2 Grundlagen des Softwaretestens

Dieses einleitende Kapitel erklärt die Grundbegriffe des Softwaretestens, die in den weiteren Kapiteln vorausgesetzt werden. Wichtige Begriffe werden zusätzlich an dem praxisnahen Fallbeispiel VSR-II-System veranschaulicht, das im gesamten Buch immer wieder zur Illustration und Motivation des Lehrstoffs verwendet wird. Die sieben Grundsätze des Testens werden vorgestellt. Hauptteil des Kapitels ist der Testprozess, der mit seinen einzelnen Aktivitäten detailliert erläutert wird. Am Ende des Kapitels wird auf psychologische Probleme beim Testen eingegangen und wie diese umgangen bzw. verringert werden können.

2.1 Begriffe und Motivation

Anforderungen an die Qualität

Bei der Herstellung eines Industrieprodukts werden die entstehenden Produkte üblicherweise daraufhin kontrolliert, ob sie den gestellten Anforderungen entsprechen. Es wird meist durch Stichproben geprüft, ob das Produkt die geforderte Aufgabe löst. Je nach Produkt gibt es auch unterschiedliche Anforderungen an die Qualität der Lösung. Erweist sich ein Produkt als fehlerhaft, so müssen ggf. Korrekturen im Produktionsprozess oder in der Konstruktion erfolgen.

Software ist immateriell.

Was allgemein für die Herstellung eines Industrieprodukts gilt, trifft entsprechend für die Produktion bzw. Entwicklung von Software zu. Die Prüfung der Teilprodukte bzw. des Endprodukts gestaltet sich allerdings schwieriger, da die erstellte Software immateriell ist und daher nicht »greifbar« und eine Prüfung deshalb nicht »handfest« durchgeführt werden kann. Eine optische Prüfung ist nur sehr eingeschränkt durch intensives Lesen der Entwicklungsdokumente möglich.

Fehlerhafte Software führt zu Problemen.

Software, die unzuverlässig oder inkorrekt arbeitet, kann zu erheblichen Problemen führen. Hierzu gehören der Verlust von Geld und Zeit, die Schädigung des Geschäftsrufs bis hin zu Verletzungen von Personen oder sogar deren Tod. Beispiele für gravierende Softwarefehler finden sich oft in der aktuellen Tagespresse, wenn etwa die »Autopilot«-Software eines teilautonom fahrenden Autos fehlerhaft ist und zu spät oder falsch reagiert.

Testen liefert eine Einschätzung der Qualität.

Es ist daher wichtig, die Qualität der Software zu prüfen, um das Risiko eines Softwareausfalls oder eines Softwarefehlers zu minimieren. Das Testen von Software liefert eine Einschätzung der Qualität und verringert die Risiken beim Einsatz der Software, da mögliche Fehler während des Testens aufgedeckt werden können. Dem Testen von Software kommt somit eine sehr wichtige, aber auch sehr schwierige Aufgabe zu.

Beispiel: Risiko durch Softwarefehler

Jedes Release des VSR-II-Systems unseres Fallbeispiels muss vor Auslieferung und Einsatz angemessen geprüft werden, um mögliche Fehler vorab zu erkennen und beheben zu können. Führt das System beispielsweise Bestellvorgänge falsch aus, könnte dies für Kunden und Händler, aber auch für den Autokonzern unter Umständen einen großen finanziellen Schaden und/oder Imageverlust zur Folge haben. Jedenfalls birgt die Nichterkennung eines solchen Fehlers ein hohes Risiko beim Einsatz der Software.

Testen ist eine stichprobenhafte Prüfung.

Oft wird unter Testen die (im Allgemeinen stichprobenartige) Ausführung[1] der zu prüfenden Software (Testobjekt) auf einem Rechner verstanden. Dazu werden einzelne Testfälle ausgeführt, d.h., das Testobjekt wird mit Testdaten versehen und ausgeführt. Die anschließende Bewertung prüft, ob das Testobjekt die geforderten Eigenschaften erfüllt und sich konform zu den Anforderungen verhält.[2]

Testen ist mehr als die Ausführung von Testfällen auf dem Rechner.

Zum Testen gehört aber weit mehr als die Ausführung von Testfällen. Software zu testen ist ein Prozess – der Testprozess, der unterschiedliche Aktivitäten umfasst. Die Testdurchführung, inklusive der Prüfung der Ergebnisse, ist nur eine der Aktivitäten. Weitere Aktivitäten im Testprozess sind die Testplanung, die Analyse, das Design und die Realisierung von Tests. Die Anfertigung von Berichten über den Testfortschritt und über die Testergebnisse sowie die Beurteilung der Qualität eines Testobjekts und die Risikobewertung sind zusätzliche Aufgaben. Die Testaktivitäten werden je nach Lebenszyklus der Software unterschiedlich organisiert und durchgeführt. Testaktivitäten und Testdokumentation werden häufig vertraglich zwischen Auftraggeber und Auftragnehmer oder durch gesetzliche oder Firmenstandards festgelegt. Eine detaillierte Beschreibung der einzelnen Aktivitäten im Testprozess folgt in Abschnitt 2.3 und in Abschnitt 6.3.

1. Hier ist das dynamische Testen (s. Kap. 5) gemeint. Beim statischen Test (s. Kap. 4) wird das Testobjekt nicht ausgeführt.
2. Es ist nicht möglich, die korrekte Umsetzung aller Anforderungen durch Testen nachzuweisen (s. u.).

Statisches und dynamisches Testen

Neben den Tests, die auf dem Rechner ausgeführt werden (dynamische Test, s. Kap. 5), können und sollen auch Dokumente wie Anforderungsspezifikation, User Stories und Quellcode so früh wie möglich einer Prüfung unterzogen werden. Derartige Tests werden statische Tests (s. Kap. 4) genannt. Je früher Fehler in den Dokumenten gefunden und behoben werden, je besser ist es für die weitere Entwicklung der Software, da nicht mit fehlerbehafteten Dokumenten weitergearbeitet wird.

Verifizierung und Validierung

Testen umfasst aber nicht nur die Prüfung, ob sich das System entsprechend den Anforderungen, User Stories oder anderen Spezifikationen verhält, sondern auch die Prüfung, ob sich das System entsprechend den Vorstellungen und Wünschen der Nutzer bzw. Anwender in der Betriebsumgebung verhält, ob die beabsichtigte Nutzung möglich ist bzw. das System seinen Zweck erfüllt. Testen beinhaltet somit auch die Validierung (s.a. 7. Grundsatz: Trugschluss: Keine Fehler bedeutet ein brauchbares System in Abschnitt 2.1.6).

Kein umfangreiches System ist fehlerfrei.

Ein fehlerfreies Softwaresystem gibt es derzeit nicht und wird es in naher Zukunft wahrscheinlich auch nicht geben, sobald das System einen gewissen Grad an Komplexität und Umfang an Programmzeilen umfasst. Häufig liegt ein Fehler darin begründet, dass sowohl während der Entwicklung als auch beim Testen der Software gewisse Ausnahmesituationen nicht bedacht bzw. nicht überprüft wurden. Sei es das Schaltjahr, das nicht richtig berechnet wird, oder die nicht berücksichtigten Randbedingungen, wenn es um Zeitverhalten oder Ressourcenbedarf geht. Es ist daher durchaus üblich – oder oft auch unumgänglich – dass Software und Systeme in Betrieb genommen werden, obwohl Fehler bei bestimmten Eingabekonstellationen auftreten. Auf der anderen Seite arbeiten aber sehr viele Softwaresysteme in ganz unterschiedlichen Bereichen zuverlässig tagein, tagaus.

Fehlerfreiheit nicht durch Testen erreichbar

Selbst wenn alle ausgeführten Tests keinen einzigen Fehler mehr aufdecken, kann (außer bei sehr sehr kleinen Programmen) nicht ausgeschlossen werden, dass es zusätzliche Tests gibt, die weitere Fehler aufzeigen würden. Fehlerfreiheit kann mit Testen nicht nachgewiesen werden.

2.1.1 Fehlerbegriff

Testbasis als Grundlage

Eine Situation kann nur dann als fehlerhaft eingestuft werden, wenn vorab festgelegt wurde, wie die erwartete bzw. als korrekt spezifizierte Situation aussehen soll. Zur Bestimmung der korrekten Situation werden die Anforderungen an das zu testende System(teil), aber auch weitere Informationen herangezogen. In diesem Zusammenhang wird von der Testbasis gesprochen, gegen die getestet wird und die als Grundlage der Entscheidung dient, ob ein korrektes oder fehlerhaftes Verhalten vorliegt.

Was gilt als Fehler?

Ein Fehler ist somit die Nichterfüllung einer festgelegten Anforderung, eine Abweichung zwischen dem Istverhalten (während der Ausführung der Tests oder des Betriebs festgestellt) und dem Sollverhalten (in der Spezifikation, den Anforderungen oder den User Stories festgelegt). Wann liegt aber ein nicht anforderungskonformes Verhalten des Systems vor?

Im Gegensatz zu physischen Systemen entstehen Fehler in einem Softwaresystem nicht durch Alterung oder Verschleiß. Jeder Fehler ist seit dem Zeitpunkt der Entwicklung in der Software vorhanden. Er kommt jedoch erst bei der Ausführung der Software zum Tragen.

Fehlerwirkung

Für diesen Sachverhalt wird der Begriff Fehlerwirkung verwendet. Der englische Fachbegriff hierfür lautet »Failure«. Weitere Bezeichnungen sind Fehlfunktion, äußerer Fehler oder Ausfall. Beim Test der Software oder auch erst bei deren Betrieb wird eine Fehlerwirkung für den Tester oder Anwender nach außen sichtbar. Zum Beispiel ist ein Ausgabewert falsch oder das Programm stürzt ab.

Fehlerzustand

Zwischen dem Auftreten einer Fehlerwirkung und deren Ursache muss unterschieden werden. Eine Fehlerwirkung hat ihren Ursprung in einem Fehlerzustand (»Fault«) der Software. Dieser Fehlerzustand wird auch als Defekt oder innerer Fehler bezeichnet. Auch das englische Wort »Bug« ist gebräuchlich. Dies ist beispielsweise eine falsch programmierte oder vergessene Anweisung im Programm.

Fehlermaskierung

Es ist durchaus möglich, dass ein Fehlerzustand durch einen oder mehrere andere Fehlerzustände in anderen Teilen des Programms kompensiert wird (Fehlermaskierung). Eine Fehlerwirkung tritt in diesem Fall erst dann zutage, nachdem der oder die maskierenden Fehlerzustände korrigiert worden sind. Korrekturen können somit zu Seiteneffekten führen.

Ein Problem ist, dass ein Fehlerzustand nicht zu einer Fehlerwirkung führen muss. Eine Fehlerwirkung kann gar nicht, einmal oder immer und somit für alle Benutzer des Systems auftreten. Eine Fehlerwirkung kann weit entfernt vom Fehlerzustand zum Tragen kommen. Ein Beispiel ist eine kleine Verfälschung von gespeicherten Daten, die bei der Programmausführung erst zu einem späteren Zeitpunkt aufgedeckt wird.

Fehlhandlung

Ursache für das Vorliegen eines Fehlerzustands ist wiederum die vorausgegangene Fehlhandlung einer Person, wie z. B. eine fehlerhafte Programmierung durch den Entwickler. Der englische Begriff hierfür ist »Error«.

Fehlhandlungen können aus vielfältigen Gründen entstehen. Einige typische Fehlhandlungen bzw. Gründe (bzw. Grundursache) für Fehlhandlungen sind:

- Menschen machen Fehler – wir alle!
- Ein hoher Zeitdruck ist vorhanden – was in Softwareprojekten sehr häufig vorkommt.
- Die Komplexität der umzusetzenden Aufgabe, der Architektur, des Designs sowie des Programmtextes ist sehr hoch.
- Es gibt Missverständnisse zwischen den Projektbeteiligten – oft ein unterschiedliches Verständnis bzw. eine Auslegung der Anforderungen und weiterer Dokumente.
- Es gibt Missverständnisse über die Systeminteraktionen (systeminterne und systemübergreifende Schnittstellen), da deren Anzahl bei größeren Systemen oft sehr hoch ist.
- Die Komplexität der genutzten Technologien ist hoch oder es werden neue, bei den Projektbeteiligten (noch) unbekannte Technologien verwendet, die noch nicht richtig verstanden und daher falsch angewendet werden.
- Es liegt Unerfahrenheit oder unzureichende Ausbildung bei den Projektbeteiligten vor.

Eine Fehlhandlung einer Person führt zu einem Fehlerzustand in einem Programmstück, was zu einer Fehlerwirkung führt, die außen sichtbar ist und durch Testen aufgezeigt werden soll (s. Abb. 2–1, Debugging s. u.). Statische Tests (s. Kap. 4) können im Programmtext direkt Fehlerzustände aufdecken.

Fehlerwirkungen können aber auch durch Umweltbedingungen ausgelöst werden, wie Strahlung, Magnetismus etc., oder auch durch Umweltverschmutzung mit den entsprechenden Auswirkungen auf die Firmware und Hardware. Diese Art Fehler wird hier nicht behandelt.

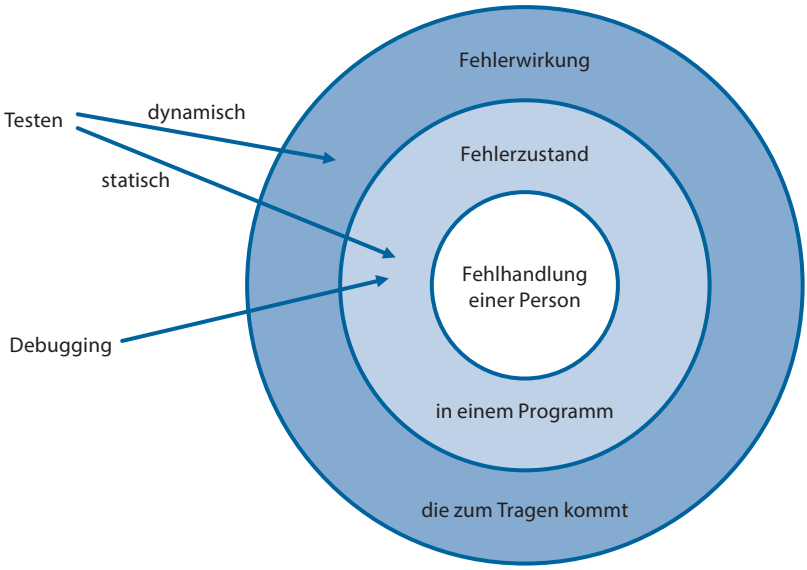

Abb. 2–1
Zusammenhang zwischen Fehlhandlung, Fehlerzustand und Fehlerwirkung

Falsch positives Ergebnis und falsch negatives Ergebnis

Nicht jedes unerwartete Ergebnis der Tests ist auch immer eine Fehlerwirkung. Es kann vorkommen, dass ein Testergebnis eine Fehlerwirkung anzeigt, obwohl der Fehlerzustand bzw. die Ursache für die Fehlerwirkung nicht im Testobjekt liegt. Dies wird als »falsch positives Ergebnis« (»false-positive Result«) bezeichnet. Die umgekehrte Situation kann ebenfalls vorkommen, dass Fehlerwirkungen nicht auftreten, obwohl die Tests diese hätten aufdecken sollen. Dies wird als »falsch negatives Ergebnis« (»false-negative Result«) bezeichnet. Bei jeder Auswertung von Testergebnissen ist somit zu beachten, ob eine der beiden Möglichkeiten vorliegt. Es gibt noch zwei weitere Ergebnisse: »richtig positiv« (Fehlerwirkung durch den Testfall aufgedeckt) und »richtig negativ« (erwartetes Verhalten bzw. Ergebnis des Testobjekts mit dem Testfall nachgewiesen). Nähere Ausführungen hierzu finden sich in Abschnitt 6.4.1.

Aus Fehlern lernen

Konnten Fehlerzustände aufgedeckt und die Fehlhandlungen ermittelt werden, die zu den Fehlerzuständen führten, dann lohnt es sich, mögliche Ursachen zu analysieren, um daraus zu lernen und in Zukunft gleiche oder ähnliche Fehlhandlungen zu vermeiden. Die so gewonnenen Erkenntnisse können zur Prozessverbesserung genutzt werden, um das Auftreten von zukünftigen Fehlhandlungen und damit Fehlerzuständen zu verringern oder zu verhindern.

Beispiel: Unklare Anforderung als Ursache von Softwarefehlern

Über das Teilsystem VSR-II-*EasyFinance* kann sich der Kunde verschiedene Optionen zur Finanzierung seines neuen Fahrzeugs berechnen und vorschlagen lassen. Der bei einer Kreditfinanzierung vom System verwendete Zinssatz wird aus einer im System hinterlegten Zinssatztabelle entnommen. Für Fahrzeuge, die im Rahmen von werblichen Sonderaktionen angeboten und verkauft werden, können davon abweichend andere Zinssätze gelten.

Im neuen VSR-II soll zusätzlich folgende Anforderung realisiert werden:

REQ: Wenn der Kunde der »Online-Bonitätsprüfung« zugestimmt und diese absolviert hat, dann reduziert VSR-II-*EasyFinance* den Kreditzinssatz gemäß der im System hinterlegten Zinssatz-Bonus-Tabelle.

Der Autor der Anforderung hat allerdings vergessen klarzustellen, dass diese Reduktion nicht erlaubt ist, wenn es um ein Fahrzeug geht, das in einer Sonderaktion verkauft wird. Entsprechend wurde dieser Fall in den Tests des ersten Release nicht überprüft. Kunden aus Sonderaktionen erhielten daher online Kreditangebote mit zu niedrigem Zinssatz angeboten und beschwerten sich, als die erste Kreditabrechnung einen höheren Zinssatz auswies.

2.1.2 Testbegriff

Testen ist nicht Debugging.

Um einen Fehlerzustand korrigieren zu können, muss der Fehlerzustand im Softwareprodukt lokalisiert werden. Bekannt ist zunächst nur seine Wirkung, aber nicht die genaue Stelle in der Software, die den Fehlerzustand beinhaltet. Das Lokalisieren und Beheben des Fehlerzustands ist Aufgabe des Softwareentwicklers und wird auch als »Debugging« (Fehlerbereinigung, Fehlerkorrektur) bezeichnet. Debugging und Testen werden oft gleichgesetzt, sind aber zwei völlig unterschiedliche und getrennte Aufgaben. Während Debugging das Ziel hat, Defekte bzw. Fehlerzustände zu lokalisieren, ist es Aufgabe des Tests, Fehlerwirkungen-gezielt aufzudecken (s.a. Abb. 2–1).

Fehlernachtest

Die Behebung des Fehlerzustands führt zur Qualitätsverbesserung des Produkts, da in den meisten Fällen keine neuen Fehlerzustände durch die Änderung hinzugefügt werden. Tests, die prüfen, ob die Korrekturmaßnahmen erfolgreich waren, werden als Fehlernachtests bezeichnet. Oft sind Tester für diese Fehlernachtests verantwortlich, während Softwareentwickler neben dem Debugging auch die zugehörigen Komponententests durchführen. In der agilen Entwicklung und in einigen anderen Lebenszyklen kann diese Aufgabentrennung aufgehoben sein.

In der Praxis kommt es aber leider vor, dass durch die Korrektur eines Fehlerzustands ein oder sogar mehrere neue Fehlerzustände »hineinprogrammiert« werden. Der neue Fehlerzustand kann dann bei einer ganz anderen Eingabekonstellation zur Wirkung kommen. Sol-

che unbeabsichtigten Seiteneffekte erschweren den Test und bedingen, dass nach Änderungen nicht nur die Tests zu wiederholen sind, die den Fehlerzustand zur Wirkung gebracht haben, sondern auch weitere Tests, die mögliche Seiteneffekte aufdecken könnten.

Testziele

Testen – und hier sind sowohl statisches als auch dynamisches Testen gemeint – verfolgt mehrere Ziele:

- Die qualitative Bewertung von Arbeitsergebnissen wie Anforderungsspezifikation, User Stories, Design und Programmtext
- Der Nachweis, dass alle spezifischen Anforderungen vollständig umgesetzt sind und dass das Testobjekt so funktioniert, wie es die Nutzer und andere Interessenvertreter (Stakeholder) erwarten
- Informationen zur Verfügung stellen, damit die Stakeholder die Qualität des Testobjekts fundiert einschätzen können und somit Vertrauen in die Qualität des Testobjekts schaffen[3]
- Die Höhe des Risikos bei mangelnder Qualität der Software kann durch Aufdeckung (und Behebung) von Fehlerwirkungen verringert werden. Die eingesetzte Software enthält dann weniger unentdeckte Fehlerzustände.
- Analysieren des Programms und der Dokumente, um Fehlerzustände zu vermeiden und vorhandene zu erkennen (und dann zu beheben)
- Analysieren und Ausführen des Programms mit dem Ziel, Fehlerwirkungen nachzuweisen
- Informationen über das Testobjekt zu erhalten, um zu entscheiden, ob das Systemteil für die Integration mit weiteren Systemteilen freigegeben (einchecken, »commit«) werden kann
- Aufzeigen, dass das Testobjekt konform zu vertraglichen, rechtlichen oder regulatorischen Anforderungen oder Standards ist und/oder Überprüfung der Einhaltung der Anforderungen oder Standards durch das Testobjekt

Ziele abhängig vom Kontext

Abhängig vom Kontext können die Ziele des Testens variieren. Je nach Softwareentwicklungsmodell (z.B. agil oder konventionell) und Teststufe (Komponententest, Integrationstest, Systemtest, Abnahmetest, s. Abschnitt 3.4) können unterschiedliche Ziele verfolgt werden.

3. Werden bei einem gründlichen Test wenige oder keine Fehlerwirkungen nachgewiesen, dann führt das zu einer Erhöhung des Vertrauens in die getestete Software.

So steht beim Test einer Komponente im Vordergrund, möglichst viele Fehlerwirkungen aufzudecken, um die zugrunde liegenden Fehlerzustände frühzeitig zu identifizieren (»Debugging«) und zu beheben. Ein weiteres Ziel kann es sein, die Tests so zu wählen, dass die erreichte Überdeckung des Programmtexts (s. Abschnitt 2.3.1) möglichst hoch ist.

Ein Ziel des Abnahmetests ist der Nachweis, dass das System sowohl wie erwartet benutzt werden kann als auch wie erwartet arbeitet und somit die nicht funktionalen und funktionalen Anforderungen erfüllt. Ein weiteres Ziel ist die Bereitstellung von Informationen über das Risiko einer Systemfreigabe für die Stakeholder, damit sie eine fundierte Entscheidung über die Freigabe (ja oder nein) treffen können.

Exkurs: Benennung von Tests

Es gibt verwirrend viele Bezeichnungen für verschiedene Arten von Softwaretests. Einige werden im Rahmen der Darstellung der verschiedenen Teststufen (s. Abschnitt 3.4) genauer erklärt. Dieser Exkurs soll etwas Systematik in die Begriffsvielfalt bringen. Dazu ist es hilfreich, folgende Kategorien, nach denen Tests bezeichnet werden können, zu unterscheiden:

1. **Testziel**
   Die Benennung einer Testart erfolgt aufgrund des Testzwecks (z. B. Lasttest).
2. **Testmethode/Testverfahren**
   Der Test wird nach der Methode/dem Verfahren benannt, die zur Spezifikation oder Durchführung der Tests eingesetzt wird (z. B. zustandsbasierter Test, s. Abschnitt 5.1.3).
3. **Testobjekt**
   Der Test wird nach der Art des Testobjekts, das getestet wird, benannt (z. B. GUI-Test – Test der grafischen Bedienoberfläche oder DB-Test – Test der Datenbank).
4. **Teststufe**
   Der Test wird nach der Stufe des zugrunde liegenden Softwareentwicklungsmodells benannt (z. B. Systemtest).
5. **Testperson**
   Der Test wird nach dem Personenkreis bezeichnet, der die Tests durchführt (z. B. Entwicklertest, Anwendertest).
6. **Testumfang**
   Tests werden nach dem Umfang unterschieden (z. B. partieller Regressionstest).

Nicht hinter jedem Begriff verbirgt sich also eine neue oder andere Art von Test. Vielmehr wird, je nachdem unter welchem Blickwinkel die Testarbeiten betrachtet werden, einer der aufgeführten Aspekte in den Vordergrund gerückt.

2.1.3 Testartefakte und ihre Beziehungen

In den vorherigen Abschnitten wurden bereits einzelne Artefakte beim Testen genannt und kurz erklärt. Im Folgenden wird ein Überblick über die beim dynamischen Testen erforderlichen bzw. zu erstellenden Artefakte gegeben.

Testbasis

Grundlage für alle Überlegungen zum Testen ist die Testbasis. Wie oben bereits erwähnt, werden alle Dokumente als Testbasis bezeichnet, die herangezogen werden können, um eine Entscheidung treffen zu können, ob beim Testen eine Fehlerwirkung aufgetreten ist oder nicht. Die Testbasis legt somit das Sollverhalten des Testobjekts fest. Aber auch der gesunde Menschenverstand oder Fachwissen können als »Teile« der Testbasis angesehen und für die Entscheidung herangezogen werden. In den meisten Fällen liegt ein Anforderungsdokument, eine Spezifikation oder eine User Story vor, die als Testbasis dient.

Testfall und Testlauf

Auf Grundlage der Testbasis werden die Testfälle erstellt. Ein Testfall wird auf dem Rechner ausgeführt, d.h., das Testobjekt wird mit entsprechenden Testdaten versehen und zur Ausführung gebracht – ein Testlauf wird durchgeführt. Die Ergebnisse des Testlaufs werden geprüft und es wird entschieden, ob eine Fehlerwirkung vorliegt, also eine Abweichung zwischen erwartetem Sollergebnis bzw. Sollverhalten und dem sich nach dem Testlauf ergebenen Istergebnis bzw. Istverhalten. Um Testfälle ausführen zu können, sind meist Vorbedingungen einzuhalten, z.B. muss die zu verwendende Datenbank nutzbar und mit entsprechenden Daten gefüllt sein.

Testbedingung

Da jeder Testfall die Testbasis nicht als Ganzes prüfen kann, muss eine Fokussierung auf bestimmte Aspekte erfolgen. Aus der Testbasis sind sogenannte Testbedingungen[4] abzuleiten, die für das Erreichen bestimmter Testziele (s.o.) relevant sind. Eine Testbedingung (»Test Condition«) ist durch ein oder mehrere Testfälle zu prüfen. Eine Testbedingung kann eine Funktion, eine Transaktion, ein Qualitätsmerkmal oder ein strukturelles Element einer Komponente oder eines Systems sein. Konkrete Beispiele für eine Testbedingung sind die Preisberechnung beim VSR-II-System, die Kombination der Fahrzeugmodelle (s. Abschnitt 5.1.5), das »Look and feel« der Benutzungsoberfläche oder das Antwortzeitverhalten des zu testenden Systems.

4. Auch als einzelne Testanforderung oder Testsituation bezeichnet.

Testelement

Ebenso kann »auf der anderen Seite« ein Testobjekt in der Regel nicht als Ganzes getestet werden. Es werden sogenannte Testelemente ermittelt, die durch die einzelnen Testfälle getestet werden. Zur Testbedingung Preisberechnung beim VSR-II-System ist das entsprechende Testelement die Methode `calculate_price()` (s. Abschnitt 5.1.1), die mit den entsprechenden Testfällen getestet wird. Die Testfälle werden unter der Verwendung von Testverfahren (s. Kap. 5) spezifiziert.

Testsuite und Testausführungsplan

Testfälle einzeln auszuführen ist wenig sinnvoll. Testfälle werden in Testsuiten zusammengefasst, die in einem Testzyklus ausgeführt werden. Im Testausführungsplan ist der zeitliche Ablauf der Ausführung der Testsuiten festgelegt.

Testskript

Ein Testskript wird implementiert, das die Testsuite automatisiert ausführt. In den Testskripten ist die Reihenfolge der Testfälle mit allen erforderlichen Aktionen zur Herstellung der Vorbedingungen und zum Aufräumen nach der Durchführung realisiert. Werden Testsuiten nicht automatisiert durchgeführt, sind diese Informationen für den manuellen Test ebenfalls bereitzustellen (Testablauf, »Test Procedure«).

Protokoll

Die Ergebnisse der Testläufe sind zu protokollieren und zusammenfassend in einem Testbericht zu dokumentieren.

Testkonzept

Zu Beginn aller Überlegungen zum Testen eines Testobjekts ist ein Testkonzept zu erstellen, in dem alles festlegt wird, was für die Testdurchführung erforderlich ist (s. Abschnitt 6.2.1). Hierzu gehören u.a. die Auswahl der Testobjekte und Testverfahren, die Festlegung der Testziele und der Testberichterstattung ebenso wie die Koordination der einzelnen Testaktivitäten untereinander.

In der Abbildung 2–2 sind die Zusammenhänge zwischen den Artefakten dargestellt. Durch die Angabe der Aktivitäten des Testprozesses (s. Abschnitt 2.3) wird verdeutlicht, wann die Artefakte erstellt werden.

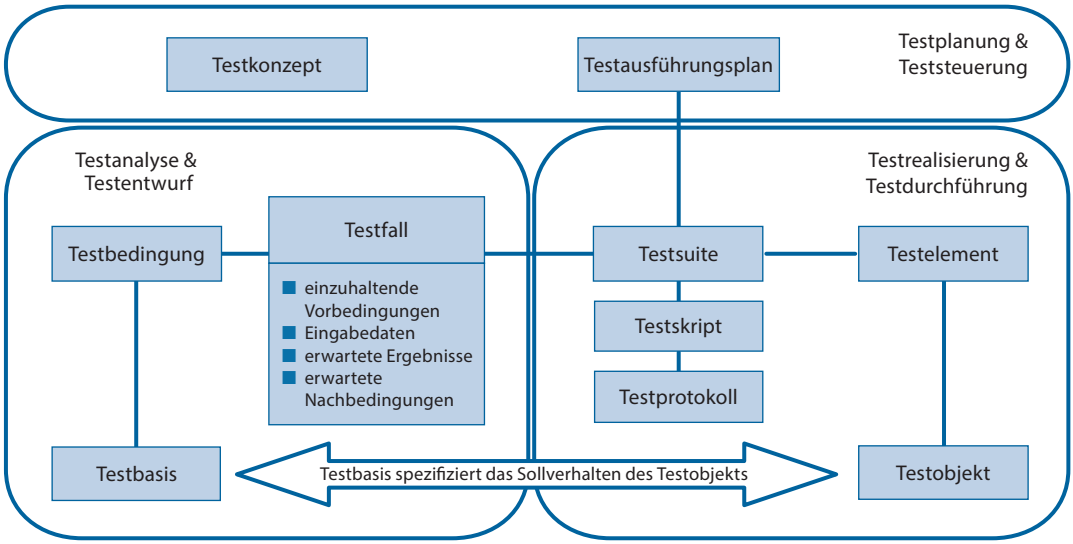

Abb. 2–2 *Testartefakte und ihre Beziehungen*

2.1.4 Aufwand für das Testen

Testaufwand abhängig vom Projekt(umfeld)

Das Testen nimmt einen großen Teil des Entwicklungsaufwands in Anspruch, auch wenn immer nur ein Teil aller denkbaren Tests – genauer aller denkbaren Testfälle – berücksichtigt werden kann. Eine allgemein gültige Aussage oder Empfehlung über die Höhe des zu investierenden Testaufwands ist jedoch schwierig, da dies stark vom Charakter des Projekts abhängt.[5]

Der Stellenwert des Testens – und damit der mögliche Aufwand für das Testen – wird oft auch durch das Zahlenverhältnis von Testern zu Entwicklern deutlich. In der Praxis sind folgende Verhältnisse anzutreffen: von einem Tester, der auf zehn Entwickler kommt, bis hin zu drei Testern pro Entwickler. Der Testaufwand bzw. das für das Testen investierte Budget divergiert in der Praxis sehr stark.

Beispiel: Testaufwand und Fahrzeugvarianten

Mit VSR-II kann ein Kaufinteressent sein Wunschfahrzeug am Bildschirm selbstständig konfigurieren. Welches Zubehör für welches Modell lieferbar ist und welche Optionen untereinander oder mit vorkonfektionierten Ausstattungspaketen kombiniert werden können, unterliegt vielfältigen Regeln.

Bei VSR konnte der Kaufinteressent Kombinationen auswählen, obwohl diese nicht lieferbar waren. Für VSR-II soll (u.a. als Konsequenz aus der

5. Abschnitt 6.2.5 behandelt das Thema detaillierter. Hier wird ein erster Einstieg in die Thematik gegeben.

QS/Test-Plan-Vorgabe »Funktionelle Eignung/DreamCar = high«, s.u.) sichergestellt werden, dass nicht erlaubte Kombinationen keinesfalls ausgewählt werden können.

Der für die Komponente »DreamCar« verantwortliche Product Owner möchte herausfinden, wie viel Testaufwand es erfordert, diesen Aspekt der *DreamCar*-Funktionalität möglichst vollständig zu testen. Dazu überschlägt er, wie viele Ausstattungskombinationen es maximal geben kann und identifiziert folgende Möglichkeiten:

10 Fahrzeugmodelle mit je 5 Motorvarianten, 10 Felgentypen mit Sommer- oder Winterbereifung, 10 Lackfarben mit Matt-, Glanz- oder Perleffekt-Beschichtung und 5 verschiedene Entertainment-Systeme. Alleine hieraus resultieren $10\times5\times10\times2\times10\times3\times5 = 150.000$ verschiedene Kombinationen. Wenn das Testen jeder Kombination nur 1 Sekunde dauert, wären schon 1,7 Tage dazu erforderlich.

Weitere 50 optionale Zubehörteile multiplizieren die Kombinationsanzahl nochmals mit 50 Fakultät und führen zu theoretisch möglichen rund $4{,}6\times10^{69}$ Kombinationen. Für jede dieser Kombinationen muss *DreamCar* entscheiden können, ob diese lieferbar und somit »anklickbar« ist.

Dem Product Owner ist intuitiv klar, dass sein Team nicht gegen jede Kombination testen muss, sondern stattdessen »nur« die Funktion sämtlicher Regeln, die definieren, was »nicht lieferbar« ist. Andererseits besteht aber das Risiko, dass *DreamCar* manche Kombination aufgrund irgendwelcher denkbarer Fehlerzustände fälschlicherweise doch als »lieferbar« klassifiziert (oder andersrum).

Welcher Testaufwand ist hier nun mindestens erforderlich oder wirtschaftlich maximal gerechtfertigt? Der Product Owner beschließt, bei Gelegenheit die Test/QS-Leiterin hierzu um Rat zu fragen. Ein Ausweg aus dem Dilemma bietet möglicherweise das kombinatorische Testen (s. Exkurs, Abschnitt 5.1.5).

Exkurs: Wann ist ein hoher Testaufwand vertretbar?

Ist ein hoher Testaufwand bezahlbar und gerechtfertigt? Die Gegenfrage von Jerry Weinberg lautet: »Im Vergleich zu was?« [DeMarco 93]. Er weist damit auf die zu bedenkenden Risiken bei fehlerhaften Softwaresystemen hin. Das Risiko wird aus der Eintrittswahrscheinlichkeit und der zu erwartenden Höhe des Schadens ermittelt. Im Test nicht gefundene Fehlerzustände können beim Einsatz der Software hohe Kosten verursachen.

Beispiel: Fehlerkosten

»Das mehrere hundert Millionen Euro teure Weltraumteleskop Hitomi geriet Ende März 2016 nach einer Verkettung von Softwarefehlern in zu schnelle Rotation und ging verloren. Die Software war fälschlicherweise von einer unerwünschten langsamen Rotation des Satelliten ausgegangen und versuchte, die scheinbare Drehung durch Gegenmaßnahmen zu kompensieren. Die Signale der redundanten Kontrollsysteme wurden falsch gedeutet, und schließlich wurde der Satellit immer stärker in Rotation versetzt, bis er wegen der zu groß werdenden Fliehkräfte schließlich zerbrach« (aus [URL: Fehlerkosten]).

Der Testaufwand muss in einem vernünftigen Verhältnis zum erzielbaren Ergebnis stehen.

»Testen ist ökonomisch sinnvoll, solange die Kosten für das Finden und Beseitigen eines Fehlers[6] im Test niedriger sind als die Kosten, die mit dem Auftreten eines Fehlers bei der Nutzung verbunden sind« [Pol 00]. Der Testaufwand muss daher immer in Abhängigkeit mit dem im Fehlerfall verbundenen Risiko und der Gefährdungsbewertung stehen. Für den entstandenen Schaden (Totalverlust) des Weltraumteleskops Hitomi hätten sehr viele Tests durchgeführt werden können.

Beispiel: Risiko und Schaden im Fehlerfall

Während der Kaufinteressent sein Wunschfahrzeug konfiguriert, zeigt *DreamCar* laufend den Kaufpreis der aktuellen Konfiguration an. Bestandskunden des Herstellers oder Interessenten, die sich vorher bei einem Händler ausgewiesen und autorisiert haben, können ihr Wunschfahrzeug online bestellen! Sobald sie »verbindlich bestellen« anklicken und ihre »PIN« eintippen, ist das Fahrzeug gekauft und bestellt. Nach der gesetzlichen Widerrufsfrist für Online-Käufe wird die gewählte Konfiguration dann automatisch an den Produktionsplanungsrechner übermittelt, der die Produktion einplant und initiiert.

Wird dabei vom System eine fehlerhafte Preisberechnung durchgeführt, kann der Kunde auf dem angezeigten Preis bestehen. Denn es ist ein verbindlicher Kaufvertrag zustande gekommen! Somit kann eine fehlerhafte Preisberechnung dazu führen, dass Tausende von Fahrzeugen zu einem zu geringen Preis verkauft werden. Der Gesamtschaden kann, je nachdem wie stark sich das VSR-II-System je Fahrzeug »verrechnet«, in die Millionen gehen. Dass jeder Vertrag manuell überprüft wird, ist keine Option. Denn der Sinn der Funktion ist ja gerade, dass der Hersteller mit VSR-II einen solchen 100 % automatisierten Online-Verkaufsprozess realisieren kann.

Testintensität und Testumfang in Abhängigkeit vom Risiko festlegen

Systeme oder Systemteile mit einem hohen Risiko müssen ausgiebiger getestet werden als solche, die im Fehlerfall keine großen Schäden verursachen.[7] Wobei die Risikoeinschätzung für die einzelnen Systemteile oder sogar für einzelne Fehlermöglichkeiten durchzuführen ist. Bei einem hohen Risiko im Fehlverhalten eines Systems oder Teilsystems ist für den Test ein höherer Aufwand zu veranschlagen als bei weniger kritischen System(teil)en. Die Intensität des Tests muss entsprechend hoch sein. Internationale Standards für die Produktion von sicherheitskritischen Systemen geben ein solches Vorgehen vor, indem sie die Verwendung von verschiedenen aufwendigen Verfahren zum Testen vorschreiben (z.B. [RTC-DO 178B] für den Luftfahrtbereich).

6. Bei den Kosten sind alle Aspekte zu berücksichtigen, wie die Auswirkungen von schlechter Publicity, gerichtliche Auseinandersetzungen usw., nicht nur die Kosten für Korrektur, erneuten Test, Verteilung und Austausch der korrigierten Software.
7. In Abschnitt 6.2.4 wird das Thema ausführlich behandelt.

Für die Herstellerfirma eines Computerspiels kann ein fehlerhaftes Speichern eines Spielstandes ein sehr hohes Risiko bedeuten (obwohl es keine direkten Schäden verursacht), da das fehlerhafte Spiel bei der Kundschaft nicht akzeptiert wird und zu hohen Absatzverlusten, möglicherweise bei allen Spielen der Firma, führen kann.

2.1.5 Testwissen frühzeitig und damit erfolgreich nutzen

Erfolgsfaktor Testen

Mit dem Testen – genauer mit den Überlegungen und vorbereitenden Arbeiten zum Testen – soll so früh wie möglich begonnen werden. Im Folgenden werden Beispiele aufgeführt, die die Vorteile verdeutlichen, wenn Tester mit entsprechendem Testwissen bei einzelnen Aktivitäten im Softwareentwicklungslebenszyklus involviert sind:

Enge Zusammenarbeit zwischen Entwicklern und Testern während der gesamten Softwareentwicklung

- Beteiligen sich Tester an der Prüfung der Anforderungen (z.B. durch Reviews, s. Abschnitt 4.2) oder an der Verfeinerung (»Refinements«) von User Stories und bringen ihr Testfachwissen ein, können Unklarheiten und Fehler in den Arbeitsprodukten aufgedeckt und behoben werden. Die Identifizierung und Behebung der fehlerhaften Anforderungen reduzieren das Risiko der Entwicklung falscher oder nicht testbarer Funktionen.
- Die enge Zusammenarbeit von Testern mit Systemdesignern während des Entwurfs des Systems kann das Verständnis jeder Partei für das Design und dessen Test erheblich verbessern. Dieses erhöhte Verständnis kann das Risiko grundlegender Konstruktionsfehler reduzieren und ermöglicht die frühzeitige Identifikation potenzieller Tests, z.B. zur Prüfung der Schnittstellen beim Integrationstest (s. Abschnitt 3.4.2).
- Arbeiten Entwickler und Tester während der Codeerstellung zusammen, kann das Verständnis auf beiden Seiten für den Code und dessen Test verbessert werden. Dieses reduziert das Risiko von Fehlerzuständen im Programmtext und auch von fehlerhaften Tests zur Prüfung des Programmtextes (falsch negatives Ergebnis, s. Abschnitt 6.4.1).
- Wenn Tester die Software vor deren Freigabe verifizieren und validieren, können weitere Fehlerzustände erkannt und behoben werden, die ansonsten unentdeckt geblieben wären. Dies erhöht die Wahrscheinlichkeit, dass die Software den Bedürfnissen der Interessenvertreter gerecht wird und die Anforderungen erfüllt.

Zusätzlich zu diesen Beispielen trägt das Erreichen der oben definierten Testziele zum allgemeinen Erfolg der Softwareentwicklung bzw. der Wartung bei.

2.1.6 Grundsätze des Testens

In den vorherigen Abschnitten wurde das Testen von Software behandelt. In diesem Abschnitt werden die wesentlichen Grundsätze des Testens zusammenfassend dargestellt, die sich aus den zurückliegenden Jahrzehnten herauskristallisiert haben und als generelle Leitlinien beim Testen angesehen werden.

1. **Grundsatz:**
   Testen zeigt die Anwesenheit von Fehlerzuständen
   Mit Testen wird das Vorhandensein von Fehlerwirkungen und damit von Fehlerzuständen nachgewiesen. Testen verringert – je nach Testaufwand und Intensität – die Wahrscheinlichkeit, dass noch unentdeckte Fehlerzustände im Testobjekt vorhanden sind. Mit Testen lässt sich jedoch nicht beweisen, dass keine Fehlerzustände im Testobjekt vorhanden sind. Selbst wenn keine Fehlerwirkungen im Test aufgezeigt wurden, ist dies kein Nachweis für Fehlerfreiheit oder Korrektheit.
2. **Grundsatz:**
   Vollständiges Testen ist nicht möglich
   Ein vollständiger Test, bei dem alle möglichen Eingabewerte und deren Kombinationen unter Berücksichtigung aller unterschiedlichen Vor- und Randbedingungen ausgeführt werden, ist nicht durchführbar, mit Ausnahme bei sehr trivialen Testobjekten. Tests sind immer nur Stichproben, und der Testaufwand ist deshalb nach Risiko und Prioritäten zu steuern.
3. **Grundsatz:**
   Frühes Testen spart Zeit und Geld
   Testaktivitäten – sowohl statische als auch dynamische – sollen im System- oder Softwarelebenszyklus so früh wie möglich beginnen und definierte Ziele verfolgen. Frühes Testen wird auch mit »shift left« bezeichnet. Durch frühzeitiges Prüfen werden Fehlerzustände frühzeitig erkannt. Es hilft dabei, im Softwareentwicklungslebenszyklus späte und damit kostenintensive Änderungen zu reduzieren oder zu vermeiden.
4. **Grundsatz:**
   Häufung von Fehlerzuständen
   In der Regel ist keine Gleichverteilung der Fehlerzustände über das gesamte System gegeben. Vielmehr finden sich die meisten Fehlerzustände in nur wenigen Teilen (Modulen) eines Systems. Die geschätzte oder auch tatsächlich beobachtete Anhäufung von Fehler-

zuständen in einzelnen Systemteilen kann zur Risikoanalyse genutzt werden. Der Testaufwand kann dann auf die fehlerträchtigen Teile konzentriert werden (s. auch Grundsatz 2).

5. **Grundsatz:**
   Vorsicht vor dem Pestizid-Paradoxon
   So wie Schädlinge gegen Pflanzenschutzmittel resistent werden, lassen Tests in ihrer »Wirksamkeit« nach. Werden Tests an unveränderten Systemversionen nur wiederholt, decken sie keine neuen Fehlerwirkungen mehr auf. Damit die Effektivität der Tests nicht absinkt, sind die vorhandenen Testfälle regelmäßig zu prüfen und durch neue oder modifizierte Testfälle zu ergänzen. Bisher nicht geprüfte Teile der Software oder unberücksichtigte Konstellationen bei der Eingabe werden dann ausgeführt und somit mögliche weitere Fehlerwirkungen nachgewiesen. Das Pestizid-Paradoxon kann auch einen vermeintlich positiven Effekt hervorrufen. Wenn zum Beispiel der automatisierte Regressionstest eine geringe Anzahl von Fehlerwirkungen aufdeckt, kann dies nicht an der guten Qualität der Software, sondern an der »Unwirksamkeit« der (veralteten) Testfälle liegen.

6. **Grundsatz:**
   Testen ist kontextabhängig
   Je nach Einsatzgebiet und Umfeld des zu prüfenden Systems ist das Testen anzupassen. Keine zwei Systeme sind auf die exakt gleiche Art und Weise zu testen. Intensität des Testens, Definition der Endekriterien usw. sind bei jedem System entsprechend seines Einsatzumfelds festzulegen. Eingebettete Systeme (»Embedded Systems«) verlangen andere Prüfungen als etwa ein E-Commerce-System. Testen in agilen Projekten ist anders durchzuführen als das Testen in einem Projekt mit sequenziellem Lebenszyklus.

7. **Grundsatz:**
   Trugschluss: Keine Fehler bedeutet ein brauchbares System
   Trotz gründlicher Tests aller spezifizierten Anforderungen und Beheben aller gefundenen Fehlerzustände kann ein System entwickelt worden sein, das schwer zu nutzen ist, das die Bedürfnisse und Erwartungen der Nutzer nicht erfüllt oder das geringe Qualität aufweist im Vergleich zu ähnlichen anderen Systemen (oder dem Vorgängersystem). Frühzeitige Einbeziehung der späteren Nutzer in den Entwicklungsprozess und die Nutzung von Prototyping sind vorbeugende Maßnahmen zur Vermeidung des Problems.

Exkurs

2.2 Softwarequalität[8]

Das Testen von Software dient durch die Identifizierung von Fehlerwirkungen und deren anschließender Beseitigung zur Steigerung der Softwarequalität. Die Testfälle sollten so gewählt werden, dass sie weitgehend der späteren Benutzung der Software entsprechen. Die nachgewiesene Qualität des Programms während der Tests entspricht dann der zu erwartenden Qualität während der späteren Nutzung.

Softwarequalität umfasst aber mehr als nur die Beseitigung der beim Test aufgedeckten Fehlerwirkungen. Nach ISO 25010 [ISO 25010] wird Softwarequalität in zwei Modelle unterteilt:

ISO 25010: quality in use model & product quality model

- das Nutzungsqualitätsmodell (*quality in use model)* und
- das Produktqualitätsmodell (*product quality model)*.

Im Nutzungsqualitätsmodell werden folgende fünf Eigenschaften (*characteristics*) zusammengefasst:

- Effektivität (*effectiveness)*
- Effizienz (*efficiency)*
- Nutzungszufriedenheit (*satisfaction)*
- Risikofreiheit (*freedom from risk)*
- Kontextabdeckung (*context coverage)*

Das Produktqualitätsmodell umfasst acht Eigenschaften:

- Funktionelle Eignung (*functional sustainability)*
- Leistungseffizienz (*performance efficiency)*
- Kompatibilität (*compatibility)*
- Benutzungsfreundlichkeit (*usability)*
- Zuverlässigkeit (*reliability)*
- Sicherheit (*security)*
- Wartbarkeit (*maintainability)*
- Portabilität (*portability)*

Im Produktqualitätsmodell finden sich die meisten Überschneidungen zur Vorgängerversion, der ISO-Norm 9126. Detaillierte Informationen zum Datenqualitätsmodell (*data quality model)* sind der ISO-Norm 25012 [ISO 25012] zu entnehmen.

Alle diese Eigenschaften bzw. Qualitätsmerkmale sind beim Testen zu bedenken, um die Gesamtqualität eines Softwareprodukts umfassend beurteilen zu können. Welches Qualitätsniveau das Testobjekt dabei in jeder einzelnen Eigenschaftsdimension aufweisen soll, muss vorab als Qualitätsanforderung festgelegt werden. Die Erfüllung dieser Anforderungen ist dann je Kriterium durch geeignete Tests zu überprüfen.

8. Um den Begriff Softwarequalität für den Leser »greifbarer« zu machen, ist dieser Abschnitt als Exkurs aufgenommen worden.

Beispiel: Anwendung der ISO 25010 für VSR-II

Die verantwortliche Leiterin für Test/QS des VSR-II-Gesamtsystems schlägt dem Projektlenkungsausschuss vor, als Grundlage für die Bewertung und Verfolgung der Produktqualität von VSR-II das Produktqualitätsmodell aus der ISO 25010 zu verwenden. Dies findet breite Zustimmung und die Leiterin Test/QS wird beauftragt, bis zur nächsten Sitzung einen Vorschlag zu erarbeiten, wie ISO 25010 im VSR-II-Projekt angewendet wird. Als Kern ihres Vorschlags erarbeitet sie eine Matrix, die darstellt, welche Qualitätseigenschaft für welche Produktkomponente welche Relevanz hat und welche Interpretation gelten soll. Die erste Fassung dieser Matrix sieht folgendermaßen aus:

Tab. 2–1 *Festlegung der Qualitätseigenschaften*

Qualitäts-eigenschaften	*Dream-Car*	*Easy-Finance*	*FactBook*	*Just-InTime*	*NoRisk*	*Connected-Car*
Funktionelle Eignung	high	high	high	high	high	high
Leistungs-effizienz	high	mid	high	mid	mid	high
Kompatibilität	VSR	VSR	VSR	VSR	VSR	–
Benutzungs-freundlichkeit	high	mid	–	mid	mid	high
Zuverlässigkeit	mid	mid	high	high	mid	high
Sicherheit	mid	mid	high	high	mid	high
Wartbarkeit	high	mid	mid	mid	mid	high
Portabilität	low	low	–	low	low	mid

Die Einstufungen sind relativ zueinander zu verstehen. Für jede Matrix-Zelle begründet sie, warum die jeweilige Einstufung erfolgt, z. B.:

- **Funktionelle Eignung/alle**
  Jedes der Systeme bedient sehr hohe Anwenderzahlen bzw. verarbeitet deren Daten, funktionelle Fehler haben daher hohes wirtschaftliches Schadenspotenzial. Die Anforderungen an die Funktionelle Eignung sind daher durchgehend mit »high« bewertet.
- **Kompatibilität/*ConnectedCar***
  Keine Anforderung, da das System komplett neu entsteht.
- **Benutzungsfreundlichkeit/*FactBook***
  Keine Anforderung, da das System ein Backend-System ist und die API bereits feststeht.
- **Portabilität/*DreamCar***
  Einstufung »low«, da das Framework, das verwendet wird, dies ohne besondere Maßnahmen weitgehend abdeckt.

Welche Prüfungen und Tests jeweils vorzusehen sind, muss im Zuge der QS/Test-Planung je Teilsystem festgelegt werden. Auf dieser obersten Ebene können jedoch Rahmenbedingungen vorgegeben werden, wie beispielsweise: Zu einem »high«-Kriterium müssen automatisierte Tests für »Continuous In-

tegration« (s. Abschnitt 3.2) vorliegen; bei »mid« reichen Abnahmetests einmalig pro Iteration aus; bei »low« genügt eine schriftliche Designvorgabe, wie das Thema durch das Entwicklungsteam »adressiert« wird. Hier wird die Leiterin Test/QS sicher noch einige Abstimmungsrunden mit den Teams drehen müssen.

Insgesamt 40 Teilmerkmale

Die oben aufgeführten 13 Qualitätsmerkmale für Softwarequalität werden in der ISO-Norm 25010 in weitere Teilmerkmale verfeinert. Bei den beiden Modellen (*quality in use model* und *product quality model*) sind es insgesamt 40 unterschiedliche Teilmerkmale bzw. Eigenschaften. Es ist nicht möglich, alle 40 hier detailliert zu beschreiben. Der interessierte Leser sei auf die ISO-Norm 25010 [ISO 25010] verwiesen. Einige der Qualitätsmerkmale werden im Folgenden jedoch beispielhaft beschrieben:

Funktionelle Eignung – Funktionalität

Die Funktionelle Eignung (*functional sustainability*) oder Funktionalität des Produktqualitätsmodells umfasst alle Charakteristiken, die die geforderten Fähigkeiten des Produkts oder Systems beschreiben.

Das Qualitätsmerkmal untergliedert sich in drei Teilmerkmale:

- **Funktionale Vollständigkeit** (*functional completeness*)
  Deckt der Satz von Funktionen alle spezifizierten Aufgaben und Benutzerziele ab?
- **Funktionale Korrektheit** (*functional correctness*)
  Werden in dem Produkt oder System die richtigen Ergebnisse mit der erforderlichen Genauigkeit geliefert?
- **Funktionale Angemessenheit** (*functional appropriateness*)
  Inwieweit werden durch die Funktionen bestimmte Aufgaben und Ziele erreicht?

Ob unter bestimmten Bedingungen die angegebenen und implizierten Anforderungen in Bezug auf die Funktionalität vom System eingehalten werden, kann durch entsprechende Tests nachgewiesen und die gestellten Fragen somit beantwortet werden.

Die Funktionalität wird meist durch ein spezifiziertes Ein-/Ausgabeverhalten und/oder eine entsprechende Reaktion oder Wirkung des Systems auf entsprechende Eingaben beschrieben. Aufgabe des Tests ist es, nachzuweisen, dass jede einzelne geforderte Fähigkeit im System so realisiert wurde, dass das spezifizierte Ein-/Ausgabeverhalten oder die spezifizierte Reaktion erfüllt wird.

Zuverlässigkeit

Die Zuverlässigkeit (*reliability*) des Produktqualitätsmodells beschreibt die Fähigkeit eines Systems, ein Leistungsniveau unter festgelegten Bedingungen über einen definierten Zeitraum zu bewahren.

Das Qualitätsmerkmal untergliedert sich in vier Teilmerkmale:

- **Reife** (*maturity*)
  Wie gut erfüllt ein System, ein Produkt oder eine Komponente die Anforderungen an die Zuverlässigkeit im Normalbetrieb?
- **Verfügbarkeit** (*availability*)
  Wie hoch ist die Betriebsbereitschaft des Systems, des Produkts oder einer Komponente und wie leicht zugänglich ist deren Verwendung bei entsprechendem Bedarf?

- **Fehlertoleranz** (*fault tolerance*)
  Wie gut funktioniert ein System, ein Produkt oder eine Komponente trotz Vorhandensein von Hard- oder Softwarefehlern?
- **Wiederherstellbarkeit** (*recoverability*)
  Wie lange dauert bei dem Produkt oder dem System nach einer Unterbrechung oder einem Ausfall die Wiederherstellung von direkt betroffenen Daten und die Wiederherstellung des gewünschten Zustands des Systems?

Zufriedenheit

Die Nutzungszufriedenheit (*satisfaction*) des Nutzungsqualitätsmodells (*quality in use model*) umfasst alle Charakteristiken, inwieweit die Bedürfnisse der Benutzer erfüllt werden, wenn ein Produkt oder System in einem bestimmten Anwendungskontext verwendet wird.

Das Qualitätsmerkmal untergliedert sich in vier Teilmerkmale:

- **Nützlichkeit** (*usefulness*)
  Wie zufrieden ist ein Nutzer mit der von ihm wahrgenommenen Erreichung pragmatischer Ziele, einschließlich der Ergebnisse der Nutzung und der Folgen der Nutzung?
- **Vertrauen** (*trust*)
  Wie hoch ist das Vertrauen, das ein Nutzer oder ein anderer Interessenvertreter hat, dass sich ein Produkt oder System wie beabsichtigt verhält?
- **Vergnügen** (*pleasure*)
  Wie viel »Freude« hat ein Nutzer an der Erfüllung seiner persönlichen Anforderungen durch das verwendete System?
- **Komfort** (*comfort*)
  Wie angenehm empfindet der Nutzer das System – auch im Hinblick auf körperliches Wohlbefinden?

Die meisten der Merkmale des Nutzungsqualitätsmodells unterliegen sehr stark einer persönlichen Einschätzung und lassen sich nur in Ausnahmen objektiv bzw. exakt messbar bewerten. Für den Nachweis dieser Qualitätsmerkmale empfiehlt es sich, auf mehrere Nutzer(gruppen) zurückzugreifen, um aussagekräftige (Test-)Ergebnisse zu erhalten.

Ein Softwaresystem kann nicht alle Qualitätsmerkmale gleich gut erfüllen. Teilweise kann die Erfüllung eines Merkmals auch die Nichterfüllung eines anderen bewirken. So wird ein hocheffizientes Softwaresystem nur sehr eingeschränkt übertragbar sein, weil die Entwickler zur Effizienzsteigerung meistens spezielle Eigenschaften der verwendeten Plattform ausnutzen, die wiederum die Portabilität negativ beeinflussen.

Qualitätsmerkmale priorisieren

Es muss deshalb festgelegt werden, welche Qualitätseigenschaften mit welcher Priorität umgesetzt werden sollen. Diese Festlegung dient dann auch für den Test als Richtschnur zur Bestimmung der Intensität der Überprüfung der einzelnen Qualitätsmerkmale.