7 Der Testprozess

Testen ist keine Tätigkeit, die als Ganzes am Ende der Software-Entwicklung durchgeführt wird. Zum einen ist Testen eine den gesamten Entwicklungsprozess begleitende Maßnahme. Zum anderen ist es – wie bei jeder anderen umfangreichen Aufgabe – auch hier ratsam, den Vorgang des Testens in verschiedene Teilaktivitäten (Phasen) aufzuteilen und diese Aktivitäten Personen mit verschiedenen Rollen zuzuteilen.

Dieses Kapitel beschreibt die Grundlagen eines strukturierten Testkonzepts (Phasenmodell, organisatorische Einbettung, Werkzeuge und Infrastruktur, brauchbare Testmethoden), die Positionierung von Testen im SE-Prozess sowie die Phasen innerhalb des Testprozesses.

7.1 Rollen im Testprozess

Wir haben im Kapitel 6 schon gesehen, dass die Anzahl der Durchführenden im Testprozess von einer Einzelperson bis zu einem ganzen Testteam reichen kann. Vor allem bei größeren Testteams ist es wichtig, dass den einzelnen Mitarbeitern Rollen mit eigenen Kompetenzbereichen zugewiesen werden. Es ist dabei durchaus möglich, dass Einzelpersonen mehrere Rollen ausführen. In den meisten Testprozessen können folgende Rollen identifiziert werden:

Testmanager. Der Testmanager ist für den funktionierenden Ablauf des Tests verantwortlich. Er führt in Zusammenarbeit mit der Projektleitung die Testplanung durch, teilt den Testern und Testentwicklern Arbeitspakete zu, kontrolliert den Testfortschritt, erstellt Testberichte und stellt die Schnittstelle zwischen Testteam und Projektleitung dar.

Testingenieur. Der Testingenieur hat die Aufgabe, die vom Testmanager entwickelte Teststrategie umzusetzen. Dazu gehört die Auswahl von geeigneten Testmethoden und Testwerkzeugen sowie Entwurf und Leitung der Realisierung von Testfällen.

Testentwickler. Der Testentwickler realisiert die vom Testingenieur entworfenen Testfälle. Dies kann z.B. die Konkretisierung von generischen Testfällen oder die Entwicklung der Testfälle mittels eines Capture-/Replay-Werkzeugs sein.

Tester. Der Tester ist derjenige, der einen Test tatsächlich durchführt. Als Grundlage für seine Arbeit dienen die vom Testentwickler erzeugten Testfälle samt vorgegebener Infrastruktur, sowie die Direktiven vom Testmanager. Seine Testprotokolle sind die Basis für Korrekturmaßnahmen des Entwicklerteams sowie für die Fortschrittskontrolle und das Berichtswesen des Testmanagers.

7.2 Die Positionierung von Testen im Software-Entwicklungsprozess

In den meisten Entwicklungsmodellen für Software-Entwicklung hat Testen eine fixe Position im Entwicklungsprozess. Allerdings ist Testen keine Aktivität, die mit dem Ende der Implementierung beginnt und mit dem Gebrauch durch den Endbenutzer aufhört. In einem gut durchdachten (und auch gelebten) Software-Entwicklungsprozess beginnen die Test-Aktivitäten schon sehr früh, nämlich mit Überlegungen zur Testbarkeit des entworfenen Systems. Zusätzlich

müssen Tests während der früheren Phasen als der Testphase in der Software-Entwicklung geplant und vorbereitet werden. Ansonsten werden diese Aktivitäten unnötigerweise Teil des kritischen Pfades und verzögern so unweigerlich das Projekt

Testen ist somit ein kontinuierlicher Prozess, der den Software-Entwicklungsprozess vom Anfang bis zum Ende begleitet und während der Test-Phase zur Hauptaktivität wird. Die meisten Prüf-Aktivitäten, wie sie im Deming-Kreis oder PDCA-Zyklus (siehe Kapitel 3) vorgeschlagen werden, sind Test-Aktivitäten. Diese beinhalten Reviews und Checks, Low-Level-Tests und High-Level-Tests als größte Kategorien.

Der Testprozess folgt selbst dem Deming-Kreis. Er muss auch geplant, ausgeführt, geprüft und an neue Umstände angepasst werden. Daher macht es auch Sinn, Testen als aus Phasen bestehende Prozedur einzuführen, d.h. als Testprozess mit einem Phasenmodell. Diese Phasen sind vom Deming-Kreis abgeleitet. Zusätzlich macht es die Aufteilung des amorphen Terms „Test“, wie er in traditionellen Software-Entwicklunsmethoden zu finden ist, in kleinere Teile weniger wahrscheinlich, dass der Test vernachlässigt wird, wenn das Budget sowie der Zeitrahmen zu Ende gehen ("*test design will be given short shrift when the budget's meager and the schedule's tight*" [Beizer, 1984]).

Ein Phasenmodell ist bloß einer der Schritte in Richtung eines strukturierten Testprozesses. Jedes strukturierte Testkonzept besteht aus vier Eckpfeilern [Pol et al., 2000], wie in Abbildung 7.1 dargestellt:

- ein mit dem Entwicklungsprozess in Zusammenhang stehendes Phasenmodell (P) der Testaktivitäten
- eine gute organisatorische Einbettung (O)
- die richtige Infrastruktur und Tools (I)
- sowie brauchbare Techniken (T) für die Durchführung der Aktivitäten

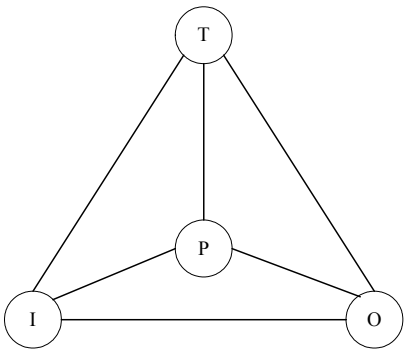

Abbildung 7.1: Die vier Eckpfeiler eines strukturierten Testkonzepts [Pol et al., 2000]

Testkosten verschlingen oft 30 bis 40 Prozent des Budgets in einem Software-Entwicklungsprojekt [Pol et al., 2000]. [Beizer, 1984] spricht sogar von 50 bis 80 Prozent. Das Risiko, dass ein Produkt von niedriger Qualität am Markt nicht akzeptiert wird, ist sehr hoch. Diese beiden Aspekte

rechtfertigen die Einführung eines strukturierten und verwaltbaren Testkonzepts. Grundlage eines strukturierten Testkonzepts sind die Hauptaktivitäten Planung, Vorbereitung und Durchführung. Als Faustregel für die Verteilung werden 20 % für die Planung, 40 % für die Vorbereitung und nur 40 % für die Durchführung gerechnet.

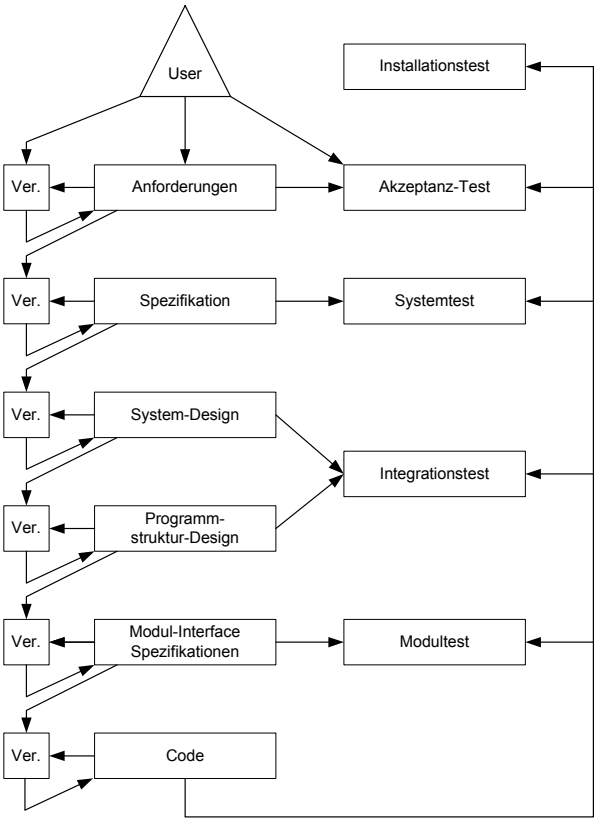

Abbildung 7.2: Die Positionierung von Testen im Entwicklungsprozess [Myers, 1979]

Eine weitere Sache ist wichtig, damit ein Test erfolgreich ist: dass das System für einen Test entworfen wurde. "*Good testing works best on good code and good design. And no testing technique can ever change garbage into gold.*" [Beizer, 1984].

Software-Projekte werden unter Einschränkungen im Zeitrahmen, von menschlichen Ressourcen und im Budget durchgeführt. Diese Einschränkungen können gegeneinander abgewägt, der Fokus untereinander ausgetauscht werden. Wenn das Ziel des Projekts ist, etwas zu produzieren, das ohne Absturz rennt, und das die Anforderungen zu erfüllen scheint, wird der Schwerpunkt bei der Codierung liegen. Das scheint in der heutigen, in kurzen Investitionszyklen denkenden Welt das Ziel von vielen Software-Projekten zu sein.

Ein vernünftigeres Ziel ist es, die Anforderungen eines formalen Tests zu erfüllen, und zufrieden stellende Leistung über eine Gewährleistungsfrist hinweg nach Einkauf des Produkts zu bieten. Mit diesem Ziel wandert der Schwerpunkt weg von bloß lauffähigem Code zu testbarem Code.

Ein noch längerfristiges Ziel ist ein Produkt mit langer Lebensspanne: dann wandert der Schwerpunkt hin zu wartbarem Code. Wartbarer Code setzt eine Basis von funktionierendem, sorgfältig getestetem Code und eine Codestruktur voraus, die nach Änderungen durch noch unbekannte zukünftige Programmierer leicht erneut verifiziert werden kann. Daher ist wartbarer Code in großem Ausmaß auch testbarer Code. Eine der Charakteristiken aller Industriezweige ist, dass, wenn sie reifen, ihre Ziele immer langfristiger werden. Computer und Software sind da keine Ausnahme, und so wie die Ziele für Software weiter in Richtung einer zufrieden stellenden Lebensspanne wandern, so wandert der Schwerpunkt von reiner Lauffähigkeit hin zu Testbarkeit.

Wenn Testautomatisierung zur Verringerung von Testkosten oder zur Erhöhung der Produktqualität eingesetzt werden soll, ist ein solch strukturierter Entwicklungs- und Testprozess eine grundlegende Voraussetzung.

Abhängig vom Fortschritt im Entwicklungsprozess können drei Stufen von Testaktivitäten unterschieden werden: Reviews, Low-Level-Tests und High-Level-Tests. Jeder Test einer Stufe ist ein Prozess, der in Phasen eingeteilt werden kann. Daher ist ein Phasenmodell für Testaktivitäten genauso wichtig wie für Entwicklungsaktivitäten. Je nach getesteten Qualitätskriterien können verschiedene Testtypen unterschieden werden. Jede Teststufe beinhaltet Tests von einem oder mehreren dieser Typen. Im folgenden wollen wir näher auf die Teststufen eingehen.

7.2.1 Der Modultest

Das Ziel eines *Modul-*, *Komponenten-* oder *Unit-Tests* ist es, Fehler in der Implementierung zu finden. Daher werden für Modultests hauptsächlich White-Box-Techniken verwendet.

Jedes vernünftig entworfene Software-Produkt kann in Subkomponenten, Module genannt, unterteilt werden. Ein Modul oder eine Unit besitzt die folgenden Eigenschaften [Beizer, 1984]:

1. Es ist das Werk eines Entwicklers, oder ist so geplant.
2. Es hat eine dokumentierte Spezifikation, die zumindest folgende Punkte beinhaltet:
   a. Definition der Eingaben
   b. Definition der Verarbeitung
   c. Definition der Ergebnisse
   d. Definition der Datenbasis
   e. Definition der Schnittstellen
3. Es ist ein sichtbares, identifizierbares Produkt, das in das Programm, von dem es einen Teil darstellt, explizit integriert wird.
4. Es kann kompiliert oder interpretiert werden, und kann getrennt von anderen Komponenten getestet werden, mit Ausnahme solcher Sub-Module, die es aufruft.

5. Es ist notwendig.

Im allgemeinen wird ein Modultest durch Übergabe von Daten an ein einzelnes Modul und Prüfung der Ausgabe durchgeführt. Der Schwerpunkt liegt auf der Tatsache, dass eine einzelne Komponente gestestet wird. Eingaben, die im Endprodukt von anderen Komponenten kommen, müssen entweder automatisch an die Schnittstelle des Moduls gereicht werden oder manuell.

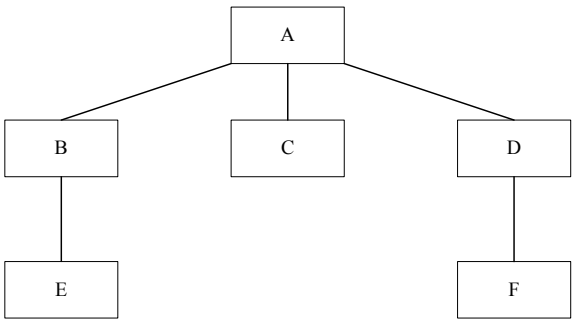

Abbildung 7.3: Beispiel: Aufruf-Hierarchie eines 6-Modul-Programms [Myers, 1979]

Der Test eines jeden Moduls braucht ein spezielles Treiber-Modul und eine variable Anzahl an Stub-Modulen. Ein Treiber-Modul ist ein kleines Modul, das nur dazu dient, Eingabedaten an das zu testende Modul zu schicken. Ein Stub-Modul wird anstelle eines echten Moduls aufgerufen. Um z.B. das Modul B in Abbildung 7.3 zu testen, müssen zuerst Testfälle definiert werden, und dann deren Eingabedaten für Modul B über ein Treiber-Modul an Modul B geschickt werden. Alternativ zum Treiber-Modul kann ein Test-Werkzeug verwendet werden. Das Treiber-Modul sollte dem Tester auch die Ergebnisse von B anzeigen. Zusätzlich muss etwas da sein, das den Kontrollfluss fortsetzt, wenn Modul B das Modul E aufruft. Das wird durch ein Stub-Modul erreicht, das die Funktion von Modul E simulieren muss [Myers, 1979].

In den meisten Fällen führen die Programmierer – eher unstrukturierte – Tests ihrer eigenen Module durch. Dabei wird der Grundsatz, dass nicht Programmierer ihre eigenen Programme testen sollten, verletzt. Allerdings wird das gerechtfertigt durch den hohen Aufwand, den Tester zum Verständnis jedes einzelnen Moduls und dessen Struktur aufbringen müssten, durch die normalerweise geringe Qualität in frühen Phasen, und durch die Verzögerungen und hohen Kosten, die mit einem richtigen Fehlermeldungssystem verbunden sind [Pol et al., 2000].

Allerdings sollte diese Art von Tests nicht mit Modultests verwechselt werden. [Beizer, 1984] nennt Testen wie oben beschrieben „Privates Testen“. Es ist im allgemeinen unstrukturiert und undokumentiert, aber dennoch notwendig. Fehler, die während der privaten Tests gefunden werden, bleiben privat, d.h. nur der Programmierer erfährt davon.

Im Gegensatz dazu stellt ein Unit-Test einen „öffentlichen Test“ dar, der von anderen Personen als dem Entwickler des Moduls durchgeführt wird. Ein guter Modul-Test ist in Phasen eingeteilt, wird geplant, durchgeführt und dokumentiert. Er wird so beschrieben, dass zumindest andere Programmierer in der Lage sind, ihn auszuführen. Seine Fehler werden gesammelt, korrigiert, auf ihre Ursachen hin analysiert, und sie dienen als Basis für Verbesserungsmaßnahmen – genauso wie Tests aller anderen Stufen.

7.2.2 Der Integrationstest

Das Ziel des Integrations-Tests ist es, Fehler im Programm zu finden, die dem Systementwurf entstammen. Die meisten Arten von Integrationstests sind White-Box-Tests.

Bei einem Integrationstest vereinen Programmierer einzelne Module zu größeren Strukturen. Tester prüfen dann die Interaktion dieser Module. Das Endziel ist es, alle Module zu einer Struktur zu verbinden, die dem Gesamtsystem entspricht.

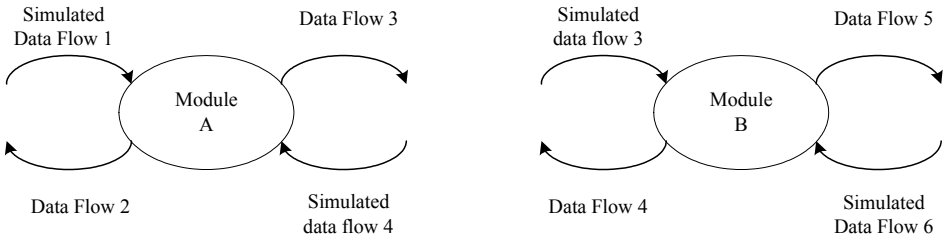

Abbildung 7.4: Zwei alleinstehende Module in ihrer Testumgebung

Es gibt zwei grundlegende Möglichkeiten, wie man Integrationstests angehen kann [Myers, 1979]. Bei der *non-inkrementellen* oder „*Big-Bang*“ Integration werden zuerst alle Module allein stehend getestet (siehe Abbildung 7.4), und dann alle auf einmal zum Gesamtsystem zusammengefügt. Der alternative Ansatz ist *inkrementelles Testen*. Anstatt alle Module allein stehend zu testen, wird das nächste zu testende Modul zuerst mit der Menge der bereits getesteten Module verbunden (siehe Abbildung 7.5). Datenflüsse werden durch Stubs und Treiber simuliert.

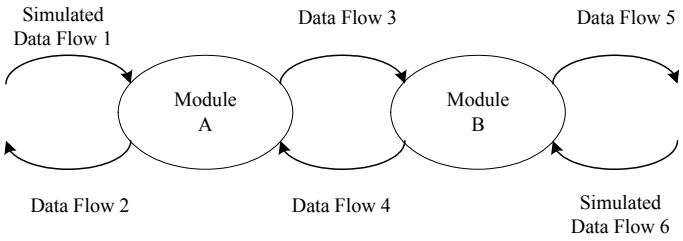

Abbildung 7.5: Zwei verbundene Module in ihrer Testumgebung

Inkrementelles Testen ist dem non-inkrementellen Testen aus folgenden Gründen vorzuziehen:

- Non-Inkrementelles Testen ist aufwendiger. Das Programm in Abbildung 1.3 3 benötigt fünf Treiber und fünf Stubs. Inkrementelles Testen verlangt entweder fünf Treiber oder fünf Stubs.
- Programmfehler, die durch nicht zusammenpassende Schnittstellen oder falsche Annahmen eine Schnittstelle betreffend verursacht wurden, werden früher erkannt wenn inkrementelles Testen verwendet wird.

- Debugging ist bei inkrementellem Testen einfacher, weil ein auftretender Fehler meist in Beziehung mit dem zuletzt hinzugefügten Modul steht. Bei non-inkrementellem Testen ist die Fehler-Lokation nicht so schnell bestimmbar.
- Inkrementelles Testen resultiert in sorgfältigeren Tests. Fehler in anderen Modulen können bei non-inkrementellen Tests Fehler in einem gegebenen Modul überdecken.

Man kann zwei verschiedene Ansätze für das inkrementelle Testen unterscheiden: *top-down* und *bottom-up*.

Top-Down-Strategie. Die Top-Down-Strategie startet mit dem obersten Modul in der Programmhierarchie. Danach kann ein Modul hinzugefügt werden, wenn zumindest eines seiner aufrufenden Module bereits getestet wurde. Zwei Richtlinien sollten beachtet werden

- Wenn es kritische Sektionen im Programm gibt, entwirf die Integrationsreihenfolge so, dass diese kritischen Teile so früh wie möglich hinzugefügt werden.
- Entwirf die Reihenfolge der Integration so, dass I/O-Module so früh wie möglich hinzugefügt werden.

Bottom-Up-Strategie. Die Bottom-Up-Strategie beginnt mit den untersten Modulen im Programm. Wenn diese Module getestet sind, kann ein Modul getestet werden, wenn alle von ihm aufgerufenen Module bereits getestet wurden.

Vorteile	**Nachteile**
Vorteilhaft, wenn schwere Probleme in der obersten Hierarchiestufe des Programms auftreten.	Stub-Modules müssen erzeugt werden
Sobald I/O-Funktionen integriert sind, ist die Darstellung von Testfällen einfacher.	Stub-Module sind meist komplexer als es zuerst den Anschein hat.
Frühes Programmskelett erlaubt Demonstrationen und fördert die Moral.	Bevor I/O-Funktionen integriert sind, kann die Darstellung von Testfällen in Stubs schwierig sein.
	Manche Testbedingungen können sehr schwer oder unmöglich erreichbar sein
	Beobachtung der Testausgaben ist schwieriger
	Erlaubt einem zu glauben, dass Design und Test überlappbar sind Verführt einen zur Annahme, dass Tests bestimmter Module bereits abgeschlossen sind

Tabelle 7.1: Vor- und Nachteile der Top-Down Integration nach [Myers, 1979]

Vorteile	Nachteile
Vorteilhaft, wenn schwere Probleme in der untersten Hierarchieebene des Programmsauftreten.	Treiber-Module müssen erzeugt werden.
Testbedingungen sind einfacher zu schaffen.	Das Programm als Gesamtheit existiert nicht bis das letzte Modul hinzugefügt ist.
Beobachtung der Testausgaben ist einfacher.	

Tabelle 7.2: Vor- und Nachteile der Bottom-Up Integration nach [Myers, 1979]

Wie man in Tabelle 7.1 und Tabelle 7.2 sehen kann, hat keine der beiden Strategien einen entscheidenden Vorteil gegenüber der anderen. Allerdings scheinen die schweren Konsequenzen des vierten Nachteils beim Top-Down-Testen der Bottom-Up-Strategie einen Vorsprung zu geben. Der sicherste Weg zu einer Entscheidung ist, die Faktoren der Tabelle im Kontext des zu testenden Programms abzuwägen [Myers, 1979].

[Beizer, 1984] schlägt vor: „*bottom-up the small; top-down the controls; big bang the backbone; and refine*“.

7.2.3 Der Systemtest

Das Ziel des Systemtests ist, Fehler im Programm, die aus der Analyse bzw. der Systemspezifikation stammen, zu finden Systemtests verwenden sowohl Black-Box- als auch White-Box-Testmethoden. Obwohl die Komponenten des Systems schon in früheren Teststufen ausgiebig getestet worden sein sollten, ist noch nicht sicher, ob das Gesamtsystem die geforderten Qualitätskriterien erfüllt. “*Bridges don't fall down because of bad steel, but because of bad architecture*” [Beizer, 1984]. Systemtests umfassen funktionale Tests, Stress-, Last- und Performance-Tests, Konfigurationstests, Sicherheitstests und statische Tests [Beizer, 1984].

Der Systemtest ist der erste Test, der das komplett zusammengefügte Gesamtsystem betrifft. Er stellt die umfangreichste Teststufe dar, weil hier alle relevanten Qualitätskriterien getestet werden müssen, und der Bereich für die Komplexität von Fehlern sehr weit ist. Fehler, die im Modul- oder Integrationstest gefunden werden, können leicht auf einzelne Module zurückgeführt werden. Fehler, die im Systemtest gefunden werden, können einer großen Anzahl von Modulen entstammen und sind daher sehr schwierig zu finden und zu korrigieren. Aus diesem Grund sollten auch die Low-Level-Tests so intensiv wie möglich durchgeführt werden. Die meisten am Markt verfügbaren Test-Tools unterstützen den Systemtest.

7.2.4 Der Regressionstest

Wenn Fehler korrigiert werden, können neue Fehler ins System eingebracht werden. Regressionstests sind diejenigen Tests, die nach Verbesserungen oder Korrekturen jedweder Art am Programm durchgeführt werden. Der Zweck eines Regressionstests ist es, herauszufinden, ob die Änderung zu Regression an anderen Teilen oder Aspekten des Programms geführt hat. Er wird für gewöhnlich durch erneute Ausführung einer bestimmten Untermenge aller Testfälle des Systemtests durchgeführt. Regressionstests sind wichtig, weil Änderungen und Fehlerkorrekturen

meist viel fehleranfälliger sind als die originalen Programmteile (genauso wie die meisten Tippfehler in Zeitungen das Ergebnis von Änderungen in letzter Minute und nicht im Originaltext sind) [Myers, 1979].

Theoretisch müssen alle Komponenten getestet werden, in denen ein Fehler korrigiert wurde, und alle Komponenten, die damit verbunden sind, und alle damit verbundenen Komponenten und so fort. Das bedeutet, dass für einen Regressionstest beinahe ein gesamter Systemtest durchgeführt werden müsste. In den wenigsten Fällen sind die damit verbundenen hohen Kosten zu rechtfertigen. Daher werden in der Praxis zur Verminderung der Kosten meist nur die geänderten Komponenten und vielleicht die „nächsten Nachbarn“ getestet.

Tester können Regressionstests nach Tests jeder Teststufe durchführen; aus diesem Grund beinhaltet Abbildung 7.2 keine Regressionstests.

Funktionale Test-Tools (siehe Kapitel 11) sind für gewöhnlich bei Regressionstests am effektivsten.

7.2.5 Der Akzeptanztest

Das Ziel des Akzeptanz- oder Abnahmetests ist es, dass der Kunde zu zeigen versucht, dass das gelieferte System die ursprünglichen Anforderungen nicht erfüllt. Der Kunde führt daher selbst Tests aus, welche die für ihn wichtigsten Qualitätskriterien betreffen, und vergleicht das Programm mit den ursprünglichen Anforderungen. Aufgrund der Anwendersicht sind Akzeptanztests hauptsächlich Black-Box-Tests.

7.2.6 Der Installationstest

Bei der Installation vieler Software-Systeme muss eine Vielzahl von Optionen durch den Benutzer ausgewählt werden, müssen Dateien und Bibliotheken zugeordnet und registriert werden, muss das System mit anderen Programmen verknüpft werden und muss eine gültige Hardware-Konfiguration vorhanden sein. Der Zweck eines Installationstests ist es, während dieses Installationsprozesses auftretende Fehler aufzudecken.

Unter anderem sollten die Testfälle des Installationstest überprüfen, ob eine zusammenpassende Menge an Optionen selektiert wurde, ob alle Komponenten des Systems existieren, ob alle Dateien angelegt oder kopiert wurden und den nötigen Inhalt haben, und ob die Hardware-Konfiguration angemessen ist [Myers, 1979].

Installationstests sollten im Idealfall von der Organisation, die das System entwickelt hat, als Teil des Systems ausgeliefert werden, damit sie nach der Installation gestartet werden können.

7.3 Phasen-Modell für den Testprozess

Tests von jeder Stufe sind Prozesse und können in Phasen eingeteilt werden. [Pol et al., 2000] schlagen ein Phasenmodell aus fünf Stufen vor (Abbildung 7.6).

Planung & Verwaltung. Zeitplanung und administrative Tätigkeiten im Testprozess, Erstellung von Teststrategie, Testplan, Berichten, Testdokumentation und detaillierte Planung.

Vorbereitung. Prüfung von System und Dokumentation, Festlegung der zu testenden Komponenten, Auswahl der Testfall-Spezifikationsmethoden, Spezifikation (und Setup) der Infrastruktur.

Spezifikation. Definition der Testfälle mittels Testfall-Spezifikationsmethoden, Definition der initialen Datenbank, Definition der Testskripts, Prüfung des Testobjekts und Setup der Infrastruktur (falls nicht schon getan).

Durchführung. Ausführung des Vorbereitungstests, Einrichtung der initialen Datenbank, Ausführung der Tests, Prüfung und Auswertung der Ergebnisse, Erstellung von Fehlerberichten, Update von Testfällen und Checklisten.

Abschluss. Sichern der Testware, Bewertung des Testobjekts und des Testprozesses, Abschlussbericht.

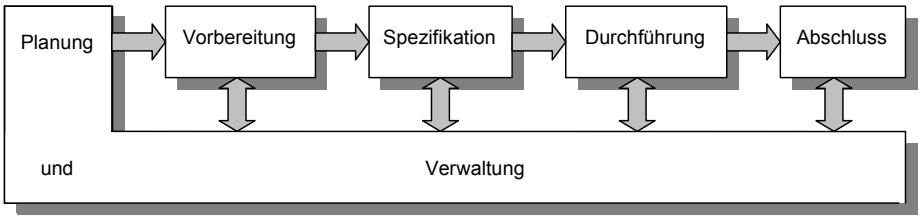

Abbildung 7.6: Phasenmodell für den Testprozess nach [Pol et al., 2000]

7.3.1 Phase Planung und Verwaltung

Wie Abbildung 7.6 zeigt, ist die Phase Planung & Verwaltung die erste Phase und begleitet in Folge alle anderen Phasen im Testprozess. Die wichtigsten Produkte dieser Phase sind der Testplan, der definiert wird, bevor alle anderen Phasen beginnen, laufende Testberichte und Templates für verschiedene Dokumente, die in späteren Phasen des Testtools erstellt werden, z. B. Testprotokolle, Fehlerberichte und Testfallspezifikationen.

Tätigkeiten in der Phase Planung und Verwaltung umfassen zum Beispiel:

- Definition der Teststrategie
- Sammlung der Systemdokumentation
- Einrichtung von Verwaltung und Organisation
- Einrichtung der Testdokumentation: Definition von Templates für Fehlerberichte, Testfallspezifikationen, etc.
- Verwaltung des Testprozesses, der Infrastruktur und der Testprodukte
- Erstellung von Berichten
- Erstellung und Aktualisierung des Testplans

- Detaillierte Planung

Testplan. Es ist möglich, alle Teststufen in einem einzigen Testplan einzubetten, oder einen Testplan für jede Teststufe zu schreiben. Wenn mehrere Testpläne existieren, ist es empfehlenswert, einen „Master-Testplan“ zu erstellen, der die Abhängigkeiten zwischen den Teststufen beschreibt. Ein guter Testplan beinhaltet ([Pol et al., 2000], [Myers, 1979]):

Ziele und Teststrategie. Die Ziele jeder Testphase müssen definiert werden, und die Strategie, sie zu erreichen.

Testendekriterien. Kriterien für das Ende jeder Testphase müssen festgelegt werden. Die zwei häufigsten Kriterien sind (1) Ende wenn die Zeit abläuft und (2) Ende wenn alle Testfälle ohne Fehler durchlaufen. Ein besseres, aber nicht häufig benutztes Kriterium ist, zu beenden, wenn eine bestimmte Abdeckung erreicht ist.

Zeitplan. Zeitpläne werden für die Aktivitäten jeder Phase benötigt. Vor allem sollten sie die Zeitpunkte beinhalten, zu denen Testfälle entworfen, erstellt und ausgeführt werden.

Erklärung der Aufwandsschätzung. Der Zeitplan basiert auf einer Aufwandsschätzung, die hier erklärt werden soll, sodass Änderungen an der Teststrategie, am getesteten Objekt oder in den zugrunde liegenden Annahmen berücksichtigt werden können.

Verantwortlichkeiten. Für jede Phase sollen die Personen, die Testfälle entwerfen, erstellen und ausführen, sowie diejenigen, welche die entdeckten Fehler korrigieren, identifiziert werden (d.h. alle Rollen in Abbildung 7.8).

Testfallbibliotheken und Standards. In einem großen Projekt sind systematische Methoden zur Identifikation, zur Erstellung und zur Ablage von Testfällen nötig.

Tools. Die benötigten Tools müssen identifiziert werden, inklusive einem Plan, wer sie entwickeln oder beschaffen soll, und wann sie benötigt und eingesetzt werden.

Benötigte Infrastruktur und Hardware-Konfiguration. Wenn spezielle Hardware-Konfigurationen oder Geräte benötigt werden, ist ein Plan vonnöten, der die Anforderungen beschreibt, wie sie eingehalten werden sollen, und wann die Hardware benötigt wird.

Integration. Teil des Testplanes ist eine Definition, wie das Programm zusammengesetzt werden soll.

Fehlerverfolgung. Mittel für die Meldung von Fehlern sowie für die Verfolgung des Korrekturfortschritts müssen definiert werden (siehe Abbildung 7.8). Idealerweise werden dafür Werkzeuge verwendet. Diese ermöglichen neben der Fehlerverfolgung die rasche Erkennung von fehleranfälligen Modulen und eine Abschätzung des Testfortschritts.

Regressionstests. Ein Plan für Regressionstests (d.h. wer, wie, wann) ist eine weitere Notwendigkeit.

Risiken und Gegenmaßnahmen. Risikomanagement ist genauso wichtig für Tests wie für das Gesamtprojekt.

Testberichte. Testberichte können folgende Information beinhalten:

- Was wurde bereits getestet?

- Was muss noch getestet werden?
- Wie viele Fehler wurden bisher gefunden?
- Von welchem Typ sind die gefundenen Fehler (Fehler in Analyse, Design, Implementierung oder Konfiguration, Anwenderfehler, Änderungsanforderungen (engl. Change Requests), etc.)?
- Wie viele Fehler wurden pro Subsystem gefunden?
- Können Trends festgestellt werden, z. B. in der Anzahl der Fehler?
- Welche Aufwände sind bisher angefallen?

7.3.2 Phase Vorbereitung

In dieser Phase werden vorbereitende Aktivitäten für die Phasen Testfall-Spezifikation und Testdurchführung erledigt. Diese umfassen beispielsweise:

- Prüfung der Systemdokumentation mittels Checklisten
- Definition der Subsysteme entweder anhand von Design-Dokumenten oder anhand einer Analyse des Systems.
- Prüfung der Testbarkeit jedes Subsystems mittels Checklisten
- Definition von zu testenden Modulen in den Subsystemen unter Beachtung von Abhängigkeiten zwischen Funktionen.
- Zuweisung von Testspezifikationstechniken zu den Test-Modulen, in Abhängigkeit von der Teststrategie.
- Spezifikation der Infrastruktur: sowohl Hardware (Zielplattform) als auch Software (Testdatenbank, Test-Tools, etc.)
- Möglicherweise Einrichtung der Infrastruktur.

7.3.3 Phase Spezifikation

Zur Phase Spezifikation gehören alle Tätigkeiten, die mit der Testfall-Spezifikation zu tun haben. Dies sind beispielsweise:

- Spezifikation der Testfälle je nach Methode, die jeder Testeinheit zugewiesen wurde. Testfall-Spezifikationsmethoden werden in Kapitel 9 und 10 beschrieben.
- Definition der initialen Test-Datenbank und der Eingabefiles aus den in den Testfällen verwendeten Daten.
- Definition von konkreten und ausführbaren Sequenzen von Testaktivitäten und Bedingungen (Test-Skripts) für jeden Testfall.
- Definition der Reihenfolge, in welcher die Test-Skripts ausgeführt werden sollen.

- Prüfung des Testobjekts und der Infrastruktur mittels Checklisten.
- Definiere Vorbereitungstest als Untermenge der „echten“ Testfälle. Dieser Vorbereitungstest wird zu beginn der Phase Durchführung ausgeführt.
- Einrichtung der Infrastruktur, wie sie in der Vorbereitungsphase definiert wurde, falls nicht bereits geschehen.

7.3.4 Phase Durchführung

Dies ist die Phase, in welcher der eigentliche Test stattfindet. Gemeinsam mit den Testaktivitäten werden Fehler korrigiert und nachgetestet. Um Entwicklungs- und Testaktivitäten effizient zu koordinieren, muss ein strukturierter Prozess zur Fehlermeldung und Fehlerverfolgung als Teil des Configuration Management definiert werden.

Tätigkeiten in der Phase Durchführung

Mögliche Tätigkeiten in der Durchführungsphase umfassen:

- Prüfung mittels Checklisten und Programmausführung, ob das Testobjekt und die Infrastruktur eine bestimmte minimale Reife besitzen.
- Befüllen der initialen Datenbank und von Eingabe-Dateien, wie in der Spezifikations-Phase definiert.
- Tests (erneut) durchführen: Ausführung von Test-Skripts und/oder Prüfungen anhand von Checklisten. Verfolgung der Ergebnisse gemäß der im Testplan definierten Prozedur.
- Meldung von Fehlern: nur reproduzierbare Fehler, alle Schritte und Vorbedingungen (soweit bekannt) beschreiben
- Testergebnisse prüfen und bewerten:
    - Fehler bei der Testdurchführung (Test muss wiederholt werden)
    - Fehler in Testfall-Spezifikation
    - Fehler im getesteten Objekt
    - Fehler in dieser Version nicht relevant (verschoben)
    - Unzulänglichkeiten in der Testumgebung
    - Inkonsistenzen oder Unklarheiten in der System-Dokumentation
    - Duplikat eines bereits bekannten Fehlers
    - Change request
- Aktualisierung von Test-Skripts und Checklisten

Configuration Management (CM)

Eine sehr wichtige Vorbedingung für die Durchführung von Tests ist ein funktionierendes Configuration Management (CM). Tester dürfen nicht Teile des Systems testen, die sich noch in der Entwicklung befinden. Entwickler dürfen nicht mit dem Testprozess interferieren. Daher müssen Mechanismen für das Versions-Management und Management von Test-Umgebungen eingerichtet werden. Diese Mechanismen sind Teil des Configuration Managements, das für die Verwaltung aller Produkte und Tools zuständig ist, die im Projekt erzeugt oder verwendet werden. CM ist die „logistische Drehscheibe“ in einem Software-Projekt, wie Abbildung 7.7 zeigt [Zopf, 2001]. Es stellt die Schnittstelle dar zwischen

- Software-Entwicklung,
- Projekt-, Produkt- und Qualitätsmanagement,
- Test-, Integrations- und Freigabemanagement,
- sowie der Wartung.

CM beantwortet Fragen wie die folgenden: [Kit, 1995]:

- Wie sieht unsere derzeitige Software-Konfiguration aus?
- Wie ist deren Status?
- Wie kontrollieren wir Änderungen an unserer Konfiguration?
- Welche Änderungen wurden an unserer Software durchgeführt?
- Haben Änderungen weitere Auswirkungen auf unsere Software?

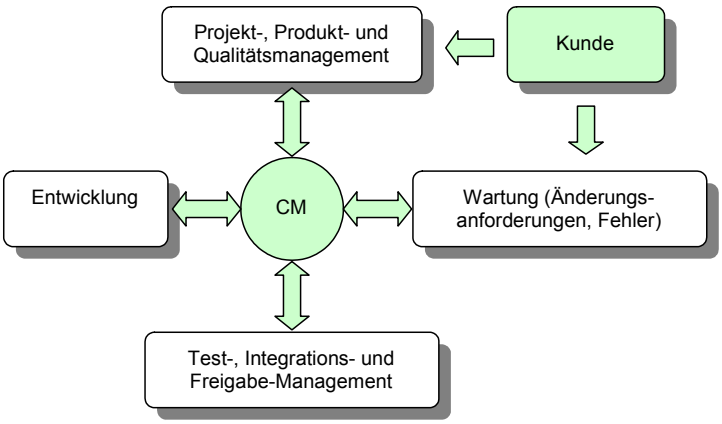

Abbildung 7.7: Konfigurations-Management als „logistische Drehscheibe“ [Zopf, 2001]

Eine ausführliche Ausarbeitung des CM wäre hier zu viel. Die Lektüre von Büchern über Configuration Management wird jedenfalls empfohlen.

Fehlerverfolgung

Abbildung 7.8 zeigt einen möglichen Kontrollfluss für die Fehlerverfolgung. Darin werden vier verschiedene Rollen unterschieden: Tester (oder auch Anwender), Test-Manager (häufig der Projektleiter), Entwickler und CM (Configuration Management). Die Personen, die diese Rollen einnehmen, müssen nicht unbedingt verschieden sein. Außerdem können auch ganze Organisationen anstelle von Einzelpersonen diese Rollen einnehmen.

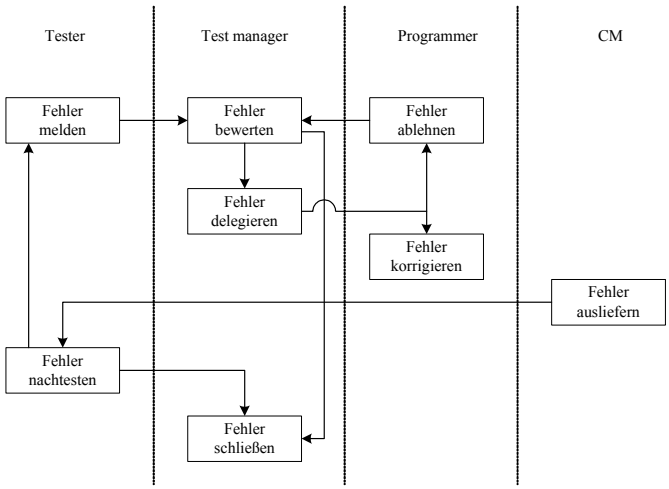

Abbildung 7.8: Workflow für die Fehlerverfolgung

Fehler melden. Der Tester oder Anwender führt einen Test aus und meldet einen aufgetretenen Fehler. Hier wird dem Fehler eine eindeutige Identifikation zugewiesen. Die Meldung findet am besten über ein Fehlerverfolgungstool wie z.B. Lotus Notes, Rational Clearquest oder eine eigene Datenbank statt; sie kann aber auch in Papier-Form erfolgen. Der Tester weist dem Fehler eine Kategorie zu (leichter, schwerer, betriebsverhindernder oder testverhindernder Fehler, ...). Der Fehler muss so ausführlich wie möglich beschrieben werden (Vorbedingungen, Aktionen, Beschreibung des Fehlers, Screenshots, Anmerkungen zur Reproduzierbarkeit, ...) Es sollten nur reproduzierbare Fehler gemeldet werden. Es ist zudem wichtig, dass nur ein Fehler pro Fehlermeldung beschrieben wird, nicht mehrere in derselben.

Fehler bewerten. Der Test-Manager bzw. der Projekt-Manager klassifiziert den Fehler (Fehler, Change Request, Duplikat, unwichtig und verschoben, ...) und weist ihm eine Priorität zu. Duplikate werden sofort geschlossen. Für Change Requests gibt es mehrere Möglichkeiten:

- gleich behandeln wie Fehler
- nicht behandeln und Fehler schließen
- alle Change Requests sammeln und einbauen, wenn Finanzierung geklärt ist

Unter Umständen müssen Spezialisten, z.B. Entwickler, zur Unterstützung bei der Bewertung herangezogen werden.

Fehler delegieren. Fehler, die korrigiert werden müssen, werden einem Entwickler zugewiesen.

Fehler ablehnen. Unter Umständen kann der Entwickler, dem der Fehler zugewiesen wurde, diesen nicht, nur mit hohem Aufwand, oder nur mit höherem Aufwand als ein anderer Entwickler korrigieren. In diesem Fall kann der Entwickler den Fehler ablehnen. Der Test-Manager muss in diesem Fall den Fehler neu bewerten und neu delegieren.

Fehler korrigieren. Der Entwickler korrigiert den Fehler und klassifiziert ihn bezüglich seiner Ursache (Software, Hardware, Integration, Konfiguration, etc.).

Korrektur ausliefern. Das Configuration Management liefert die Fehlerkorrektur entweder an den Kunden oder in eine interne Testumgebung aus. In beiden Fällen wird eine neue Programmversion erstellt. In den meisten Fällen werden wegen der mit der Auslieferung verbundenen Kosten mehrere Fehlerkorrekturen gesammelt ausgeliefert.

Fehler nachtesten. Der Tester bzw. Anwender versucht, den Fehler nach der Korrektur wieder zu produzieren. Das Ergebnis dieses Tests meldet er an den Test-Manager.

Fehler schließen. Der Test-Manager schließt den Fehler und beendet somit dessen Bearbeitungszyklus. Unter Umständen führt er auch Aktivitäten für spätere Evaluierungen aus.

7.3.5 Phase Abschluß

Wenn eine bestimmte Bedingung erfüllt ist, ist der Test zu Ende. Solch eine Bedingung kann z.B. eine bestimmte Qualitätsstufe oder, wie in den meisten Fällen, ein Termin sein. Manche Aktivitäten müssen noch durchgeführt werden, um ein sauberes und kontrolliertes Ende des Tests zu erreichen. Diese umfassen beispielsweise:

- Bewertung des Test-Objekts, z.B. Zuweisung eines Werts zu jedem Qualitätskriterium, basierend auf Fehlerstatistiken und Trends beim Auftreten von Fehlern.
- Bewertung des Testprozesses für Verbesserungen im nächsten Projekt.
- Verfassung eines Abschlussberichts mit Fehlerstatistiken, Trends, Kosten, etc.
- Sicherung der Testware, d.h. der Datenbanken, Testfallspezifikationen, Berichte etc.

7.4 Organisation

„*Jeder Testprozess, dessen Organisation eine unzureichende Qualität aufweist, läuft unweigerlich auf ein Fiasko hinaus. Die Beteiligung vieler verschiedener Unternehmensbereiche* (Siehe Abbildung 7.9), *die gegensätzlichen Interessen, die Unvorhersehbarkeit, die komplizierten Verwaltungsaufgaben, der Mangel an Erfahrung(-szahlen), und der Zeitdruck führen dazu, dass die Einrichtung und die Verwaltung der Testorganisation keine einfache Aufgabe ist.*“ Ansätze, wie ein Test zu organisieren ist, können in [Kit, 1995] and [Dustin et al., 1999] gefunden werden.

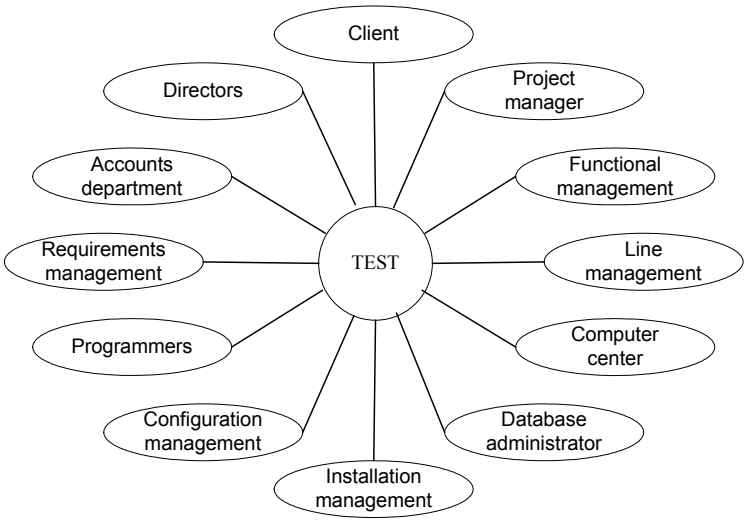

Abbildung 7.9: Die am Test beteiligten Unternehmensbereiche [Pol et al., 2000]

Ein Test-Manager muss die folgenden Aspekte in der Organisation eines strukturierten Tests berücksichtigen [Pol et al., 2000]:

- Betrieblicher Testprozess: Größe und Zusammensetzung des Test-Teams, Wissen und Fähigkeiten im Team, Zeitpläne, …
- Strukturelle Testorganisation: Organisationsstruktur (flach, hierarchisch, etc.), ein Test-Team für alle Projekte oder jeweils ein eigenes, ...
- Test-Management: Verwaltung des Testprozesses, der Testinfrastruktur, der Testprodukte, ...
- Personal und Ausbildungen: Wissen und Fähigkeiten jedes einzelnen, Kurse, Motivation, ...

7.5 Infrastruktur und Tools

Die Infrastruktur für das Testen beinhaltet alle notwendigen Einrichtungen und Mittel, um den Anforderungen entsprechend testen zu können. Test-Umgebungen werden benötigt, um den Test durchzuführen, und Test-Tools bieten zusätzliche Unterstützung [Pol et al., 2000].

7.5.1 Testumgebung

Traditionell stehen für das Testen drei Testumgebungen zur Verfügung:

- die Entwicklungsumgebung für die Low-Level-Tests,
- die (besser) verwaltbaren Systemumgebungen für High-Level-Tests,

- sowie produktive oder „produktionsnahe“ Umgebungen, hauptsächlich für die Abnahmetests.

Testumgebungen müssen früh im Entwicklungsprozess definiert werden. Die Einrichtung der Umgebungen erfolgt durch das Configuration Management.

7.5.2 Test-Tools

Zur Unterstützung aller Phasen des Test-Prozesses sind Tools verfügbar. Die wichtigsten darunter sind diejenigen für Planung und Verwaltung, für Fehler-Verfolgung, für automatische Testfallgenerierung und für die automatische Testdurchführung. Die Tools für Planung und Verwaltung sind größtenteils dieselben wie für die Verwaltung jedes anderen Prozesses, wie z.B. Planungs-Software, Spreadsheets oder Programme zur Risikoanalyse.

Eine Vielzahl an Tools, die die unterschiedlichen Phasen des Testprozessesunterstützen, stehen zur Verfügung. Am wichtigsten sind dabei Tools zur Planung und Kontrolle, für das *Bug Tracking*, zur *automatischen Erzeugung von Testfällen* und zur *automatischen Testausführung*. Die Tools zur Planung und Kontrolle sind weitestgehend dieselben wie für andere Prozesse, z.B. Scheduling Software, Tabellenkalkulationen oder Programme zur Risikoanalyse. Kapitel 11 beschreibt die verschiedenen erhältlichen Tools im Detail.

7.6 Zusammenfassung

Testen ist ein bedeutender Bestandteil aller Entwicklungsmodelle für Software. Anders als man vermuten könnte ist Testen jedoch keine Tätigkeit, die mit der Fertigstellung der Implementierung beginnt und mit der Auslieferung an den Kunden aufhört, sondern eine den gesamten Entwicklungsprozess begleitende Maßnahme. Testen ist ein zeitaufwändiger und kostspieliger Aspekt der Software-Entwicklung, der oft 30 bis 40 Prozent des Gesamtbudgets eines Software-Entwicklungsprojekts verschlingt [Pol et al., 2000]. Bereits während der Implementierung ist darauf zu achten, testbaren Code zu produzieren, wobei besonders wartbarer Code in großem Ausmaß auch testbarer Code ist. Um Tests erfolgreich durchführen zu können wird eine entsprechende Testinfrastruktur benötigt. Dies umfasst die Testumgebung, die benötigt wird, um Tests durchführen zu können, und Test-Tools zur Unterstützung des Testprozesses.

Ein funktionierendes Configuration Management ist eine wichtige Vorbedingung für die Durchführung von Tests und ist für die Verwaltung aller Produkte und Tools zuständig, die im Projekt erzeugt oder verwendet werden. Um Testaktivitäten effizient zu koordinieren, muss ein strukturierter Prozess zur Fehlermeldung und Fehlerverfolgung definiert werden. Tester dürfen nicht Teile des Systems testen, an denen noch entwickelt wird, und Entwickler dürfen nicht mit dem Testprozess interferieren.

Abhängig vom Fortschritt im Entwicklungsprozess können drei Stufen von Testaktivitäten unterschieden werden: Reviews, Low-Level-Tests und High-Level-Tests. Verschiedene Testtypen können je nach getesteten Qualitätskriterien unterschieden werden: Modultests, Integrationstests, Systemtests, Regressionstests, Akzeptanztests und Installationstests.

Wie auch bei anderen umfangreichen Aufgaben im Software-Entwicklungsprozess ist es auch beim Testen ratsam, den Testprozess in verschiedene Phasen und Rollen zu unterteilen. Tests von jeder Stufe sind Prozesse und können in folgende Phasen unterteilt werden: Planung & Verwaltung, Vorbereitung, Spezifikation, Durchführung und Abschluss. Die erste Phase, Planung & Verwaltung begleitet alle folgenden Phasen im Testprozess.

In den meisten Testprozessen können folgende Rollen identifiziert werden: Testmanager, Testingenieur, Testentwickler und Tester. Diese Rollen müssen nicht unbedingt von verschiedenen Personen eingenommen werden.

7.7 Literaturreferenzen

[Beizer, 1984] Boris Beizer: "System Testing and Quality Assurance", van Nostrand, 1984, ISBN 0-442-21306-9

[Kaner et al., 1999] Cem Kaner, Jack Falk, Hung Quoc Nguyen: “Testing Computer Software", Wiley, 1999, ISBN 0-471-35846-0

[Kit, 1995] Edward Kit: "Software Testing in the Real World", Addison-Wesley, 1995, ISBN 0-201-87756-2

[Myers, 1979] Glenford J. Myers: "The Art of Software Testing", Wiley, 1979, ISBN 0-471-04328-1

[Pol et al., 2000] Martin Pol, Tim Koomen, Andreas Spillner: “Management und Optimierung des Testprozesses: ein praktischer Leitfaden für Testen von Software, mit TPI und TMap”, dpunkt.verlag, 2000, ISBN 3-932588-65-7

[Thaller1997] Georg Erwin Thaller: „Software-Test: Verifikation und Validation“, Heise, 1997, ISBN 3-88229-183-4)

[LIGGES2002] Peter Liggesmeyer „Software-Qualität: Testen, Analysieren und Verifizieren von Software“, Spektrum Akademischer Verlag, 2002, ISBN 3-8274-1118-1)

7.8 Übungen und Fragen