Testen im Software Lebenszyklus

Dieter Volland, MicroConsult GmbH
d.volland@microconsult.de

Die Systeme werden komplexer, die Entwicklungsteams größer. Der Ruf nach Struktur und Systematik in der Entwicklung wird lauter. Ein Prozess muss her.

Ein Projekt umfasst mehrere Prozesse wie Projekt- und Qualitätsmanagement als übergeordnete Prozesse, den Software-Entwicklungsprozess mit den Phasen Analyse, Design und Implementierung und natürlich Test. Als begleitende Prozesse gehören dazu das Requirements-Engineering und -Management zum Finden und Verwalten von Anforderungen sowie das Konfigurations-Management, welches das Änderungs-, Versions- und Release-Management beinhaltet.

Der Testprozess ist ein wichtiger Bestandteil des Projektes, Testen findet nicht isoliert statt, es ist ein wichtiger Teil-Prozess des Gesamtprozesses. Wie das W-Modell sehr deutlich zeigt, sind Testaktivitäten immer bezogen auf Softwareentwicklungsaktivitäten. Das Testen beginnt also bereits mit der Analyse der Anforderungen und endet mit der Außerbetriebnahme des Produktes.

In diesem Vortrag wird der Testprozess detailliert vorgestellt.

Das W-Modell ist eine Erweiterung des sehr verbreiteten V-Modells, es werden die Testaktivitäten als paralleler Prozess zur Entwicklung der Software betrachtet. Alle Testaktivitäten, die vor der Ausführung der Testfälle auf den unterschiedlichen Teststufen durchzuführen sind, können und sollen bereits parallel zu den frühen Entwicklungsphasen bearbeitet werden.

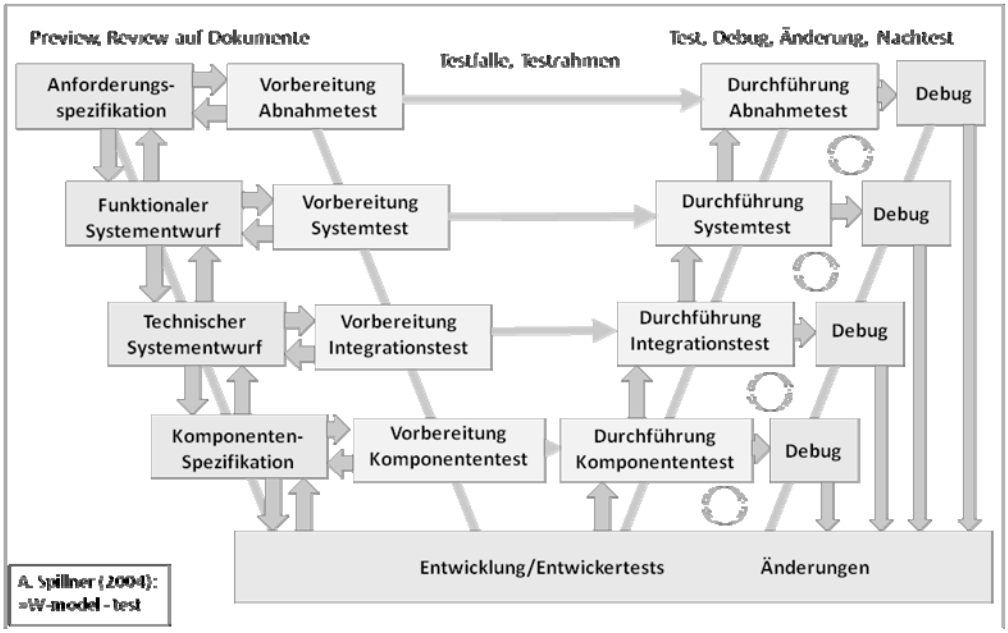

Bild 1: W-Modell, Vorgehensmodell für die Softwareentwicklung, Prof. Dr.-Ing. A. Spillner

1. Der Testprozess

Der fundamentale Testprozess besteht aus 5 Hauptaktivitäten.

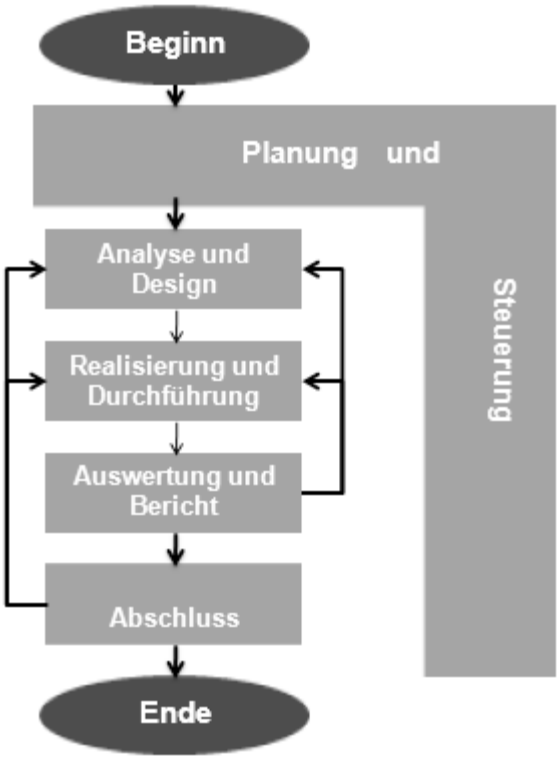

Bild 2: Der Testprozess

Testplanung

In der Phase Testplanung werden die allgemeine Testziele definiert, wie aufdecken von Fehlerzuständen bevor sie zur Auswirkung kommen, vorbeugen von Fehlerzuständen, erzeugen von Vertrauen bezüglich des Qualitätsniveaus des Systems. Um diese Ziele zu erreichen, muss eine Teststrategie (dokumentiert im Testkonzept/Testplan) entworfen werden, die den notwendigen Aufgabenumfang definiert. Dazu gehören **Testobjekte** und **Funktionen** identifizieren, ein Konzept zur Auswahl von effektiven Testfällen festlegen, Teststufen, Testmethoden und Testintensität festlegen, Testaktivitäten festlegen, Resourcen festlegen, Zeitplan erstellen, Testumgebung planen, Testdokumentation festlegen, Testendekriterium festlegen.

Teststeuerung

Der Testfortschritt muss laufend überprüft werden, um Abweichungen vom Plan festzustellen und Korrekturmaßnahmen einzuleiten. Um Tests steuern zu können, ist es notwendig, projektbegleitend geeignete Fortschrittsdaten (Metriken) zu ermitteln.

Testanalyse

Review der **Testbasis** (z.B. Anforderungen, Risikoanalysebericht, Architektur-, Design- und Schnittstellenspezifikationen), Bewertung der Testbarkeit von Testbasis und Testobjekten

Identifizierung und Priorisierung der **Testbedingungen** auf Grundlage der Testobjektanalyse, der Spezifikation, des Verhaltens und der Struktur der Software.
Unter Testbedingung versteht man eine Einheit oder ein Ereignis, z.B. eine Funktion, eine Transaktion, ein Feature, ein Qualitätsmerkmal oder ein strukturelles Element einer Komponente oder eines Systems, welche bzw. welches durch einen oder mehrere Testfälle verifiziert werden kann.

Testentwurf

Für die identifizierten Testbedingungen werden abstrakte Testfälle entworfen, das heißt, Testfälle ohne konkrete Ein- und Ausgabewerte für Eingabedaten und vorausgesagte Ergebnisse. Er verwendet logische Operatoren, weil die konkreten noch nicht definiert oder verfügbar sind.
Zu dieser Phase gehört auch der Entwurf des Testumgebungsaufbaus und Identifikation der benötigten Infrastruktur und Werkzeuge und die Sicherstellung der Rückverfolgbarkeit zwischen Testbasis und Testfällen in beiden Richtungen.

Testrealisierung (Testabläufe spezifizieren)

Für die abstrakten Testfälle werden nun konkrete Testfälle und Testdaten entwickelt. Die Testfälle werden priorisiert und die Testablaufreihenfolge wird festgelegt. Es werden Testszenarien erstellt.

Testdurchführung

Die Testszenarien werden manuell oder automatisiert ausgeführt. Ist-Ergebnisse werden mit den vorausgesagten Ergebnissen verglichen, die Ergebnisse werden protokolliert, die Version des jeweiligen Testobjekts und der eingesetzten Testwerkzeugen und Testmittel wird dokumentiert.
Gefundene Fehlerwirkungen oder Abweichungen werden festgehalten und analysieren, um den Grund eines Problems festzustellen (z.B. Fehler im Code, in spezifizierten Testdaten, im Testdokument oder Fehler bei der Durchführung passiert).
Bei aufgedeckten Fehlerwirkung nach der Behebung der Ursachen muss ein Fehlernachtest durchgeführt werden und eventuell ein Regressionstest, um sicherzustellen, dass die Fehlerbehebung keinen negativen Einfluss auf bereits bestehende Funktionalität hatte.

Bewertung von Testendekriterien und Bericht

Die Testprotokolle werden in Hinblick auf die im Testkonzept (Testplan) festgelegten Testendekriterien ausgewertet und entschieden, ob mehr Tests durchgeführt oder die festgelegten Testendekriterien angepasst werden müssen.
Sind die festgelegten Testendekriterien erreicht, wird ein Testabschlussbericht erstellt.

2. Teststufen

Getestet wird auf allen Stufen.

Komponententest

Im Komponententest werden die in der Entwicklung erstellten Softwarebausteine erstmalig einem systematischen Test unterzogen. Es werden komponenteninterne Aspekte geprüft.

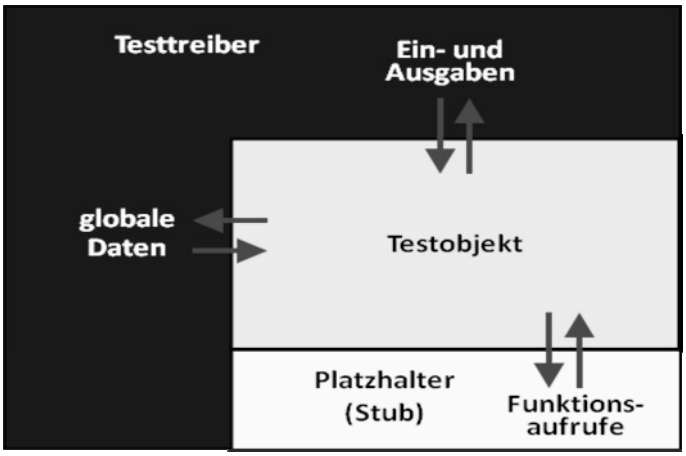

Bild 3: Komponententest

Typische **Testobjekte** sind einzelne Softwarebausteine wie Klassen, Objekte, Module, Komponenten.
Die **Testfälle** werden aus den Komponentenanforderungsspezifikationen, Komponenten-designspezifikationen, oder aus dem Sourcecode abgeleitet.
Testrahmen und **Testumgebung** sind Entwicklungsumgebung und Debugging-Werkzeuge oder spezielle Softwaretesttools (Tessy, Parasoft C++test, …), Unit Testtools (J-Unit, C-Unit, Embedd-Unit, …)
In der Praxis sind oft die für den Code verantwortlichen **Entwickler** an den Komponententests beteiligt. Sie werden deshalb auch Entwicklertests genannt. Der Vorteil ist, dass gefundenen Fehler sofort korrigiert werden können und gar nicht erst formell behandelt werden müssen.
Das **Ziel** des Komponententest ist, Module, Programme, Objekte, Klassen, etc. isoliert vom Rest des Systems separat testen, um Fehler zu finden und den Nachweis zu erbringen, dass das Testobjekt die spezifizierten Funktionalität erfüllt.
Anzuwendende **Testarten** sind funktionale Tests (Blackboxtests), strukturorientierte Tests (Whiteboxtests) und nicht-funktionale Tests (Performance, Robustheit, Wartbarkeit, Zuverlässigkeit, Sicherheit).

Integrationstest

Im Integrationstest werden bereits einzeln getestete Teile zu einer größeren Baugruppe integriert und es wird getestet, ob das Zusammenspiel aller Einzelteile miteinander richtig funktioniert.

Ein Komponentenintegrationstest prüft das Zusammenspiel der Softwarekomponenten und wird nach dem Komponententest durchgeführt.
Ein Systemintegrationstest prüft das Zusammenspiel verschiedener Softwaresysteme oder zwischen Hardware und Software und kann nach dem Systemtest der Einzelsysteme durchgeführt werden.
Je größer der Umfang einer Integration ist, desto schwieriger ist die Isolation von Fehlerzuständen in einer spezifischen Komponente oder einem System, was zur Erhöhung des Risikos und zusätzlichem Zeitbedarf zur Fehlerbehebung führen kann.

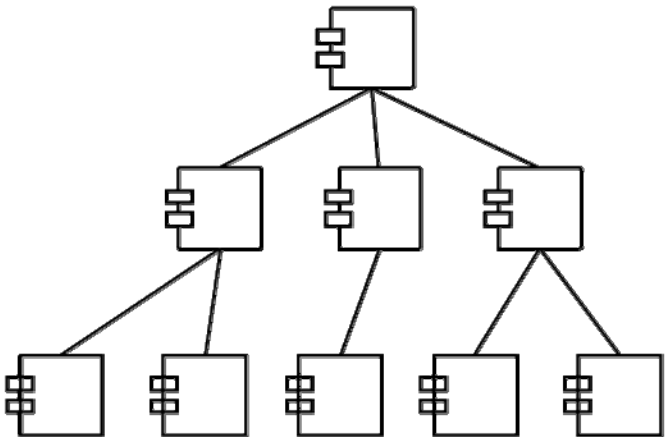

Bild3: Integration

Die **Testfälle** für den Integrationstest werden aus den System- und Softwaredesignspezifikationen, den Nutzungsabläufen/Funktionsabläufen, Anwendungsfälle (use cases) abgeleitet.
Typische **Testobjekte** sind Subsysteme, Erweiterungen bestehender Systeme, Schnittstellen zwischen Systemen und Netzwerke.
Testrahmen und **Testumgebung** sind Tools aus dem Komponententest, Monitore, Protokollanalysatoren.
Der Integrationstest wird in der Regel von einer unabhängigen **Testabteilung** durchgeführt.
Ziel des Integrationstests sind der Test der Interaktionen zwischen verschiedenen Teilen eines Systems, Schnittstellen zwischen Komponenten/Systemen, Schnittstellen zum Betriebssystem, zur Hardware.
Anzuwendende **Testarten** sind funktionale Tests (Blackboxtests), strukturorientierte Tests (Whiteboxtests) und nicht-funktionale Tests (Performance, Robustheit, Wartbarkeit, Zuverlässigkeit, Sicherheit) und Schnittstellenüberdeckungstests.

Je nach Art und Umfang des zu integrierenden Systems kommen unterschiedliche **Integrationsstrategien** zur Anwendung.

Top Down Integration eignet sich für Systeme mit User Interface, mit Komplexität in den oberen Komponenten oder wenn die Anforderungen unklar sind. Es werden die oberen Komponenten zuerst integriert. Das hat den Vorteil, dass schnell ein vorzeigbares System vorhanden ist und man braucht keine Treiber. Dafür sind aber Platzhalter (Stubs) für die noch nicht vorhandenen Teile erforderlich.

Bei der **Bottom-Up Integration** werden die unteren Module werden zuerst integriert. Neue Komponenten werden erst nach dem Test aller vorhandenen Komponenten hinzugefügt. Diese Integrationsstrategie eignet sich besonders für embedded Systeme und Systeme mit Komplexität in den unteren Komponenten.
Vorteil ist, dass keine Platzhalter (Stubs) erforderlich sind, dafür werden aber Treiber gebraucht.
Nachteilig ist auch die späte Verfügbarkeit eines lauffähigen Gesamtsystems.

Ad hoc Integration eignet sich, wenn die Fertigstellung der Komponente nicht planbar ist, die Komponenten werden nach Verfügbarkeit integriert. Nach Fertigstellung eines Moduls kann es sofort integriert werden, man braucht aber Treiber und Stubs.

Bei der **Hardes first Integration** werden kritische, d.h. besonders kompliziert zu testende oder potentiell fehlerbehaftete Komponenten zuerst getestet.
Dies setzt eine Risikoanalyse u.Erfahrung voraus und eine "Rangliste" der kompliziertesten Komponenten muß aufgestellt werden. Bei großer Verteilung dieser Komponenten ist eine Integration schwierig.
Vorteil ist, dass kritische Komponenten zuerst getestet werden, Nachteil ist, dass Testtreiber und Stubs notwendig sind.

Bei der Big Bang Integration werden alle Module gleichzeitig integriert. Dies eignet sich bei kleinen Projekten oder wenn die Komponenten eine gute Qualität haben.
Vorteile ist, dass keine Testtreiber und Stubs notwendig sind. Nachteilig ist, dass Fehler schwer lokalisierbar sind.

Systemtest

Die **Testfälle** für den Systemtest werden aus der System- und Anforderungsspezifikation, von Anwendungsfällen (use cases), aus Risikoanalyseberichten abgeleitet.
Typische **Testobjekte** sind System-, Anwender- und Betriebshandbücher, Systemkonfiguration und Konfigurationsdaten.
Testrahmen und **Testumgebung** sollten mit der finalen Ziel- oder Produktivumgebung so weit wie möglich übereinstimmen. In der Regel wird der Systemtest von einer unabhängigen Testabteilung durchgeführt.
Testziel ist der Test des Verhalten eines Gesamtsystems/-produktes. Es ist der letzter Test vor der Abnahme.
Anzuwendende **Testarten** sind funktionale Tests (Blackboxtests), strukturorientierte Tests (Whiteboxtests) und nicht-funktionale Tests (Performance, Robustheit, Wartbarkeit, Zuverlässigkeit, Sicherheit).

Abnahmetest

Die **Testfälle** für den Abnahmetest werden aus den Systemanforderungen, von Anwendungsfällen (use cases), aus Risikoanalyseberichten abgeleitet.
Typische **Testobjekte** sind Teile eines Systems oder ein voll integriertes System
Der Abnahmetest liegt meist im Verantwortungsbereich der Kunden oder Benutzer des Systems. **Testziel** ist, Vertrauen in das System zu gewinnen und die Bewertung der Bereitschaft eines Systems für den Einsatz und die Nutzung.
Das Finden von Fehlerzuständen ist nicht das Hauptziel beim Abnahmetest.
Testarten sind funktionale und nicht-Funktionale Tests

Das **Ergebnis** eines Abnahmetests kann sein: Abnahme, Abnahme unter Vorbehalt, Teilabnahme oder keine Abnahme.
Ggf. sind Nacharbeiten und ein erneuter Abnahmetest notwendig

Wartungstest

Der Wartungstest wird nach dem Abnahmetest an einem **betreibbaren System** ausgeführt. Tests sind bedingt durch Modifikationen, Migrationen oder Einzug (Test vor der Archivierung) des Systems oder Software. Es werden die eingebrachten Modifikationen getestet. Der Umfang der Wartungstests ist abhängig vom dem mit der Änderung verbundenen Risiko, der Größe des existierenden Systems und dem Umfang der Änderung. In einer **Auswirkungsanalyse** wird entschieden, wie viele Regressionstests durchzuführen sind.

3. Testarten

Eine Testart ist eine Gruppe von Testaktivitäten, mit der Absicht, eine Komponente oder ein System auf einige zusammenhängende Qualitätsmerkmale zu prüfen. Eine Testart ist auf ein bestimmtes Testziel fokussiert, wie z.B. Funktionalitätstest, Zuverlässigkeitstest, Regressionstest, Benutzbarkeitstest.

Man unterscheidet folgende **Testarten**:
Funktionaler Test, testen der nach außen sichtbaren Funktionalität (Black-Box-Test).
Die Funktionalität besagt, „**was**“ das System leistet. Es soll sicher stellen, dass das System tut was es soll, und nicht tut, was es nicht soll.

Nicht-funktionaler Test, testen der nicht-funktionalen Softwaremerkmale, d.h. testen mit welcher **Qualität** (z.B. Software-Qualitätsmerkmale nach ISO 9126) das System die Funktionalität erbringt.
Die Qualität besagt, „**wie**“ das System arbeitet.
Es wird das von außen sichtbare Verhalten der Software (Black-Box-Test) betrachtet.

Strukturorientierter Test, testen der interne Struktur bzw. Architektur der Software (Whitebox).
Testen der Codeüberdeckung, Anweisungs- oder Entscheidungsüberdeckung, dabei wird jeweils der prozentuale Anteil der überdeckten Strukturelemente angegeben. Die Testüberdeckung ist ein Maß dafür, inwiefern eine Struktur durch Testfälle geprüft bzw. ausgeführt (überdeckt) wurde.

Fehlernachtest ist ein erneuter Test nach **Behebung** eines Fehlerzustandes. Er soll nachweisen, dass vorherige Fehler korrigiert wurden. Es werden alle Test wiederholt, die Fehlerwirkungen erzeugt haben. Tests für Fehlernachtests müssen wiederholbar sein.
Nicht zu verwechseln mit Debugging (Lokalisieren, und Entfernen eines Fehlerzustands), das ist eine Entwicklungs- und keine Testaufgabe.

Regressionstest ist ein erneuter Test einer **getesteten** Software. Er soll nachweisen, dass bei Änderung von Softwareteilen, nach Fehlerkorrektur, Wartungsarbeiten, Weiterentwicklung oder Änderung der Umgebung keine neuen Defekte eingebaut, oder bisher maskierte Fehlerzustände freigelegt wurden. Eine Einflussanalyse zur Bestimmung der Auswirkungen und Seiteneffekte der Änderung wird durchgeführt. Eine Risikoanalyse bestimmt den

Umfang des Regressionstest. Dabei wird abgewogen zwischen niedrigen Kosten und hohem Risiko.
Tests für die Regressionstests müssen wiederholbar sein.

Die Testarten ergänzen sich und verfolgen gleiche übergeordnete Ziele (Fehler finden) und decken unterschiedliche Typen von Fehlern auf.

4. Testmethoden

Statische Prüfung

Jedes Arbeitsergebnis der Softwareentwicklung kann einer statischen Prüfung unterzogen werden. Anforderungsspezifikationen, Designspezifikationen, Testkonzepte, Testspezifikationen, Testfälle, Testskripte, Anwenderhandbücher und natürlich auch Quellcode.
In Reviews, die von informell bis sehr formal durchgeführt werden können, werden Fehler und Fehlerzustände im Code, Fehlerzustände in Anforderungen und im Design, Abweichungen von der Spezifikation und Abweichungen von einzuhaltende Vorgaben und Standards aufgedeckt.
60% bis 90% aller Fehler in der Spezifikation und im Code werden allein durch Inspektionen gefunden. Was eine Beschleunigung des Codierens (Spezifikation ist stabiler) und Beschleunigung des Testens (viele Fehler sind schon behoben) zu Folge hat.

Neben dem Aufdecken von Fehlern habe Reviews noch weitere positive Effekte, wie z.B. Wissen und Verständnis im Team verbreiten, Verantwortlichkeit des Teams erhöhen, ohne dass sie dem Autor beschnitten wird und Verbesserung des Team-Geistes (bei richtiger Durchführung).
Es ist ein unbewusster Lernprozess für die Teilnehmer und damit werden erkannte Fehler in der Zukunft nicht wiederholt.

Dynamischer Test

Beim dynamischen Test wird das Testobjekt in einer Testumgebung (Debugger, Testtool) ausgeführt. Es sind Testdaten, Testtreiber und Platzhalter (Stubs) erforderlich.

Beim **Black-Box-Verfahren** werden die Testfälle oder Testdaten aus der Spezifikation einer Komponente oder eines Systems abgeleitet, es werden keine Informationen über die interne Struktur der Komponente oder des Systems verwendet.
Um eine hohe Fehlerentdeckungswahrscheinlichkeit bei minimaler Anzahl von Testfällen zu erreichen, werden Eingabewerte für die Software oder das System in Wertebereiche, so genannte **Äquivalenzklassen,** eingeteilt. Gültige Äquivalenzklassen und ungültige Äquivalenzklassen, unter der Annahme, dass sich die Software oder das System bei allen Mitgliedern einer Äquivalenzklasse gleich verhält.
Da das Verhalten an der Grenze jeder Äquivalenzklasse mit einer höheren Wahrscheinlichkeit fehlerhaft als das Verhalten innerhalb der Klasse ist, wird zusätzlich ein Test für jeden **Grenzwert** gewählt.
Ist die Funktionalität eines Systems durch Ausführung von Regeln gegeben, können die Testfälle in einer **Entscheidungstabelle** entworfen werden.
Wenn ein System in Abhängigkeit von aktuellen Gegebenheiten oder von seiner

Vorgeschichte (seinem Zustand) unterschiedliche Reaktionen zeigt, dann kann dieser Aspekt des Systems mit einem Zustandsdiagramm dargestellt werden und ein **zustandsbasierter Test** durchgeführt werden.

Beim **White-Box-Verfahren** baut der Test auf der vorgefundenen Struktur der Software oder des Systems auf, die Testfälle werden aus dem Aufbau der Software (Struktur) abgeleitet.
Ziel des Tests ist die Überdeckung von Anweisungen (C0), Verzweigungen (C1) und Bedingungen (MCDC). Der Überdeckungsgrad der Software wird in Prozent gemessen. Überdeckungstests können fehlende Testfälle für Anforderungen (Qualität der Testfallmenge), "toten Code" oder unbeabsichtigte Funktionalität aufdecken.

Beim **erfahrungsbasierten Testentwurfsverfahren** werden die Testfälle aus dem Wissen und der Erfahrung von Menschen abgeleitet. Das Wissen über wahrscheinliche Fehlerzustände und
ihre Verteilung ist eine Informationsquelle. Intuition der Tester ist eine weitere Informationsquelle.

Die Wahl, welche Testmethoden verwendet werden sollen, hängt von der Art des Systems, Kunden- oder Vertragsanforderungen , Risikostufe, Risikotyp, Testziel, verfügbarer Dokumentation, Wissen der Tester, Zeit und Geld, frühere Erfahrungen mit gefundenen Fehlerzustandsarten usw. ab.
Einige Techniken sind für bestimmte Situationen und Teststufen besser geeignet, andere Techniken sind in allen Teststufen gleichermaßen einsetzbar.

5. Zusammenfassung

Wichtig ist, dass die Testaktivitäten so früh wie möglich beginnen und über das gesamte Projekt parallele zur Entwicklung laufen. Dieser Vortrag soll dem Leser einen Überblick über die Nebenläufigkeit von Test und Entwicklung, die Teststufen, anwendbaren Testarten und Testmethoden geben.

6. Literatur

Basiswissen Software Test, Andreas Spillner, Theo Linz
Praxiswissen Software Test: Testmanagement, Andreas Spillner, Theo Linz, Mario Winter, Thomas Roßner
Praxiswissen Software Test: Testanalyst, Graham Bath, Judy McKay
ISTQB Certified Tester Foundation Level Syllabus, Version 2011
Software Test für Embedded Systeme, Microconsult GmbH

7. Autor

Dipl. Ing. Dieter Volland ist Mitbegründer der Microconsult GmbH und führt seit über 20 Jahren Trainings, Coachings und Beratungen im Bereich der Embedded Systementwicklung durch.