeitgleich von 2000 Nutzern eingesetzt wird“.

###### Testplanung

Im Anschluss an die Anforderungsanalyse werden die Tests geplant. Je komplexer die Vorbereitung, Durchführung und Auswertung eines Tests sind, desto sorgfältiger muss die Testplanung erfolgen. Typischerweise ist diese Aktivität bei Systemtests am stärk- sten ausgeprägt. Falls das vollständige System in einer möglichst produktionsnahen Umgebung getestet werden soll, ist die Einrichtung einer komplexen Testumgebung erforderlich. Die Bereitstellung von Testumgebungen dauert in der Praxis nicht selten mehrere Tage oder Wochen. Je nachdem, was für technische oder personelle Ressour- cen für die Testdurchführung benötigt werden, müssen diese eingeplant und deren Ver- fügbarkeiten zugesichert werden.

Beispiel

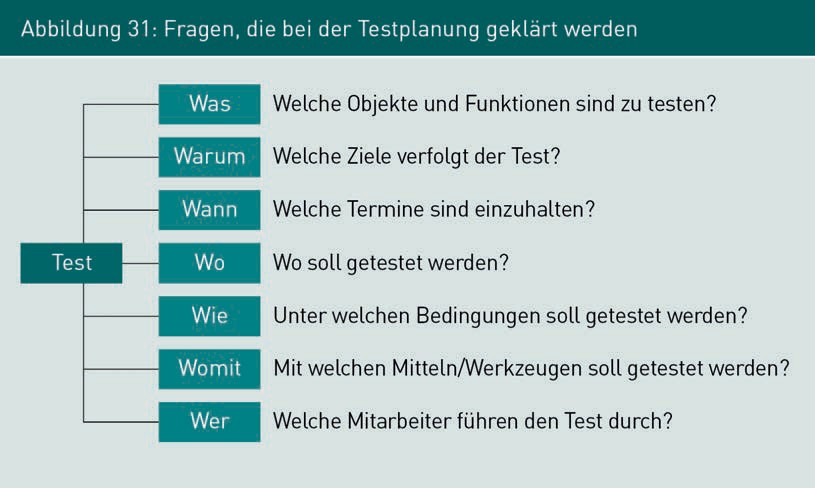
Häuﬁg richtet sich die Vergütung der Hardwarenutzung je nach Vertag mit dem Liefer- anten nach den tatsächlich für den Bedarf bereitgestellten Ressourcen (CPU, Speicher, Datentransfer). Werden Testfälle zeitgleich mit aufwendigen fachlichen Funktionen durchgeführt, kann das zu einer höheren Ressourcenauslastung und damit zu einer hohen Nachzahlung für die Bereitstellung der Infrastruktur führen. Wird das bei der Testplanung nicht mit berücksichtigt, können sich die Kosten für einen Testfall auf mehrere zehntausend Euro belaufen.

Beispiel

Für einen möglichst realitätsnahen Belastungstest wird eine große Anzahl gleichzeitiger Nutzer benötigt. Ist ein automatischer Test nicht möglich, müssen zu einem bestimm- ten Zeitpunkt möglichst viele Mitarbeiter zeitgleich vorher festgelegte Testfälle durch- führen. Dazu müssen zum einen alle Beteiligten die technischen Voraussetzungen für den Test erfüllen (beispielsweise die erforderliche Hard- und Software oder das Login auf dem System) und zum anderen müssen sie für die Durchführung des Tests geschult werden.

Im Zuge der Testplanung sind daher folgende Fragen zu beantworten:

Systematisches Testen von Software



###### Testfallspeziﬁkation

Im Zuge der Testfallspeziﬁkation werden die konkreten Testfälle erstellt und beschrie- ben und später ausgeführt. Bei der Testfallspeziﬁkation wird aufbauend auf den identi- ﬁzierten Testfallanforderungen eine detaillierte Beschreibung zu folgenden Aspekten erstellt:

* Zweck des Testfalls;
* Testgegenstände;
* Zustand des Systems vor und nach Testausführung;
* Art und Struktur der Testdaten;
* benötige Hard- und Softwareumgebung;
* benötigtes Personal sowie
* Abhängigkeiten zu anderen Testfällen.

Bei kleinen und fachlich einfachen Testfällen erfolgt die Testdatenerstellung häuﬁg zeitlich zusammen mit der Testfallspeziﬁkation. Bei komplexen Tests, wie beispiels- weise Systemtests, werden bei der Testfallspeziﬁkation Art und Struktur der Testdaten beschrieben, deren Erstellung erfolgt jedoch als separate Aktivität. Vor Beginn der Test- fallspeziﬁkation sollte festgelegt werden, auf welche Weise die Testfälle gewonnen wer- den und welcher Abdeckungsgrad mit den Testfällen erreicht werden soll.

Tabelle 16 enthält ein Beispieltemplate zur Speziﬁkation von Komponententests, das zur Speziﬁkation und Dokumentation von Testfällen verwendet werden kann.

|  |  |
| --- | --- |
| Tabelle 16: Beispieltemplate für Testfallspeziﬁkation | |
| Template für konkrete Testfälle | |
| ID des Testfalls | Eindeutige ID |
| Testgegenstand | Ofﬁzieller Name der zu testenden Systemkomponente oder der Schnittstelle oder der Funktion des Systems/der Komponente |
| Getestetes Detail | Name der Klasse(n) oder Komponente(n), die bei die- sem Test getestet werden |
| Funktion/en, die getes- tet werden | Detaillierter Name der Methode/der Funktionen/der Schnittstelle und deren Parameterliste |
| Kurzbeschreibung des Tests | Kurze Beschreibung, was mit dem Test geprüft bzw. sichergestellt werden soll |
| Testdaten/Eingabe | Welche Eingabedaten werden benötigt, um den Testfall durchzuführen? Mit welchen Parametern sollen die zu testenden Funktionen aufgerufen werden? Falls es sich um Objekte handelt: Welche Attribute sollen sie enthal- ten? Falls es sich um Dateien handelt: Welchen Inhalt haben sie? |
| Erwartetes Ergebnis/ Erwartete Systemreak- tion | Welche Ausgaben oder Systemreaktionen werden zu den Testdaten erwartet? |
| Vorbereitung/Vorbedin- gung | Konkrete Schritte, um das System/die Komponente für den Test vorzubereiten:   * Erzeugen oder Einspielen von Testdaten in die Datenbank * Aufrufen bestimmter Methoden, damit das System in den Zustand gelangt, in dem getestet werden kann (Bsp.: Anmelden von Nutzern am System) * Sicherstellen, dass bestimmte Daten in der Daten- bank vorliegen |

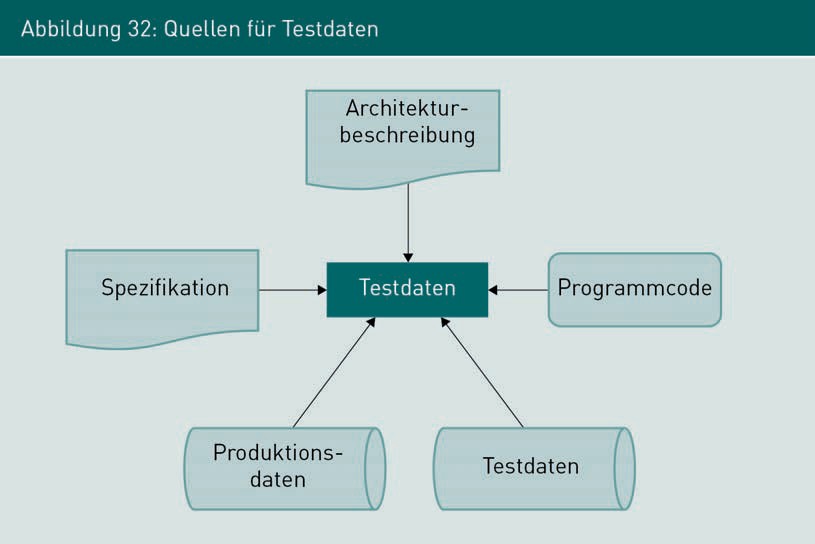
Systematisches Testen von Software

|  |  |
| --- | --- |
| Template für konkrete Testfälle | |
| ID des Testfalls | Eindeutige ID |
| Durchführung | Welche Systemfunktion(en)/Methode(n) muss/müssen beim Test ausgeführt werden?  Bei komplexeren Testfällen mit mehreren Aufrufen: In welcher Reihenfolge müssen die Methoden/System- funktionen ausgeführt werden? |
| Überprüfung/Nachbe- dingung | Geben Sie hier die konkreten Bedingungen eines bestandenen Tests an.  Welches Systemverhalten/welche Ausgaben werden vom System erwartet?  Welche Kriterien müssen die im System gespeicherten Daten erfüllen?  Bei der Erfüllung welcher Kriterien ist der Test erfolg- reich? Wie werden die Kriterien gemessen? |

###### Testdatenerstellung

Die Aktivitäten der Testdatenerstellung sind eng verbunden mit den Aktivitäten der Testfallspeziﬁkation und lassen sich häuﬁg nicht klar voneinander unterscheiden. Das Ergebnis der Testdatenerstellung sind Testdatensätze, die zur Durchführung der speziﬁ- zierten Testfälle benötigt werden. Wie viele Testdaten benötigt werden und welche Anforderungen es an die Testdaten gibt, wird in der Testfallspeziﬁkation festgelegt. Für einfache Komponenten oder Integrationstests reichen häuﬁg einfach strukturierte Test- daten aus. Für fachliche Ende-zu-Ende-Tests während eines Systemtests oder eines Abnahmetests sind häuﬁg komplexe Testdatensätze erforderlich, die ganz gezielt fachli- che Sonderbedingungen erfüllen müssen. Daher ist für die Testdatenerstellung in der Regel detailliertes Fachwissen erforderlich.

Verschiedene Techniken zur Testfallerstellung wurden bereits erläutert. Bei der Testda- tenerstellung ist darüber hinaus zu beachten, dass möglichst reale Testdaten verwen- det werden. Nur mit echten Daten lässt sich beispielsweise prüfen, ob die Größe von GUI-Elementen für die Ein- und Ausgabe von Daten ausreichend dimensioniert ist. Zur Erstellung von Testdaten können alle dem Tester zur Verfügung stehenden Quellen genutzt werden. In Abbildung 32 sind typische Quellen dargestellt: Neben der Speziﬁka- tion und Architekturbeschreibung eignen sich auch der Programmcode des Systems sowie bestehende Produktionsdaten oder der existierende Testdatenbestand.



Falls es abzusehen ist, dass die gleichen Testdaten in mehreren Tests verwendet wer- den sollen, kann sich die gezielte Erstellung und Wartung eines Testdatenbestands loh- nen. Dieser Bestand an Daten kann kontinuierlich um fachliche Spezialfälle erweitert werden. Er sollte auch Datensätze enthalten, deren Struktur erfahrungsgemäß immer Probleme bei der automatischen Verarbeitung bereitete. Mit einem gut gepﬂegten Test- datenbestand kann dafür gesorgt werden, dass typische Fehlerquellen schnell und zuverlässig erkannt werden.

###### Testausführung

Ticketsysteme Das (auch Bugtra- cker) sind Systeme zur Unterstützung der Dokumentation, Bewertung und Behebung von Soft-

warefehlern.

Die Ausführung von Softwaretests beinhaltet die Durchführung der speziﬁzierten Test- fälle unter Verwendung der Testdatensätze und der Protokollierung der Ergebnisse. Anhand der protokollierten Ergebnisse wird entschieden, ob ein für die aktuelle Pro- jektsituation akzeptabler Qualitätsgrad erreicht wurde. Wurden bei der Testdurchfüh- rung Fehler identiﬁziert, müssen diese in einer Mängelliste dokumentiert werden. Zu diesem Zweck werden in der Praxis sogenannte Ticketsysteme oder Bugtracker einge- setzt.

Die Durchführung der Tests erfolgt entweder automatisch oder manuell. Bei der manu- ellen Durchführung werden durch menschliche Nutzer die Vorbedingungen zum Start des Testfalls her- bzw. sichergestellt, der Test durchgeführt, die Nachbedingungen geprüft sowie das Systemverhalten protokolliert und bewertet.

Bei der automatischen Durchführung von Softwaretests wird die Testfallspeziﬁkation zusammen mit den Testdaten so beschrieben, dass diese von Testsystemen automa- tisch durchgeführt werden können. Automatische Tests sind grundsätzlich in allen Test-

Systematisches Testen von Software

stufen möglich. Das gilt für technische Tests, in denen Programmfunktionen entweder direkt oder über eine technische Schnittstelle aufgerufen werden. Aber auch GUI-Tests, bei denen Werte in eine Nutzeroberﬂäche eingegeben und deﬁnierte Interaktions- schritte ausgeführt werden, sind davon betroffen.

###### Testauswertung

Nach Abschluss der Ausführung werden die Testergebnisse ausgewertet und der Quali- tätsgrad des Systems beurteilt. Während die Auswertung beispielsweise von automati- sierten Komponententests mit Abschluss der Testdurchführung automatisch erstellt wird, müssen zur Auswertung von manuellen Systemtests mit mehreren Dutzend Testern die individuellen Ergebnisse erst zusammengetragen und aggregiert werden. Die Ergebnisse der Auswertungen bilden die Grundlagen für Entscheidungen über die weiteren Aktivitäten im Softwareprozess: Wurde der angestrebte Qualitätsgrad erreicht, kann beispielsweise das Quality Gate absolviert oder das System ausgeliefert werden. Wurde der angestrebte Qualitätsgrad nicht erreicht, müssen die erforderlichen Nachar- beiten durchgeführt werden.

### Komponententest (auch: Modultest, Unit-Test)

Um die Implementierung eines Softwaresystems durch mehrere Softwareentwickler zu ermöglichen, wird das System in seine logischen Bestandteile (auch: Komponenten, Module, Klassen, Bausteine) zerlegt. Jede Komponente wird separat implementiert und nach Fertigstellung in das System eingebracht (auch: integriert). Während bzw. nach der Fertigstellung einer Softwarekomponente kann diese bereits auf Einhaltung von speziﬁ- zierten Vorgaben überprüft werden. Das isolierte Prüfen von einzelnen Softwarebes- tandteilen wird Komponententest genannt.

Beispiel

* Prämienberechnung: Berechnung von Versicherungsprämien abhängig von den gewählten Leistungen;
* Warenkorb: Berechnung der Gesamtsumme aller Artikel;
* Adresskomponente: Erzeugen der richtigen Anschrift für den Briefverkehr aus den Kundendaten.

###### Testautomatisierung mit Unit-Tests

Für die meisten in der industriellen Softwareentwicklung eingesetzten Programmier- sprachen gibt es Frameworks, mit denen die automatisierte Durchführung und Auswer- tung von Unit-Tests unterstützt wird. Für die Programmiersprache Java beispielsweise gibt es das Framework JUnit. Unit-Tests werden von Entwicklern erstellt und durchge- führt, um die Qualität des selbst erstellen Programmcodes zu testen. Mit dieser Art von Tests wird die korrekte Ausführung von einzelnen Funktionen technischer Klassen und

Komponenten geprüft. Sie werden in der Regel bereits durchgeführt, bevor das Arbeits- ergebnis eines Entwicklers mit dem bereits bestehenden Programmcode zusammenge- führt wird.

Das Ziel, das mit dem Einsatz von automatischen Unit-Tests erreicht werden soll, ist eine möglichst große Testabdeckung der Anweisungen und Kontrollstrukturen im erstellten Programmcode. Unit-Tests sind White-Box-Tests, die je nach den konkreten Testfallanforderungen alle Anweisungen, Bedingungen oder Pfade einer Systemkompo- nente testen sollen. Von automatischen Unit-Tests wird häuﬁg gefordert, dass sie die Eigenschaften Reproduzierbarkeit, Unabhängigkeit und Regressionsfähigkeit haben.

Reproduzierbarkeit

Ein einmal erstellter Unit-Test sollte uneingeschränkt reproduzierbar sein. Das wird unter anderem auch durch das Verwenden von im Testfall fest deﬁnierten Testdaten erreicht, die nicht von der erfolgreichen Ausführung anderer Unit-Tests abhängen.

Unabhängigkeit

Darüber hinaus sollten Unit-Tests keine funktionalen Abhängigkeiten untereinander haben: Die Reihenfolge der Testausführung der Unit-Tests darf keine Rolle spielen. Jeder Test muss individuell dafür sorgen, dass die von ihm benötigten Vorbedingungen hergestellt werden. Dazu zählt beispielsweise auch, dass bei der Durchführung von Tests zuerst der erforderliche Datenbestand in die Datenbank eingespielt wird, auf dem dann der Test durchgeführt wird. Wird anschließend ein weiterer Test durchgeführt, müssen alle Änderungen am Datenmodell und dem internen Systemzustand wieder rückgängig gemacht werden, sodass für die Ausführung eines neuen Tests der ursprüngliche Systemzustand wiederhergestellt wird.

Regressionsfähigkeit

Insbesondere in Softwareprozessen, in denen Systeme evolutionär entwickelt werden, also dem System schrittweise neue Funktionen hinzugefügt werden, muss vor dem Release einer neuen Systemversion am Ende einer Iteration der erforderliche Quali- tätsgrad sichergestellt sein. Für die Unit-Tests bedeutet das, dass alle Unit-Tests, die bereits erfolgreich ausgeführt werden konnten, auch nach einer Überarbeitung erfolg- reich durchlaufen werden müssen. Die Durchführung bereits bestandener Tests nach Änderungen am Programmcode werden Regressionstests genannt. Am Ende einer Itera- tion müssen also alle bisher erstellten und alle neuen Unit-Tests erfolgreich durchlau- fen werden. Das ist der Grund, warum die Testautomatisierung eine wichtige Vorausset- zung für die evolutionäre Softwareentwicklung ist.

###### Vorgehensweise: Test Driven Development

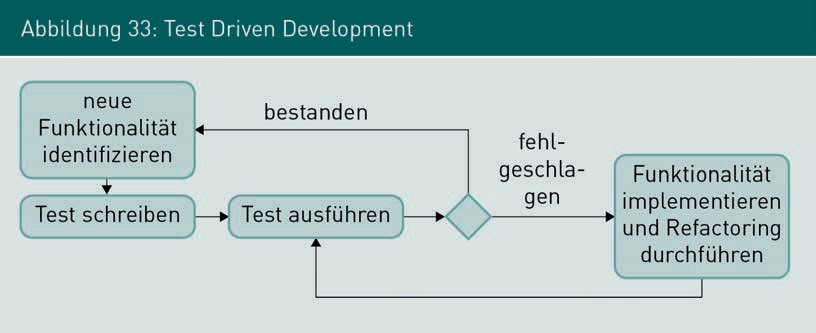
Eine Vorgehensweise, die eine enge Verzahnung der Implementierung von Programm- code und die Erstellung von Unit-Tests zum Testen des Programmcodes fokussiert, ist das sogenannte Test Driven Development. Mit Test Driven Development soll eine hohe Qualität und eine hohe Abdeckung des erstellten Programmcodes durch Unit-Tests

Systematisches Testen von Software

sichergestellt werden, in dem die Aktivitäten „Testfall erstellen“ und „Programmcode erzeugen“ genau in der umgekehrten Reihenfolge als bei der „klassischen“ Testfaller- stellung durchgeführt werden.

In Abbildung 33 ist die grundsätzliche Vorgehensweise im Test Driven Development dar- gestellt.

Der entscheidende Unterschied zur herkömmlichen Entwicklung von Testfällen ist die Erstellung von Unit-Tests zu Funktionen, bevor deren Methodenrumpf programmiert wird. Nachdem die Notwendigkeit einer neuen Funktion in einer Klasse oder einer Komponente identiﬁziert wurde, werden zuerst Unit-Tests erstellt, mit denen die neu zu erstellende Funktion getestet werden soll. Werden diese Unit-Tests ausgeführt, müssen sie zunächst fehlschlagen, da die relevanten Systemfunktionen noch nicht umgesetzt werden. Mit ihrer Umsetzung wird erst nach der Testfallerstellung begonnen. Die Sys- temfunktionen werden nun schrittweise implementiert und bereits während der Imple- mentierung getestet.



Wird bei der Implementierung festgestellt, dass weitere Funktionen benötigt werden, werden auch hier zuerst wieder Testfälle erstellt, bevor die eigentlichen Funktionen implementiert werden. In der praktischen Anwendung wechseln sich die Aktivitäten Testen und Implementieren in kurzen Zyklen ab, sodass ein Zyklus oft nur wenige Minu- ten dauert. Wird der Test erstellt, bevor der Programmcode implementiert wird, muss sich der Entwickler zuerst intensiv mit der zu erstellenden Funktion auseinandersetzen, bevor die erste Zeile Programmcode geschrieben wurde. Mit diesem Zwang zum vorhe- rigen Durchdenken der Fachlichkeit und möglicher Fehlersituationen wird ein Bewusst- sein für möglicherweise kritische und fehleranfällige Bereiche der Funktionen geschaf- fen, die dann bei der Implementierung direkt mit berücksichtigt werden können.

Test Driven Develop- ment

Das ist eine Vorge- hensweise zur Erstel- lung von Unit-Tests, bei der die Testfälle bereits vor dem zu testenden Pro- grammcode imple- mentiert werden.

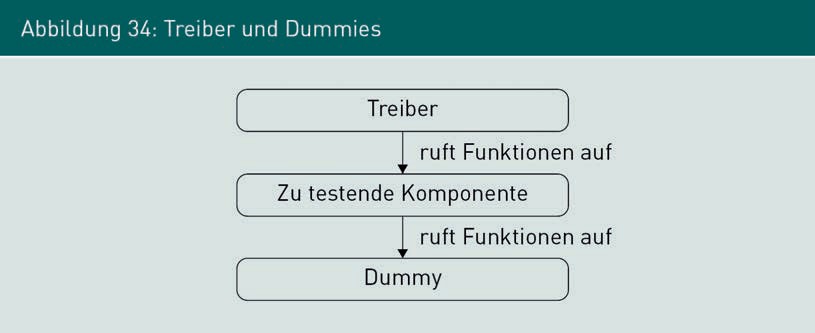
### Integrationstests

Sobald mindestens zwei Softwarekomponenten fertiggestellt wurden, können sie zu einem System zusammengeführt, also integriert werden. Während bzw. nach der Integ- ration wird in Integrationstests das Zusammenspiel von Gruppen von Komponenten getestet und geprüft, ob die Komponenten so zusammenarbeiten, wie es in der Speziﬁ- kation beschrieben wurde.

Beispiele

* + Integration eines externen Online-Bezahlsystems an einen Onlineshop;
  + Integration der Bestellkomponente eines Lieferanten in einem Onlineshop;
  + Integration einer App für mobile Plattformen, um zusätzlich zur Weboberﬂäche auch Bestellungen von mobilen Endgeräten zu ermöglichen.

Im Vergleich zum Komponententest soll bei Integrationstests nicht das interne Verhal- ten einzelner Komponenten intensiv geprüft werden, sondern das Zusammenspiel der Komponenten über deren technische Schnittstellen. Relevante Aspekte dabei sind zum einen die durch die Schnittstelle bereitgestellten Funktionen und zum anderen die über die Schnittstelle ausgetauschten Daten. Für Integrationstests müssen daher gezielt solche Testfälle erstellt und durchgeführt werden, mit denen die durch die Schnittstellen bereitgestellten Funktionen möglichst umfangreich geprüft werden. Dazu zählt insbesondere auch das Verhalten von Schnittstellen bei fehlerhaften Aufrufen und fehlerhaft übermittelten Daten. Abgesehen vom Fokus der Testfälle auf die Schnitt- stellen, unterscheiden sich Integrationstests von ihrer Art der Erstellung oder Durch- führung nicht grundsätzlich von Komponententests. Beispielsweise ist Test Driven Development auch für Schnittstellen zwischen Systemkomponenten möglich.

Treiber Das sind Softwaref- ragmente, die das Aufrufen anderer Komponenten simu-

lieren.

Dummies Das sind Softwaref- ragmente, die Kom- ponenten simulieren und durch andere Komponenten aufge-

rufen werden.

Insbesondere bei komplexen Softwaresystemen mit sehr vielen Komponenten sind nicht alle Komponenten zur gleichen Zeit fertig. In diesen Fällen werden fehlende Kom- ponenten durch sogenannte Treiber und Dummies simuliert, damit bereits mit den Test begonnen werden kann. Abbildung 34 illustriert das Zusammenspiel von Kompo- nenten, Treibern und Dummies.

Systematisches Testen von Software

Als Treiber werden dabei die Softwarefragmente bezeichnet, die das Aufrufen anderer Komponenten simulieren. Dummies hingegen simulieren Komponenten, die durch andere Komponenten aufgerufen werden. Insbesondere technische Schnittstellen zu externen Systemen werden im Integrationstest durch Dummies und Treiber simuliert. Erst das fertige Softwaresystem kann an externe Systeme angeschlossen werden, das Zusammenspiel der Schnittstellen muss jedoch bereits vorher getestet werden.

Mithilfe der Integrationsstrategie wird für ein Projekt festgelegt, in welcher Reihenfolge die einzelnen Komponenten integriert und getestet werden. Im Folgenden werden dazu die drei verschiedenen Integrationsstrategien Bottom-up, Top-down und By-Value vor- gestellt.

###### Bottom-up-Strategie

Bei der Bottom-up-Strategie werden die Basiskomponenten zuerst implementiert. Dabei werden intensiv Treiber eingesetzt. Aufbauend auf den Basiskomponenten wer- den anschließend die Komponenten implementiert, die auf Basisdienste zugreifen. Abbildung 35 stellt das Grundprinzip der Bottom-up-Strategie von links nach rechts gelesen dar: Zuerst werden die Komponenten 3 und 2 erstellt, die später von Kompo- nente 1 aufgerufen werden (t1). Die Eingabe der Testdaten bzw. das Aufrufen der kon- kret zu testenden Funktionen dieser Komponenten erfolgt durch Treiber. Anschließend wird Komponente 1 erstellt (t2), die die Funktionen von Komponente 3 und 2 benötigt und selbst von einem Treiber aufgerufen wird. Zuletzt werden mit der Nutzerschnitt- stelle (3) die Funktionen umgesetzt, welche die Komponente 1 aufrufen. Erst dann sind alle Komponenten implementiert und können vollständig ohne den Einsatz von Trei- bern getestet werden.

Bei der Bottom-up-Strategie beginnt die Systementwicklung in der untersten System- ebene und wird ebenenweise sukzessiv bis zur obersten Systemebene durchgeführt. Dabei werden die Treiber Stück für Stück durch die fertiggestellten echten Komponen- ten ersetzt.

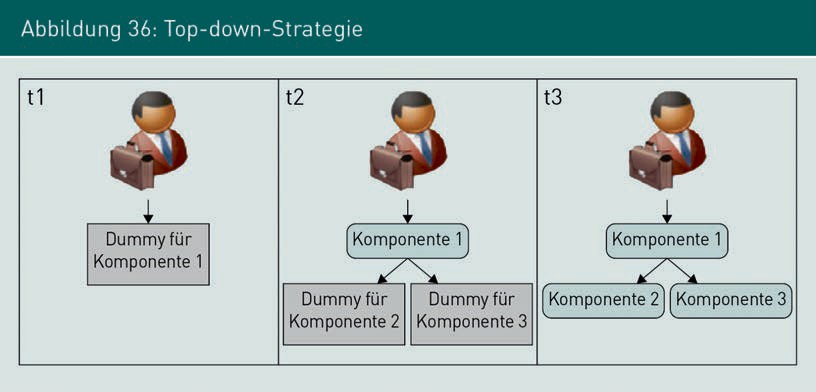
Basiskomponenten Das sind Komponen- ten, die innerhalb eines Systems grundlegende Dienste anbieten und von anderen Komponenten aufge- rufen werden.



Ein Vorteil der Bottom-up-Strategie ist die frühzeitige Prüfung von Basiskomponenten, die aufgrund ihrer Nähe zur Hardware und zur Infrastruktur bezogen auf die eingesetz- ten Technologien sehr heterogen ist. Darüber hinaus können durch den Einsatz von Treibern die erstellten Testdaten gezielt und direkt eingegeben werden. Diesen Vortei- len stehen jedoch als Nachteile gegenüber, dass den Anwendern erst am Ende der Integration das System präsentiert werden kann und damit ein wesentliches Projektri- siko nicht frühzeitig adressiert werden kann. Weiterhin können keine gezielten falschen Rückgabewerte von aufgerufenen Komponenten getestet werden, da bei der Bottom- up-Strategie keine Dummies zum Einsatz kommen.

###### Top-down-Strategie

Beim Einsatz der Top-down-Strategie beginnt die Integration der Systemkomponenten an der obersten Systemebene und endet mit der Integration der Basiskomponenten. Abbildung 36 zeigt exemplarisch das Grundprinzip der Top-down-Strategie: Begonnen wird mit der Umsetzung und dem Test der obersten Systemebene (t1), in diesem Fall der Nutzerschnittstelle. Für die Durchführung der Tests wird ein Dummy der Kompo- nente 1 benötigt, der deren Funktionen simuliert. Anschließend wird mit der Kompo- nente 1 die nächste Systemebene implementiert (t2) und in das System integriert. Dazu wird der Dummy der Komponente 1 entfernt und die Dummies für die Komponenten 2 und 3 erstellt, die von Komponente 1 benötigt werden. Zuletzt werden im Rahmen einer Top-down-Strategie die Basiskomponenten implementiert und mit deren Integration in das Gesamtsystem die verbleibenden Dummies entfernt.



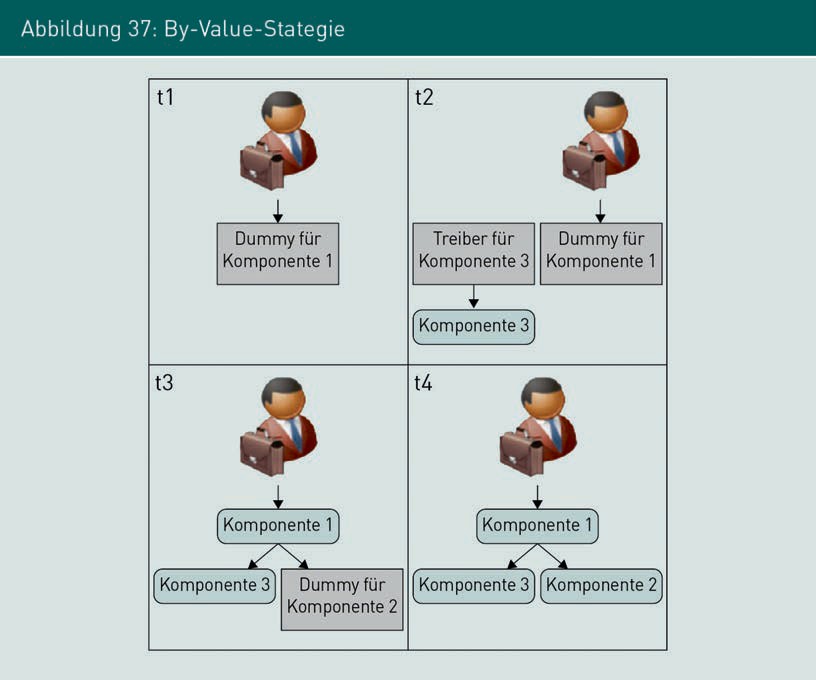
Die Vorteile der Top-down-Strategie sind eine frühzeitige Verfügbarkeit der System- ebene, mit denen die Anwender arbeiten. Daher können somit zur Steigerung der Anwenderzufriedenheit recht schnell erste Erfahrungen und Verbesserungswünsche gesammelt werden. Darüber hinaus lassen sich fehlerhafte Rückgabewerte von aufge- rufenen Basiskomponenten leicht simulieren, da diese direkt durch die Dummies erzeugt werden können. Als Nachteile der Top-down-Strategie ist allerdings die sehr späte Integration von Basiskomponenten zu nennen, die häuﬁg verschiedene Technolo-

Systematisches Testen von Software

gierisiken bergen. Außerdem können im Vergleich zur Bottom-up-Strategie die Testda- ten nicht so gezielt eingegeben werden, da bei der Top-down-Strategie keine Treiber eingesetzt werden.

###### By-Value-Strategie

Bei der in diesem Skript By-Value-Strategie genannten Integrationsstrategie erfolgt die Integration der Komponenten nach Aspekten der Wertorientierung einzelner Kompo- nenten. Im Vergleich zur Top-down-Strategie oder Bottom-up-Strategie orientiert sich die By-Value-Strategie nicht strikt entlang der technischen Ebene der Komponenten im Gesamtsystem, sondern an den tatsächlichen Bedürfnissen des Softwareprojektes. Die Reihenfolge der Integration der Komponenten in das Gesamtsystem wird projektabhän- gig festgelegt. Begonnen wird dabei mit den Komponenten, mit denen der größte Wert- beitrag geliefert werden kann. Abbildung 37 veranschaulicht beispielhaft den Einsatz der By-Value-Strategie: Zuerst wird die Nutzeroberﬂäche umgesetzt (t1), damit den Sta- keholdern recht schnell konkrete Systemartefakte gezeigt werden können. Dazu wird ein Dummy von Komponente 1 benötigt. Anschließend wird Komponente 3 umgesetzt (t2), deren Realisierung ein hohes technisches Risiko hat. Hierfür wird ein Treiber für Komponente 3 benötigt. Anschließend wird mit Komponente 1 die Zusammenschaltung von GUI und Komponente 3 realisiert (t3), wozu noch ein Dummy von Komponente 2 benötigt wird. Zuletzt wird Komponente 2 umgesetzt und damit das Gesamtsystem fer- tiggestellt.



Damit handelt es sich bei der By-Value-Strategie um eine Mischform, die sowohl Trei- ber als auch Dummies für die Durchführung der Komponententests benötigt.

Die Vorteile der By-Value-Strategie liegen in der Möglichkeit, abhängig von der aktuel- len Projektsituation entscheiden zu können, welche Komponenten integriert und getes- tet werden sollen. Mit dem Einsatz von Treibern und Dummies sind sowohl gezielte Ein- gaben von Testdaten als auch die gezielte Simulation fehlerhafter Rückgabewerte möglich. Nachteil der By-Value-Strategie ist der Einsatz sowohl von Treibern als auch von Dummies, womit der Aufwand zur Erstellung von reinen Testartefakten im Vergleich zu anderen Integrationsstrategien gegebenenfalls höher ist.

###### Diskussion der verschiedenen Integrationsstrategien

Für die Entwicklung von industriellen Informationssystemen stellen die Strategien Bot- tom-up und Top-down in der Regel Idealmuster dar, die in ihrer Reinform praktisch so keine Anwendung ﬁnden. Es werden in der Praxis die einzelnen Komponenten typi- scherweise zu unterschiedlichen Zeitpunkten fertiggestellt, die teilweise sehr deutlich von den ursprünglich geplanten Zeitpunkten abweichen. Die strenge Vorgabe der Integ- rationsreihenfolge hat zur Konsequenz, dass auf projektspeziﬁsche Risiken und Ent- wicklungen nicht gezielt reagiert werden kann. Die hier By-Value genannte Integrations- strategie bietet dem Projektteam die entsprechenden Freiräume je nach aktuellem

Systematisches Testen von Software

Bedarf, auch auf geänderte Situationen angemessen zu reagieren: Falls zum Testen des Zusammenspiels von Komponenten sowohl Treiber als auch Dummies benötigt werden, können diese bei Bedarf erstellt und eingesetzt werden.

###### Einsatz von Dummy-Komponenten

Zur Unterstützung der Durchführung von Integrationstests kann von dem Entwicklungs- team, das für die Entwicklung einer Komponente verantwortlich ist, die gleichzeitige Erstellung und Pﬂege eines Dummies dieser Komponente gefordert werden. Dieser Dummy stellt alle von der Komponente benötigten Schnittstellen und Funktionen zur Verfügung, die für Testzwecke eingesetzt werden können. Andere Entwicklungsteams, die Funktionen dieser Komponente benötigen oder ihre Funktionen mit der Kompo- nente zu Testzwecken aufrufen müssen, können jederzeit die Dummy-Komponente ein- setzen. Durch die Verantwortung zur Pﬂege einer Dummy-Komponente ist sicherge- stellt, dass alle ihre Funktionen dem aktuellen Entwicklungsstand entsprechen und Änderungen an der Komponentenschnittstelle auch direkt im Dummy der Komponente gepﬂegt werden. Abbildung 38 veranschaulicht den Einsatz von Dummy-Komponenten. Da sie alle Schnittstellenfunktionen simulieren können, können sie bei Integrations- tests sowohl als Treiber als auch als Dummies eingesetzt werden.



Für die Durchführung von Integrationstests muss zu jeder Komponente in jedem Fall ein Dummy erstellt werden. Es ist zu jedem Zeitpunkt sichergestellt, dass dieser eine Dummy sich genauso verhält wie die echte Komponente. So muss nicht jedes Entwick- lungsteam erneut die Testinfrastruktur für die Tests seiner Komponenten aufbauen, sondern kann die von den Entwicklern der Komponente bereitgestellten Dummies ein- setzen. Mit der Lieferung von Dummy und echter Komponente aus einer Hand kann eine hohe Testqualität sichergestellt werden.

### Systemtests

Performancetest Dabei wird das Ver- halten des Systems unter Belastung auf Einhaltung der spe- ziﬁzierten Qualitäts- eigenschaften und der Messung der Leistungsfähigkeit

geprüft.

Lasttests Diese testen das Systemverhalten unter besonders hohen Anforderun- gen, bspw. durch das gezielte Entziehen von Ressourcen.

Nach Abschluss der Entwicklungsarbeiten und der Integration aller Softwarebestand- teile zu einem fertigen System wird mit dem sogenannten Systemtest das System als Ganzes getestet. Ziel des Systemtests ist die Überprüfung, ob das System als Ganzes die speziﬁzierten Anforderungen erfüllt. Neben Funktionstests werden dabei auch Perfor- mancetests und Lasttests durchgeführt. Eine große Herausforderung beim Systemtest ist die möglichst originalgetreue Nachbildung der Produktivumgebung des Kunden. Dazu zählen neben der …

* … Hardware und Softwareumgebung, in der das System betrieben wird, die
* Softwaresysteme, zu denen das zu testende System technische Schnittstellen hat, eine
* realistische Auslastung und realistisches Nutzerverhalten ebenso wie
* möglichst echte Datensätze.

Wobei insbesondere die Bereitstellung von originalgetreuen Datensätzen, die gleichzei- tig den Vorschriften des Datenschutzes entsprechen, ein großes und oft unlösbares Problem darstellt.

###### Funktionstests

Ziel der Funktionstests im Rahmen des Systemtests ist die Überprüfung der funktiona- len Anforderungen an das Gesamtsystem. Im Unterschied zu den Integrationstest, in denen bereits die Zusammenschaltung der einzelnen Systemkomponenten getestet wurde, liegt der Fokus beim Systemtest auf der Sicht der Anwender. Das System wird dabei vollständig als Black Box betrachtet. Mit den Funktionstests wird geprüft, ob die Systemunterstützung von Geschäftsprozessen und Fachfunktionen genau so implemen- tiert wurden, wie durch die Speziﬁkation vorgegeben. Die Durchführung der Testfälle erfolgt auf Dateneingaben in der Nutzeroberﬂäche oder über das Aufrufen der System- schnittstellen, durch die das System mit seinen Umsystemen verbunden ist. Dabei soll- ten auch die Umsysteme keine Dummies oder Treiber sein, sondern Systeme in der Version, wie sie in der zukünftigen Ausführungsumgebung des Systems vorliegen wer- den. Die Auswertung der Testfälle erfolgt anhand des nach außen sichtbaren System- verhaltens. Dazu zählen unter anderem die Ausgaben über die Nutzerschnittstelle, die erzeugten Dokumente, die zurückgelieferten Nachrichten an Systemschnittstellen, der geänderte Datenbestand einer Datenbank oder vom System erzeugte Logdateien.

###### Performancetest (auch: Leistungstest)

Mithilfe von Performancetests wird das Verhalten des Systems unter Belastung geprüft und dabei ermittelt, ob es die speziﬁzierten Qualitätseigenschaften unterstützt. Typi- sche Szenarien eines Performancetests sind das gleichzeitige Arbeiten vieler Nutzer mit dem System oder die Verarbeitung großer fachlicher Datenmengen. Mit Ergebnissen des Performancetests können konkrete Aussagen über das Systemverhalten des erstell-

Systematisches Testen von Software

ten Systems in seiner zukünftigen Systemumgebung gemacht werden. Grundsätzlich dient ein Performancetest der Risikominimierung bei Inbetriebnahme des Systems, da er Aussagen zum Systemverhalten in der späteren Ausführungsumgebung ermöglicht. Denn erst wenn alle technischen Komponenten integriert sind und das System auf der Hardware installiert ist, auf der es später auch im Betrieb laufen soll, lassen sich zuver- lässige Aussagen dazu machen. Wichtig hierbei ist die genaue Nachbildung der Hard- ware- und Softwareumgebung, da bereits kleinste Veränderungen das Ergebnis von Per- formancetests verzerren können. Auf Basis der Testergebnisse kann auch überprüft werden, ob die vom IT-Betrieb zur Verfügung gestellte IT-Infrastruktur wie Speicher- platz, Arbeitsspeicher, Rechenzeit oder Netzwerkanbindung ausreichen. Performance- tests müssen daher sehr sorgfältig vorbereitet werden.

Neben der Verfügbarkeit der benötigten IT-Infrastruktur muss das System in die Tes- tumgebung installiert und an alle Umsysteme angeschlossen werden. Hierzu sind alle erforderlichen Aktivitäten zur Inbetriebnahme des Systems notwendig. Daher muss auch auf die Verfügbarkeit des entsprechenden Personals geachtet werden.

Typische Indikatoren für die Leistungsfähigkeit von Softwaresystemen, die bei einem Performancetest gemessen werden, sind unter anderem (Grechenig 2010, S. 328):

* + Latenz: Zeitspanne, die von der Anfrage an das System bis zur Beantwortung ver- geht; beispielsweise die Zeit, die das System bei der Bestellung in einem Online- shop vom Klick auf den OK-Button bis zur Anzeige des Bestätigungsdialogs benötigt;
  + Durchsatz: Verarbeitungsgeschwindigkeit von Funktionen oder Daten, beispielsweise die Anzahl der verarbeiteten Bestellvorgänge pro Minute;
  + Transaktionsrate: Maß zur Verarbeitungsgeschwindigkeit von deﬁnierten, komplexen Änderungen am Datenbestand, beispielsweise die Anzahl der gebuchten Überwei- sungen im Kernsystem einer Direktbank.

Der Performancetest dient auch zur Bestimmung bzw. zur Prüfung der Servicelevel, die ein Systembetreiber im Rahmen von Serviceverträgen in den sogenannten Service Level Agreements (SLA) zugesichert hat. So werden in Service Level Agreements die Dienstgü- teeigenschaften wie Antwortzeiten oder Verfügbarkeiten zu der bereitgestellten Soft- ware fest vereinbart.

Insbesondere für Funktionen der Dunkelverarbeitung, also von fachlichen Funktionen, die in der Regel in den Nachtstunden automatisch große Datenmengen verarbeiten, muss die Einhaltung genau speziﬁzierter Zeitfenster zugesichert werden. Beispielsweise muss das Überweisungssystem einer Bank in der Lage sein, alle tagsüber eingegebenen Überweisungen in den Nachtstunden und vor Beginn des nächsten Arbeitstages zu buchen. Auch wenn es sich dabei um mehrere Millionen Transaktionen handelt, muss sichergestellt werden, dass der Buchungsvorgang pünktlich abgeschlossen wird.

###### Lasttest (auch: Stresstest)

Der Lasttest eines Systems dient wie der Performancetest ebenfalls zur Bestimmung des Systemverhaltens in seiner Zielumgebung. Das Ziel ist jedoch explizit das Testen des Systemverhaltens unter besonders hohen Anforderungen an die zur Verfügung ste- henden Ressourcen. Das kann entweder durch das gezielte Entziehen von Ressourcen geschehen, oder durch das künstliche Erzeugen von massiven Datenmengen oder Funk- tionsaufrufen. Beispiele für den Ressourcenentzug sind das Wegnehmen von verfügba- rem Arbeits- oder Festplattenspeicher, die Reduzierung von Netzwerkbandbreite oder die Senkung der Verarbeitungsgeschwindigkeit des Datenbanksystems. Beispiele für die Lasterzeugung sind das gleichzeitige Anmelden und Arbeiten extrem vieler Nutzer, die Eingabe von sehr großen Datenmengen oder das Anstoßen von sehr vielen Transaktio- nen. Mit der Durchführung von Lasttests wird beobachtet, wie sich das System im Fall einer Überlast verhält, ob es dabei zum Verlust oder zu Inkonsistenzen von Daten kommt und ob sich das System bei dem Hinzufügen von Ressourcen wieder stabilisiert. Aus dem beobachteten Verhalten lassen sich häuﬁg auch Indikatoren ableiten, die für das Monitoring der Anwendungen im späteren Betrieb eingesetzt werden können.

Häuﬁg auch schon für Performancetest, jedoch spätestens bei Lasttests ist eine Werk- zeugunterstützung unbedingt erforderlich. Insbesondere die Simulation von sehr vielen gleichzeitigen Nutzern kann ohne den Einsatz von Testwerkzeugen nicht durchgeführt werden. Bei der Konzeption und Planung von Lasttests ist daher auch die Einrichtung von Testwerkzeugen zu berücksichtigen, die je nach Komplexität der Anwendung selbst wieder kleinere Softwareprojekte sind.

###### Weitere Testarten

Für die Durchführung von Systemtests lassen sich zusätzlich noch die folgenden Testar- ten identiﬁzieren (Balzert 1998, S. 539–542):

* + - Sicherheitstest: Ausprobieren typischer Angriffsszenarien und Sicherstellung, dass das System die geforderten Sicherheitseigenschaften erfüllt;
    - Usability-Test: Methodische Überprüfung der Gebrauchstauglichkeit des Systems hinsichtlich der speziﬁzierten Anforderungen an die Benutzbarkeit, die beispiels- weise als Experiment mit Probanden der späteren Nutzergruppe durchgeführt wird;
    - Wiederinbetriebnahmetest: Prüfen, ob sich das System nach einem Ausfall wieder in den ursprünglichen Betriebszustand überführen lässt. Hierzu zählen insbeson- dere auch das Einspielen von Backups und die Überprüfung, ob der ursprüngliche Systemzustand inklusive aller Anwendungsdaten wiederhergestellt werden kann.

###### Regressionsfähigkeit von Systemtests

Mit dem Begriff Regressionstests werden Testfälle bezeichnet, die bereits erfolgreich durchgeführt worden sind und nach einer Anpassung des Systems erneut durchgeführt werden. Die Eigenschaft „Regressionsfähigkeit von Testfällen“ bezeichnet daher die

Systematisches Testen von Software

Fähigkeit zur wiederholten Durchführung des Tests. In der Regel sollten alle erstellten Testfälle regressionsfähig sein. Das gilt insbesondere für Testfälle in evolutionären Soft- wareprozessen, in denen das Softwaresystem in mehreren Iterationen entwickelt wird. Beispielsweise zeigt Abbildung 7 einen Softwareprozess, bei dem jede Iteration mit einem Systemtest abgeschlossen wird. Das bedeutet, am Ende einer jeden Iteration wird jedes Mal das Gesamtsystem getestet. Dabei werden jedoch nicht nur die in der Iteration jeweils neu hinzugefügten Funktionen getestet, sondern auch alle vorher bereits umgesetzten Funktionen. Damit wird sichergestellt, dass durch Anpassungen am Programmcode bereits bestehende Funktionen nicht versehentlich beeinﬂusst werden. Aber auch falls im Projekt nicht evolutionär entwickelt wird, müssen bei jedem neuen Release der Software die bereits bestehenden Systemtestfälle durchgeführt werden, selbst wenn die Release-Zyklen mehrere Monate oder Jahre dauern.

Die Fähigkeit alle bereits bestehenden Testfälle kontinuierlich zu wiederholen, trägt daher maßgeblich zur Erreichung eines kontinuierlich hohen Qualitätsgrades bei. Daher sollte bei der Konzeption und Durchführung der Testfälle weitestgehend auf die Auto- matisierung von Tests wertgelegt werden. Falls das nicht möglich ist, müssen die Test- fälle so konzipiert und dokumentiert werden, dass sie jederzeit auch von bis dahin noch nicht mit am Projekt beteiligten Personen durchgeführt werden können.

Mittlerweile sind auch für die Automatisierung von GUI-Tests verschiedene Werkzeuge und Frameworks erhältlich und werden in der Praxis eingesetzt.

### Abnahmetests

Die letzte Teststufe ist der Abnahmetest (auch: Akzeptanztest). Hier wird das fertige Sys- tem beim Auftraggeber installiert und unter den tatsächlichen Betriebsbedingungen getestet. Beim Abnahmetest wird geprüft, ob das System aus Kundensicht die vertrag- lich vereinbarten Leistungsmerkmale aufweist. Der Auftraggeber führt den Abnahme- test selbst durch oder wird in die Durchführung miteinbezogen. Von der Art der Durch- führung ist der Abnahmetest eine spezielle Art des Systemtests, eine trennscharfe Abgrenzung ist dabei in der Regel schwierig. Der wichtigste Unterschied gegenüber dem Systemtest ist jedoch die Durchführung durch eine andere Organisation als die, die das System entwickelt hat. Ein erfolgreicher Abnahmetest ist häuﬁg die vertraglich verein- barte Voraussetzung für die Rechnungslegung. Im Unterschied zum Systemtest wird der Abnahmetest typischerweise, wie auch im Beispiel in Abbildung 7 dargestellt, nur ein- mal im Softwareprozess durchgeführt. Wohingegen Systemtests häuﬁg nach jeder Itera- tion oder jedem Meilenstein durgeführt werden.

###### Verhandlung des Testergebnis

Während die bisherigen Teststufen in der Regel durch den Systemhersteller ausgeführt werden, liegt der Abnahmetest in der Verantwortung des Kunden, der die Erstellung des Systems in Auftrag gegeben hat. Daher orientieren sich die Testfälle des Abnahmetests direkt an den zwischen Auftraggeber und Auftragnehmer vereinbarten Abnahmekrite-

rien. Im Unterschied zu den anderen Teststufen steht am Ende des Abnahmetests die Entscheidung: Annahme oder Ablehnung. In der Praxis ist es dabei sehr unwahrschein- lich, dass während der Abnahmetests keine Fehler mehr identiﬁziert werden. Daher wird nach der Durchführung des Abnahmetests in der Regel verhandelt, welche Fehler zur endgültigen Abnahme noch behoben werden müssen. Erst wenn diese Fehler beho- ben wurden, wird ofﬁziell die Abnahme erteilt.

Zusammenfassung

Das Testen von Software erfolgt je nach aktuellem Projektfortschritt in unterschied- lichen Teststufen: Komponententest, Integrationstest, Systemtest und Abnahmetest. Zur methodischen Durchführung von Softwaretests müssen die Aktivitäten Testan- forderungsanalyse, Testplanung, Testfallspeziﬁkation, Testdatenerstellung, Testaus- führung und Testauswertung unabhängig von der Teststufe durchgeführt werden.

Das isolierte Prüfen von einzelnen Softwarebestandteilen wird Komponententest genannt. Diese Tests werden durch automatische Unit-Tests und Test Driven Deve- lopment unterstützt. Die Integrationstests prüfen, ob die Komponenten so zusam- menarbeiten, wie es in der Speziﬁkation beschrieben wurde. Je nach Integrations- strategie werden hierzu Treiber und Dummies eingesetzt. Ziel des Systemtests ist die Überprüfung, ob das System als Ganzes die speziﬁzierten Anforderungen erfüllt. Neben Funktionstests werden dabei unter anderem auch Performancetests und Lasttests durchgeführt. Beim Abnahmetest wird geprüft, ob das System aus Kun- densicht die vertraglich vereinbarten Leistungsmerkmale aufweist. Ein erfolgreicher Abnahmetest ist häuﬁg die vertraglich vereinbarte Voraussetzung für die Rech- nungslegung.



# Lektion 7

## Systematische Qualitätssicherung von Anforderungen, Architekturen und Prozessen

#### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie wissen, …

… welche Aktivitäten zur Qualitätssicherung von Anforderungen erforderlich sind.

… wann im Softwareprozess die Qualität von Architekturen sichergestellt werden kann.

… welche Qualitätsattribute und Reifegrade für Softwareprozesse bestimmt werden können und wie Prozesse verbessert werden können.

DL-D-IQSS01-L07

1. Systematische Qualitätssicherung von Anforderungen, Architekturen und Prozes- sen

### Einführung

Bei den Aktivitäten zur Qualitätssicherung im Softwareprozess steht häuﬁg das erzeugte Softwaresystem im Vordergrund. Allerdings müssen auch die im Softwareprozess doku- mentierten Anforderungen und Architekturdeﬁnitionen geprüft werden. Dazu werden in dieser Lektion geeignete Vorgehensweisen vorgestellt. Darüber hinaus kann auch die Qualität des Softwareprozesses bewertet und verbessert werden.

### Qualitätssicherung von Anforderungen

Die analytische Qualitätssicherung von Anforderungen wird in der Kernaktivität Prüfen und Abstimmen aus dem Requirements Engineering durchgeführt. Das Ziel der Aktivitä- ten zum Prüfen und Abstimmen ist die Freigabe der Anforderungen zur Umsetzung in den nachgelagerten Phasen des Softwareprojekts. Im Beispielprozess in Abbildung 7 erfolgt die analytische Qualitätssicherung von Anforderungen nach der Ermittlung der fachlichen Anforderungen und anschließend nach Erstellung der technischen Speziﬁka- tion. So soll sichergestellt werden, dass vor der Weiterverarbeitung der ermittelten und dokumentierten Anforderungen deren Qualität gesichert wird. Das ist insbesondere notwendig, weil alle weiteren Aktivitäten im Softwareentwicklungsprozess direkt oder indirekt von den dokumentierten Anforderungen abhängig sind. Jeder Fehler darin, der nicht frühzeitig erkannt wird, pﬂanzt sich über die technische Speziﬁkation und Archi- tektur bis hin zum Programmcode fort.

Zusätzlich zur Prüfung von Anforderungen hinsichtlich deren fachlicher und handwer- klicher Qualität ist es jedoch insbesondere für fachliche Anforderungen wichtig, dass sie mit allen relevanten Stakeholdern abgestimmt sind. Die Menge der zur Umsetzung freigegebenen Anforderungen entsprechen dem Arbeitsauftrag eines Softwareprojekts. Außerdem lassen sich einmal umgesetzte Anforderungen nicht ohne weiteres ändern oder wieder aus dem System entfernen. Daher hilft ein etablierter und projektbeglei- tender Abstimmungsprozess bei der Auswahl der wichtigsten umzusetzenden Anforde- rungen. In Prozessmodellen, die eine evolutionäre Entwicklung unterstützen, werden beispielsweise für jeden Zyklus die zu bearbeiteten Anforderungen jeweils neu festge- legt. Eine Abstimmung zwischen den Stakeholdern ist erforderlich, sobald in der Menge der dokumentierten Anforderungen Widersprüche identiﬁziert wurden. Falls die sich widersprechenden Anforderungen von verschiedenen Stakeholdern stammen, diese jeweils auch noch widersprüchliche Ziele innerhalb des Projektes verfolgen, liegt eine Konﬂiktsituation vor. Ein weiterer Auslöser für einen Konﬂikt ist die Identiﬁkation eines Widerspruchs zwischen den dokumentierten Anforderungen und der Ansicht eines Sta- keholders. Gründe für diese Situation können beispielsweise im Projektverlauf geän- derte Anforderungen an das System oder eine unvollständige Anforderungsermittlung sein. In der Praxis zeigt sich, dass sich widersprechende Anforderungen oft aus einem unterschiedlichen Verständnis von Begriffen, Zuständigkeiten und der Motivation des Projektes resultieren. Die Aktivitäten zum Prüfen und Abstimmen von Anforderungen lassen sich in vier Schritte unterteilen:

Systematische Qualitätssicherung von Anforderungen, Architekturen und Prozessen

1. Prüfkriterien festlegen;
2. Prüfprinzipien und Prüftechniken auswählen;
3. Prüfung durchführen und Ergebnisse dokumentieren sowie
4. Abstimmen der Anforderungen/Konﬂiktmanagement.

###### Prüfkriterien festlegen

Auch für die Qualitätssicherung von Anforderungen gilt der Grundsatz „Vollständiges Testen ist nicht möglich“. In der Praxis sind auch Anforderungsdokumente in der Regel weder völlig fehlerfrei noch vollständig. Hinzu kommt, dass die industrielle Software- entwicklung ein erkenntnisgetriebener Prozess ist, und sich die Menge der fachlichen Anforderungen im Verlauf eines Entwicklungsprojektes typischerweise mehrfach ändert. Daher müssen im Vorfeld der Prüfung die Kriterien bestimmt werden, nach denen die Menge der Anforderungen genau geprüft werden soll. Welche Kriterien das im Einzel- nen sind, hängt maßgeblich von der Projektsituation, der Art und Wichtigkeit der Anfor- derungen, der verfügbaren Zeit und der dafür eingeplanten Personen ab. Der verant- wortliche Requirements Engineer ist für die Erstellung zuständig.

Prüfkriterien für Anforderungen lassen sich drei verschiedenen Qualitätsaspekten zuordnen: Inhalt, Dokumentation und Abgestimmtheit.

Prüfen des Inhalts

Die zentrale Frage, die mit den Kriterien zum Qualitätsaspekt Inhalt beantwortet werden soll, lautet sinngemäß: „Sind alle relevanten Anforderungen im erforderlichen Detail- grad so erfasst, dass sie verstanden werden können?“.

Prüfen der Dokumentation

Die Kriterien zur Prüfung der Dokumentation zielen auf die Verwendung von angemes- senen Dokumentationsformen und die Dokumentationsqualität ab. Mit den Kriterien zur Prüfung der Dokumentation wird die zentrale Frage: „Wurden sie verständlich und unter Einhaltung der Vorschriften zur Dokumentation dokumentiert?“ beantwortet.

Prüfkriterien zur Abgestimmtheit

Mit den Kriterien zur Prüfung der Abgestimmtheit der Dokumentation soll die Frage

„Stimmen alle relevanten Stakeholder den dokumentierten Anforderungen zu und sind die bekannten Konﬂikte gelöst?“ beantwortet werden. Insbesondere durch das Erlangen neuer Erkenntnisse im Verlauf des Softwareprozesses können neue Anforderungen identiﬁziert werden, die den bereits dokumentierten widersprechen.

###### Prüfprinzipien und Prüftechniken auswählen

Erst nachdem entschieden wurde, welche Prüfkriterien tatsächlich relevant sind, und wenn feststeht, wie viele Ressourcen für die Prüfungen zur Verfügung stehen, werden die zu berücksichtigenden Prüfprinzipien und Prüftechniken ausgewählt.

Im Rahmen der Prüfung von Anforderungen können verschiedene Prüfprinzipien unter- schieden werden, die zwar selbst noch keine Prüftechnik darstellen, jedoch bei der Auswahl der Prüftechniken helfen. Prüftechniken hingegen sind konkrete Verfahren oder Vorgehensweisen, die zur Durchführung der Prüfung eingesetzt bzw. genutzt wer- den. Eine Prüftechnik kann ein oder mehrere Prüfprinzipien unterstützen.

Für die Verbesserung der Qualität der Prüfergebnisse von Anforderungen eignen sich die im Folgenden beschriebenen sechs Prüfprinzipien (vgl. Pohl 2021, S. 106ff.):

* Prinzip 1: Beteiligung der richtigen Stakeholder;
* Prinzip 2: Trennung von Fehlersuche und Fehlerkorrektur;
* Prinzip 3: Prüfung aus unterschiedlichen Sichten;
* Prinzip 4: Geeigneter Wechsel der Dokumentationsform;
* Prinzip 5: Konstruktion von Entwicklungsartefakten;
* Prinzip 6: Wiederholte Prüfung.

Da die Anforderungen im Gegensatz zu Softwareartefakten nicht ausgeführt werden können, können nur Aktivitäten zur statischen Qualitätssicherung unternommen wer- den. Für die Prüfung von Anforderungen sind insbesondere die bereits vorgestellten Review-Techniken relevant. Darüber hinaus werden Anforderungen aber auch durch die versuchsweise Konstruktion von Entwicklungsartefakten oder die Überführung in eine andere Dokumentationsform qualitätsgesichert.

###### Prüfung durchführen und Ergebnisse dokumentieren

Nach Abschluss der fachlichen Vorbereitung muss die Prüfung organisatorisch geplant werden. Dazu zählen die Organisation der benötigten Infrastruktur, die zeitliche Pla- nung aller Beteiligten und gegebenenfalls die Koordination von externen Gutachtern. Wichtig bei der Durchführung ist dabei die Konzentration auf die Dokumentation der Prüfungsergebnisse. Die Fehlerbehebung und Überarbeitung der dokumentierten Anforderungen erfolgt im Anschluss und ist nicht Teil der Prüfung.

###### Abstimmen der Anforderungen/Konﬂiktmanagement

Falls sich widersprechende Anforderungen identiﬁziert wurden und diese nicht einfach behoben werden können, liegt eine Konﬂiktsituation vor. Mit den Aktivitäten zur Abstimmung von Anforderungen soll zum einen die Erreichung eines gemeinsamen und übereinstimmenden Verständnisses der Menge der Anforderungen unter den Stakehol- dern erreicht werden (vgl. Pohl 2021, S. 102). Zum anderen müssen aber auch Entschei- dungen darüber getroffen werden, welche Anforderungen tatsächlich umgesetzt wer- den sollen. Dabei soll mit den Mitteln des Konﬂiktmanagements der Konﬂikt aufgelöst und gleichzeitig die Kooperationsbereitschaft der Stakeholder aufrechterhalten werden.

Systematische Qualitätssicherung von Anforderungen, Architekturen und Prozessen

### Qualitätssicherung von Architekturen

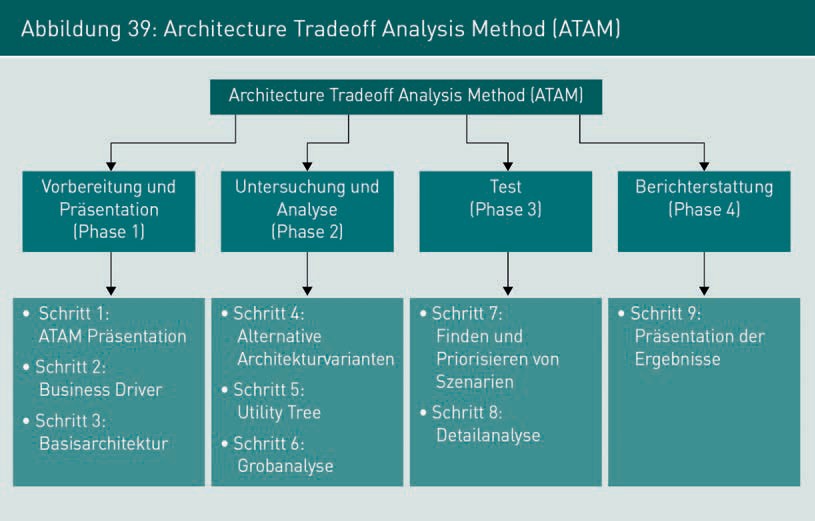
Wann und mit welchen Techniken die Qualität von Architekturen geprüft und bewertet wird, ist grundsätzlich vom Ziel der Prüfung abhängig: Zum einen kann die Eignung einer konzipierten Architektur vor der Implementierung (ex ante) und auf Basis der Architekturbeschreibung geprüft werden. Zum anderen kann begleitend zur Implemen- tierung und nach Ende der Implementierung (ex post) geprüft werden, ob sich die tat- sächlich erstellte Architektur an die Vorgaben der vor der Implementierung erstellten Architekturbeschreibung hält. Diese beiden Prüfziele sind so unterschiedlich, dass für jede dieser Prüfungen verschiedene Prüftechniken eingesetzt werden.

###### Bewertung von Architekturen vor der Implementierung (ex ante)

Mit der Bewertung einer Architektur vor deren Umsetzung soll eine Aussage über die Eignung einer Architektur zur Erfüllung der Anforderungen an ein System erreicht wer- den. Dabei bezieht sich die Bewertung in der Regel auf die Erfüllung von Qualitätsei- genschaften und Randbedingungen, da diese Arten von Anforderungen den größten Einﬂuss auf die Wahl einer geeigneten Architektur haben. Für diesen Zweck werden Techniken der szenariobasierten Architekturanalyse angewendet. Häuﬁg wird die Archi- tecture Tradeoff Analysis Method (kurz: ATAM) eingesetzt. Das Grundprinzip der szenari- obasierten Architekturanalyse ist die Bewertung von Architekturdeﬁnitionen anhand konkreter Anwendungsszenarien des zu erstellenden Systems. Nachdem eine Architek- turdeﬁnition erstellt und dokumentiert wurde, wird diese mithilfe von ausgewählten Anwendungsszenarien untersucht. Dabei wird bei der Architekturdeﬁnition bewertet, ob insbesondere die von der Anwendung geforderten Qualitätseigenschaften unterstützt werden.

###### Phasen und Aktivitäten von ATAM

Die Architecture Tradeoff Analysis Method umfasst nicht nur Aktivitäten zur Bewertung von Architekturdeﬁnitionen, sondern verbindet die Bewertung mit dem Finden mögli- cher Alternativen. Abbildung 39 stellt die im Folgenden näher beschriebenen Phasen und Aktivitäten von ATAM dar (siehe Starke 2020, S. 310-317).



Phase 1: Vorbereitung und Präsentation

Die erste Phase von ATAM dient der Vorbereitung und Präsentation der Basisarchitektur unter Einbeziehung aller relevanten Stakeholder, ggf. auch Anwender oder Fachexper- ten aus den Fachabteilungen. Zunächst wird in Schritt 1 allen Beteiligten die Vorge- hensweise ATAM vorgestellt und erläutert, aus welchen Aktivitäten diese besteht. In Schritt 2 werden die wichtigsten funktionalen Anforderungen, Qualitätsanforderungen und Randbedingungen vorgestellt. Diese werden in ATAM als Business Driver bezeich- net. Mit Schritt 2 soll durch ein gemeinsames Verständnis aller an ATAM beteiligten Sta- keholder die Zielsetzungen des Softwaresystems sichergestellt werden. Der letzte Schritt der Phase 1 ist die Vorstellung der durch die Architekten erstellten Basisarchi- tektur. Hierbei handelt es sich um einen noch zu verfeinernden Architekturentwurf, der jedoch bereits grundlegende Architekturentscheidungen enthält. Ziel von Schritt 3 ist die Darstellung des aktuellen Stands der Architekturdeﬁnition sowie die Klärung von möglicherweise noch offenen Punkten.

Phase 2: Untersuchung und Analyse

Mit den Erkenntnissen aus der ersten Phase erstellt das Architekturteam in Schritt 4 auf Grundlage der Basisarchitektur verschiedene Architekturvarianten, mit denen die vom System geforderten Funktionen und Eigenschaften erfüllt werden können. Im Hinblick auf die Evaluation der Architektur in Phase 3 werden in Schritt 5 Anwendungsszenarien erstellt und priorisiert, mit denen die konkret geforderten Qualitätseigenschaften beschrieben werden. Die Menge der Szenarien mit Qualitätseigenschaften wird in ATAM als Utility Tree bezeichnet. In Schritt 6 von ATAM werden die in Schritt 4 erstellten Architekturvarianten auf Erfüllung der in Schritt 5 hergeleiteten Szenarien grob geprüft. Dabei wird analysiert, inwieweit die verschiedenen Architekturvarianten die Menge der Qualitätseigenschaften im Utility Tree unterstützen. Ergebnis dieser Grobanalyse ist eine priorisierte Liste der Architekturvarianten, die als Input für Phase 3 in ATAM ver- wendet wird.

Systematische Qualitätssicherung von Anforderungen, Architekturen und Prozessen

Phase 3: Test

Das Ziel von Phase 3 ist die Ermittlung der am besten geeigneten Architekturvariante. Dazu wird zunächst in Schritt 7 die bereits in Schritt 5 erstellte Menge von Szenarien verfeinert, erweitert und priorisiert. Dabei werden alle relevanten Stakeholder mit berücksichtigt. In Schritt 8 erfolgt dann die detaillierte Analyse der Architekturvarianten unter Verwendung der in Schritt 7 detaillierten Menge von Szenarien. Diese Prüfung erfolgt vergleichbar wie in Schritt 6, nun allerdings auf Basis der erweiterten und detaillierteren Szenarien. Dabei wird zu den einzelnen Architekturvarianten nicht nur festgestellt, ob sie geeignet oder ungeeignet sind. Es wird ebenso eine Gesamteinschät- zung erarbeitet, bei der die Abhängigkeiten zwischen einzelnen Qualitätseigenschaften sowie der Erreichungsgrad der Eigenschaften unter Einsatz der verschiedenen Architek- turvarianten und den damit verbundenen Risiken dokumentiert werden. In der Regel müssen bei der endgültigen Architekturentscheidung Kompromisse (engl.: Trade off) bezüglich der Erreichung von Qualitätszielen eingegangen werden. Ziel der Detailana- lyse ist die Herausarbeitung dieser Kompromisse.

Phase 4: Berichterstattung

Ziel von Phase 4 ist die Dokumentation der Analyseergebnisse und der dabei getroffe- nen Architekturentscheidungen. Den Abschluss von ATAM bildet die in Schritt 9 stattﬁn- dende Berichterstattung gegenüber den beteiligten Stakeholdern. Hierbei werden die im Verlauf der Schritte 1 bis 8 gewonnenen Erkenntnisse über geforderte Qualitätsei- genschaften und die darauf basierenden Architekturentscheidungen zusammengefasst. Bei der Dokumentation von Architekturentscheidungen ist insbesondere auf deren Nachvollziehbarkeit zu achten. Werden Entwurfsentscheidungen getroffen, müssen die Grundlagen bereits getroffener Entscheidungen im weiteren Verlauf des Softwareproz- esses nachvollziehbar sein.

###### Bewertung von Architekturen während/nach der Implementierung (ex post)

Mit der Prüfung der Architektur begleitend zur Implementierung und/oder nach Abschluss der Implementierung soll sichergestellt werden, dass sich das Ergebnis der Entwicklungsarbeiten an die durch die Architekturdeﬁnition aufgestellten Vorgaben hält. Diese Aktivitäten sind in etwa vergleichbar mit einer Bauüberwachung auf einer Baustelle: Dort müssen die einzelnen Gewerke koordiniert und deren Zwischen- und Endergebnisse dahingehend kontrolliert werden, ob sie in Art und Qualität dem Bau- plan entsprechen. So muss auch bei der Erstellung eines Softwaresystems der Architekt sicherstellen, dass die vom Architekturteam getroffenen Entscheidungen auch tatsäch- lich im Programmcode umgesetzt wurden. Eine Technik hierzu ist das sogenannte Architecture Compliance Checking. Die Grundidee dabei ist die Erstellung eines Soft- waremodells auf Basis des tatsächlich erstellten Programmcodes der Anwendung, bei- spielsweise in Form eines UML-Klassendiagramms oder UML-Komponentendiagramms. Dieses Softwaremodell wird mit dem Modell verglichen, mit dem die Architektur vor der Implementierung beschrieben wurde. Können Abweichungen zwischen den beiden Modellen identiﬁziert werden, müssen diese vom Architekten bewertet und je nach Ein- schätzung vom Entwicklungsteam behoben werden.

Architecture Compli-

ance Checking Das ist eine Vorge- hensweise der Archi- tekturbewertung, bei der Softwaremodelle der Architekturdeﬁ- nition mit Modellen verglichen werden, die aus dem imple- mentierten Code erzeugt werden.

Hinweis

Die regelmäßige und strukturierte Qualitätssicherung von Architekturen ist in der industriellen Praxis häuﬁg kein Standard und wird, wenn überhaupt, nur sporadisch eingesetzt. Oft fehlt das Bewusstsein für die Notwendigkeit dieser Aktivitäten. Wer- den sie außerdem nicht von Beginn an bei der Planung des Projekts berücksichtigt, fehlen während der Projektdurchführung die benötigten Ressourcen hierzu.

### Qualitätssicherung von Softwareprozessen

Neben den Artefakten, die in einem Softwareprozess erzeugt werden, können Software- prozesse bzw. Softwareprozessmodelle ebenfalls hinsichtlich deren Qualität geprüft und verbessert werden. Die Grundannahme hinter der Qualitätssicherung und Verbes- serung von Softwareprozessen ist, dass es einen Zusammenhang zwischen der Prozess- qualität und der Produktqualität gibt. Zur Bewertung von Softwareprozessen können verschiedene speziﬁsche Prozessmerkmale herangezogen werden. Wird eine Ände- rungsnotwendigkeit erkannt, kann diese als Prozessänderungsprozess umgesetzt wer- den.

###### Qualitätsattribute für Softwareprozesse

Um die Änderungsnotwendigkeit von Softwareprozessen zu erkennen, muss zunächst eine Prozessanalyse durchgeführt werden. Das kann recht pragmatisch durch eine Befragung der am Prozess beteiligten Stakeholder erfolgen. Eine weitere Möglichkeit ist das strukturierte Bewerten verschiedener Qualitätskriterien, beispielsweise durch Sta- keholder des Softwareprozesses. Abbildung 40 stellt 10 verschiedene Eigenschaften dar, die zu Prozessen bewertet werden können.

|  |  |
| --- | --- |
| Abbildung 40: Merkmale zu Softwareprozessen | |
| Prozessmerkmal | Beschreibung |
| Verständlichkeit | In welchem Ausmaß ist der Prozess explizit deﬁniert und wie verständlich ist diese Deﬁnition? |
| Standardisierung | Wie stark basiert der Prozess auf einem generischen Stan- dardprozess? Dies kann für einige Kunden interessant sein, die auf Übereinstimmung mit einem Satz deﬁnierter Pro- zessstandards angewiesen sind. Inwieweit wird der gleiche Prozess in allen Bereichen einer Firma eingesetzt? |

Systematische Qualitätssicherung von Anforderungen, Architekturen und Prozessen

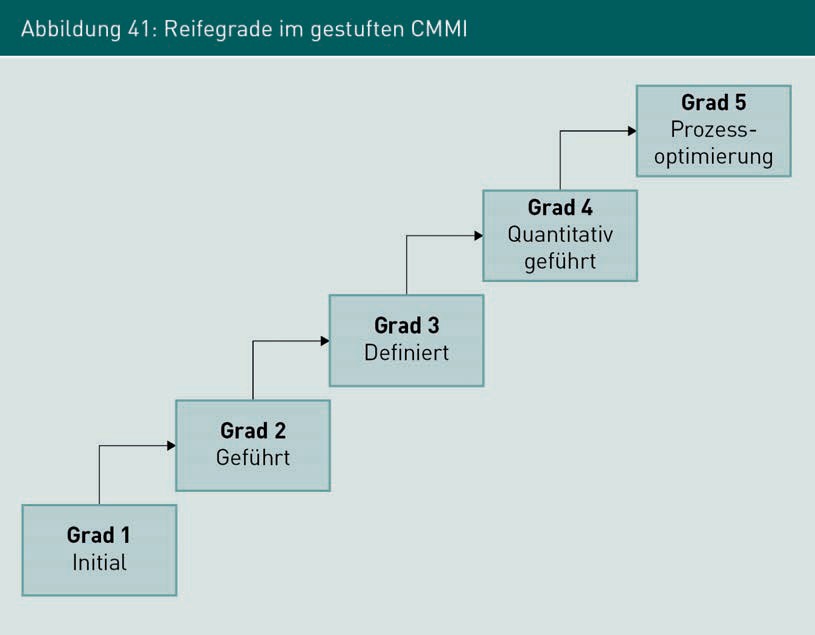
|  |  |
| --- | --- |
| Prozessmerkmal | Beschreibung |
| Sichtbarkeit | Enden die Prozessaktivitäten mit klaren Ergebnissen, sodass der Prozessfortschritt äußerlich erkennbar wird? |
| Messbarkeit | Umfasst der Prozess Datenerhebung oder andere Aktivitä- ten, die das Messen von Prozess- oder Produktmerkmalen erlauben? |
| Unterstützbarkeit | In welchem Ausmaß können Softwarewerkzeuge zur Unter- stützung der Prozessaktivitäten eingesetzt werden? |
| Akzeptanz | Ist der deﬁnierte Prozess für die Entwickler des Software- produkts akzeptabel und einsetzbar? |
| Zuverlässigkeit | Ist der Prozess so gestaltet, dass Prozessfehler vermieden oder aufgedeckt werden, bevor sie Produktfehler verursa- chen? |
| Stabilität | Kann der Prozess trotz unvorhergesehener Probleme fort- gesetzt werden? |
| Wartungsfreund- lichkeit | Kann der Prozess so weiterentwickelt werden, dass er ver- änderte betriebliche Erfordernisse oder Prozessverbesse- rungen widerspiegelt? |
| Schnelligkeit | Wie schnell kann der Prozess der Auslieferung eines Sys- tems aus einer gegebenen Speziﬁkation abgeschlossen werden? |

###### Reifegradmodell des Capability Maturity Model Integration (CMMI)

Unter dem Begriff Capability Maturity Model Integration (kurz: CMMI) wurde vom ameri- kanischen Software Engineering Institute (SEI) ein Reifegradmodell für Softwarepro- zesse entwickelt. Das CMMI-Modell ist als Rahmenwerk für Prozessverbesserungen kon- zipiert. Es enthält dazu unter anderem ein Stufenmodell zur Beurteilung des Reifegrades von Software- und Managementprozessen. Das CMMI-Modell ist sehr umfangreich und umfasst:

* 22 Prozessgebiete aus dem Software Engineering;
* Ziele, die speziﬁsch für jedes Prozessgebiet formuliert werden sowie
* Vorgehensweisen, mit denen die Ziele erreicht werden können.

Der aktuelle Fähigkeitsgrad zu einem Prozess wird dabei auf einer 5-stuﬁgen Skala bestimmt. Die Bestimmung des Reifegrades basiert auf der Analyse der aktuellen IST- Prozesse eines Unternehmens. CMMI-Grad 1 ist dabei die niedrigste Stufe, Grad 5 ent- spricht der höchsten Fähigkeitsstufe. Abbildung 41 zeigt das gestufte CMMI-Reifegrad- modell.



Die folgenden fünf Stufen lassen sich im CMMI-Modell unterscheiden:

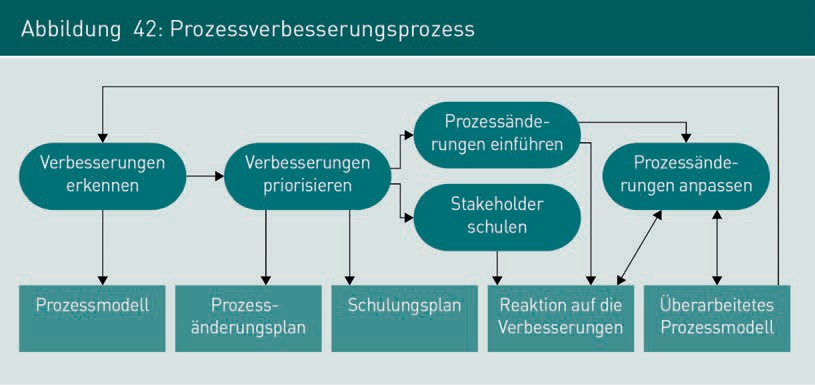
* + Stufe 1 – Initial: Es gibt keine Prozessdeﬁnition; die Aktivitäten ﬁnden ad hoc statt bzw. werden chaotisch durchgeführt. Der Prozess ist in der Regel nicht wiederholbar.
  + Stufe 2 – Geführt: Der Prozess ist wiederholbar. Umfang und Ergebnis der zu erledi- genden Aktivitäten ist den Teammitgliedern bekannt. Es gibt aber keinen detaillier- ten Prozess, der Aktivitäten, Rollen, Ergebnisse und deren Abhängigkeiten explizit deﬁniert.
  + Stufe 3 – Deﬁniert: Es gibt einen deﬁnierten Prozess, in dem Aktivitäten, Rollen und Ergebnisse verbindlich dokumentiert sind. Der Prozess enthält konkrete Methoden und Vorgehensweisen zu den einzelnen Software Engineering Aktivitäten.
  + Stufe 4 – Quantitativ geführt: Die Prozessqualität kann über Messung quantitativer Prozesseigenschaften festgestellt und gesteuert werden. Über erhobene Kennzahlen zu Aktivitäten der Prozesse sowie zu den erzeugten Ergebnissen können Havarien identiﬁziert und entsprechend gegengesteuert werden.
  + Stufe 5 – Prozessoptimierung: Prozess- und Produktmessungen werden kontinuier- lich erhoben und zur Prozessverbesserung eingesetzt. Je nach Messwert kann der Prozess auf den aktuellen Bedarf hin angepasst werden.

Systematische Qualitätssicherung von Anforderungen, Architekturen und Prozessen

Nach dem CMMI-Modell sollte sich der Reifegrad einer Organisation stufenweise von einem niedrigen zu einem höheren Grad hin entwickeln. CMMI-Reifegrade können daher zur Bewertung einer Organisation eingesetzt werden. Neben dem gestuften CMMI-Modell gibt es auch ein kontinuierliches CMMI-Modell. Dabei wird nicht der Rei- fegrad zu einer gesamten Organisation bestimmt, sondern jeweils zu verschiedenen Prozessgruppen.

###### Vorgehen zur Prozessverbesserung

Für den Fall, dass bei der Prozessanalyse Verbesserungsvorschläge erarbeitet wurden, müssen diese strukturiert in das Softwareprozessmodell übernommen werden. Dazu kann, wie in Abbildung 42 exemplarisch dargestellt, ein Prozess für Prozessverbesserun- gen etabliert werden. Dieser Prozess gibt einen Überblick über die bei einer Prozessver- besserung typischerweise durchzuführenden Aktivitäten und deren Ergebnisse. Der Pro- zess ist ein zyklischer Prozess, das bedeutet, er wird grundsätzlich niemals endgültig sein. Auch ein verbessertes Prozessmodell kann im Laufe der Zeit noch weiter verbes- sert werden, was der Umsetzung des Prinzips der ständigen Verbesserung des TQM ent- spricht.



Die erste Aktivität im Verbesserungsprozess ist das Erkennen von Verbesserungsbedarf in der aktuellen Prozessversion und die Erarbeitung möglicher Verbesserungsvor- schläge. Die Vorschläge werden anschließend priorisiert. Im Zuge der Priorisierung wer- den die Vorschläge auf deren Umsetzbarkeit und die für die Umsetzung erforderlichen weiteren Aktivitäten hin analysiert. Die Ergebnisse der Priorisierung sind ein Ände- rungsplan, der unter anderem die Zeitpunkte und Verantwortlichkeiten für die Einfüh-

rung von Prozessverbesserungen sowie mögliche Abhängigkeiten zwischen einzelnen Verbesserungsschritten dokumentiert. Darüber hinaus wird ein Schulungsplan erstellt, auf dessen Grundlage konkrete Schulungsmaßnahmen der von den Änderungen betrof- fenen Stakeholder durchgeführt werden. Neben dem Wissen um die Änderungen im Softwareprozess müssen für die Umsetzung der Prozessänderungen unter Umständen auch neue fachliche Kompetenzen erlangt werden. Das wird durch die Schulung der betroffenen Stakeholder sichergestellt. Mit der Einführung von Prozessänderungen müssen häuﬁg Vorlagen, Dokumentationen, Werkzeuge und Vorgehensweisen ange- passt werden. Insbesondere nach der Einführung von Prozessänderungen ist eine Anpassungsphase erforderlich. Hierbei werden die Prozessanpassungen beobachtet und ausgewertet. Vor allem bei größeren Änderungen ist es häuﬁg erforderlich, dass deren konkrete Umsetzung an verschiedenen Stellen noch kleinere Anpassungen erfor- dert. Daher sollte in der Anpassungsphase ein intensiver Austausch zwischen den betroffenen Stakeholdern und den Prozessgestaltern stattﬁnden. Nach Abschluss der Anpassungsphase hat das Softwareprozessmodell wieder einen stabilen Zustand erreicht und der Verbesserungszyklus kann von vorne beginnen.

Zusammenfassung

Die analytische Qualitätssicherung von Anforderungen wird in der Kernaktivität Prü- fen und Abstimmen aus dem Requirements Engineering durchgeführt. Das Ziel der Aktivitäten zum Prüfen und Abstimmen ist die Freigabe der Anforderungen zur Umsetzung in den nachgelagerten Phasen des Softwareprojekts. Die Qualität von Architekturen kann einerseits vor der Implementierung (ex ante) und auf Basis der Architekturbeschreibung geprüft werden. Eine häuﬁg dazu eingesetzte Technik ist die szenariobasierte Architekturanalyse, beispielsweise mit ATAM. Das Grundprinzip dabei ist die Bewertung von Architekturdeﬁnitionen anhand von konkreten Anwen- dungsszenarien des zu erstellenden Systems. Zum anderen kann bezüglich der Architektur nach Ende der Implementierung (ex post) geprüft werden, ob die tat- sächlich erstellte Architektur auch die Vorgaben der vor der Implementierung erstellten Architekturbeschreibung hält. Zur Bewertung von Softwareprozessen kön- nen verschiedene speziﬁsche Prozessmerkmale herangezogen werden oder dessen Reifegrad bestimmt werden. Soll ein Prozess angepasst werden, kann dies mit einem Prozessänderungsprozess umgesetzt werden.



# Anhang 1

## Literaturverzeichnis

134 Anhang 1

Literaturverzeichnis

Balzert, H. (2008): *Lehrbuch der Software-Technik: Software-Management.* 2. Auﬂage, Sprektrum Akademischer Verlag, Heidelberg.

Balzert, H. (1998): *Lehrbuch der Software-Technik, Software-Management, Software- Qualitätssicherung, Unternehmensmodellierung.* Spektrum Akademischer Verlag, Hei- delberg.

Checkstyle (o. J.): (URL: https://checkstyle.sourceforge.io/checks.html [letzter Zugriff 28.02.2022])

SpotBugs (o. J.): *Bug Descriptions.* (URL: https://spotbugs.readthedocs.io/en/latest/ bugDescriptions.html [letzter Zugriff 28.02.2022])

Gabler Wirtschaftslexikon (2014): Springer Gabler Verlag. (URL: https://wirtschaftslexi- kon.gabler.de/deﬁnition/qualitaetssicherung-44396/version-267707 [letzter Zugriff 28.02.2022])

German Testing Board (2014): S. 43. (URL: https://[www.german-testing-board.info/wp-](http://www.german-testing-board.info/wp-) content/uploads/2022/01/GTB-CTFL\_Lehrplan\_v3.1\_DE.pdf [letzter Zugriff 28.02.2022])

Grechenig, T. et al. (2010): *Softwaretechnik.* Addison-Wesley Verlag, München.

Kleiner, F. (2021): *IT Service Management.* mitp Verlag, Frechen.

Liggesmeyer, P. (2002): *Software Qualität – Testen, Analysieren und Veriﬁzieren von Soft- ware.* Spektrum Verlag, Heidelberg.

Ohno, T. (2006): *„Ask ‘why’ ﬁve times about every matter.“* (URL: http://www.toyota-glo- bal.com/company/toyota\_traditions/quality/mar\_apr\_2006.html [letzter Zugriff: 04.08.2014]).

PMD (o. J.): *Current Rulesets.* (URL: <http://pmd.sourceforge.net/pmd-5.1.1/rules/> index.html [letzter Zugriff: 28.02.2022]).

Pohl, K./Rupp, C. (2021): *Basiswissen Requirements Engineering – Aus- und Weiterbil- dung zum „Certiﬁed Professional for Requirements Engineering“. 5*. Auﬂage, dpunkt.ver- lag, Heidelberg.

Schatten, A. et al. (2010): *Best Practice Software-Engineering.* Spektrum Akademischer Verlag, Heidelberg.

Schiersmann, C./Thiel, H.-U. (2000): *Projektmanagement als organisationales Lernen.*

Leske + Budrich Verlag, Opladen.

Sommerville, I. (2018): *Software Engineering.* 10. Auﬂage. Addison-Wesley Verlag, Mün- chen.

##### Anhang 1

135

Literaturverzeichnis

Sneed, H./Baumgartner, M./Seidl, R. (2012): *Der Systemtest.* Carl Hanser Verlag, Mün- chen.

Starke, G. (2020): *Effektive Softwarearchitekturen*. 9. Auﬂage. Hanser Verlag.

Wallmüller E. (1990): *Software-Qualitätssicherung in der Praxis.* Carl Hanser Verlag, München.