

ISTQB® Certified Tester

Foundation Level
Handout für Schulungsteilnehmer



Unternehmensdarstellung der Sogeti Deutschland GmbH

Sogeti berät und unterstützt Kunden in allen Fragen der IT. Unsere Kunden sind Unternehmen aller Branchen, deren Erfolg mit anspruchsvollen Softwarelösungen steht und fällt. Sogeti ist der führende Anbieter von Technologie- und Engineering-Dienstleistungen und bietet wegbereitende Lösungen in der digitalen Transformation sowie maßgeschneiderte Expertise rund um Cloud, Cybersecurity, Digital Manufacturing, Digital Assurance und Testing sowie Trendtechnologien.

Als weltweit größter spezialisierter Dienstleistungsanbieter für Testlösungen unterstützt Sogeti Unternehmen, deren Test- und Qualitätssicherungsleistungen zu verbessern. Gemeinsam mit Capgemini verfügt Sogeti weltweit über einen der größten Pools von dedizierten Testexperten mit über 12.000 Testspezialisten der Sogeti und weiteren 14.500 Capgemini IT-Profis mit Testausbildung.

In Deutschland fokussiert Sogeti auf das Thema Software-Testen und unterstützt seine Kunden mit ausgebildeten und erfahrenen Beratern bei der Durchführung anspruchvoller Testprojekte. Sogeti Deutschland bietet ein breites Spektrum von hochwertigen, effektiven und kosteneffizienten Lösungen und der Qualitätssicherung. Es reicht von Quality Consultancy, DevOps, Testing Services, Digital Assurance über Testautomatisierung bis hin zum globalen Outsourcing in Testcentern.

Wir betreiben eine interne Basisausbildung für neue Mitarbeiter sowie kontinuierliche Weiterbildung. Jeder Testberater bekommt dabei sowohl das Basiswissen zum Softwaretest nach ISTQB® (Foundation Level) als auch entsprechende Kenntnisse unserer Methoden TMap® und TPI® vermittelt. Diese Sogeti-Methoden wurden auf Basis unserer umfangreichen Projekterfahrungen im Unternehmen selbst entwickelt, von Sogeti international veröffentlicht und werden sowohl von Praktikern als auch von Theoretikern geschätzt. TMap NEXT® und TPI NEXT® sind eingetragene Markenzeichen der Sogeti Nederland B.V. und werden laufend weiterentwickelt. ISTQB® ist eine registrierte Marke des International Software Testing Qualification Board A.I.S.B.L., Brüssel.

Alle internen sowie externen Schulungsaktivitäten werden durch erfahrene Testberater gehalten, die über Projekterfahrungen verfügen und präsentationstechnisch geschult sind. Die Kursentwicklung wird ebenso durch die weltweite Zusammenarbeit gestützt. Die Sogeti Deutschland GmbH ist in diese internationalen Aktivitäten aktiv eingebunden und für die Pflege der deutschen Version unserer Methoden verantwortlich. Konkrete Beispiele hierfür sind die Herausgabe von „TMap NEXT®“ (2008) und „TPI NEXT®“ (in 2011) auf Deutsch. Unsere Testberater sind international vernetzt und können so auf die Erfahrungen von internationalen Kollegen zurückgreifen, und teilen ihre Expertise ebenso mit der internationalen Gemeinschaft.

Sogeti verfügt in Deutschland (Stand: Juni 2018) über mehr als 400 Berater für Qualitätssicherung. Insgesamt sind 317 Berater zertifiziert für das „ISTQB Certified Tester Foundation Level“, 106 Mitarbeiter sind zertifiziert für das „ISTQB Certified Tester Advanced Level“.

Sogeti Deutschland GmbH
Balcke-Dürr-Allee 7
40882 Ratingen
Telefon +49 2102 101 4000
Telefax +49 2102 101 4100
E-Mail kontakt@sogeti.de
www.sogeti.de

Inhaltsverzeichnis

Einleitung	4
1. Grundlagen des Testens	5
1.1 Was ist Testen?	6
1.2 Warum ist Testen notwendig?.....	8
1.3 Sieben Grundsätze des Testens	10
1.4 Testprozess.....	12
1.5 Die Psychologie des Testens	16
2. Testen im Softwareentwicklungslebenszyklus	17
2.1 Softwareentwicklungslebenszyklus-Modelle	18
2.2 Teststufen.....	21
2.3 Testarten	24
2.4 Wartungstest.....	27
3. Statischer Test	29
3.1 Grundlagen des statischen Tests	30
3.2 Reviewprozess.....	32
4. Testverfahren.....	39
4.1 Kategorien von Testverfahren.....	40
4.2 Black-Box-Testverfahren.....	42
4.3 White-Box-Testverfahren	51
4.4 Erfahrungsbasierte Testverfahren	53
5. Testmanagement.....	54
5.1 Testorganisation	55
5.2 Testplanung und -schätzung.....	57
5.3 Testüberwachung und -steuerung	63
5.4 Konfigurationsmanagement.....	65
5.5 Risiken und Testen	66
5.6 Fehlermanagement	70
6. Werkzeugunterstützung für das Testen	72
6.1 Überlegungen zu Testwerkzeugen	73
6.2 Effektive Nutzung von Werkzeugen	77
Platz für Notizen.....	79

Einleitung

Hinweise zu diesem Dokument

Dieses Teilnehmer Handout dient zur Erläuterung der während der Schulung erstellten Flipchart-Bilder und zur Nachbereitung des diskutierten Stoffes. Die Gliederung dieses Dokuments folgt der des Lehrplans. Einige Kapitel können in der Schulung jedoch auch abweichend von der Lehrplanabfolge an anderer Stelle behandelt werden. Dies geschieht, falls dies aus individueller Sicht des Dozenten oder bezogen auf die Teilnehmer didaktisch sinnvoller ist.

Entlang der Gliederung werden die während der Schulung erstellten Flipchart-Bilder erläutert. Die Bilder in diesem Dokument sind als minimale Vorlage zu verstehen, auf deren Basis die Inhalte zusammen mit den Teilnehmern erarbeitet werden. Je nach Vorkenntnissen, Mitarbeit, Rückfragen oder inhaltlichem Verständnis der Schulungsteilnehmer können die tatsächlich gezeichneten Flipcharts durch den Dozenten mit weiteren Details ergänzt werden. Damit können einzelne Aspekte ggf. besonders verdeutlichend, vertiefend oder auch abkürzend skizziert werden.

Einige Bilder enthalten zudem freie Bereiche, die von den Teilnehmern selbst durch entsprechende Notizen ergänzt werden können. Diese betreffenden Themen oder Punkte werden im Rahmen der Schulung interaktiv gemeinsam am Flipchart oder an einer Moderationswand erarbeitet. Zusätzliche können die Teilnehmer selbstverständlich weitere Notizen in den Bildern ergänzen, z.B. passenden Text zu den verwendeten Icons.

Ergänzend zu den Flipchart-Bildern gliedern sich die Begleittexte in Informationen zum Bildaufbau, sowie Diskussionsvorschläge und Fragestellungen, die im Zusammenhang mit den Bildern durch die Teilnehmer erörtert werden können. Weiterhin enthalten die Begleittexte auch weiterführende Hinweise zu Beispielen, Übungsaufgaben und Schlüsselbegriffen. Es werden auch die Lernziele, die das jeweilige Thema abdeckt, referenziert.

Die Texte geben bewusst nicht den Lehrplan wieder, sondern beschreiben die Themen zusammengefasst. Insofern ersetzt dieses Handout nicht das Lesen des Lehrplans.

An einigen Stellen enthält der Lehrplan wichtige Definitionen oder Auflistungen, die hier zum Teil unverändert zitiert werden. Die entsprechenden Stellen sind durch *kursiv gesetzten Text* und die Angabe der jeweils exakten Quelle im Lehrplan gekennzeichnet.

Anmerkung:

Einige Unterkapitel des Lehrplans werden in der Schulung thematisch auf einem Flipchart-Bild zusammengefasst. Da dieses Handout zur Erläuterung der Flipchart-Bilder dient, und bewusst nicht den Lehrplan wiedergeben soll, kann die Nummerierung der Kapitel zum Teil von der des Lehrplans abweichen. Eine Verbindung zum Lehrplan ist dennoch über die in jedem Abschnitt angegebenen Lernziele leicht möglich.

1. Grundlagen des Testens

Flipchart-Bilder

- Testziele
- Fehlerwirkungskette
- 7 Grundsätze des Softwaretestens
- Testprozess

Schlüsselbegriffe im Kapitel

Debugging, Fehlerwirkung, Fehlerzustand, Fehlhandlung, Grundursache, Qualität, Qualitätssicherung, Verfolgbarkeit, Testablauf, Testabschluss, Testanalyse, Testausführungsplan, Testbasis, Testbedingung, Testdurchführung, Testdaten, Testen, Testentwurf, Testfall, Testmittel, Testobjekt, Testorakel, Testplanung, Testprozess, Testrealisierung, Teststeuerung, Testsuite, Testüberwachung, Testziel, Überdeckung, Validierung, Verifizierung

1.1 Was ist Testen?

-
- FL-1.1.1 (K1) Typische Ziele des Testens identifizieren können
 - FL-1.2.1 (K2) Beispiele dafür geben können, warum Testen notwendig ist
 - FL-1.2.2 (K2) Die Beziehung zwischen Testen und Qualitätssicherung beschreiben können und Beispiele dafür geben können, wie Testen zu höherer Qualität beiträgt
-



Beispiele	Übungen	Schlüsselbegriffe
1.1.2: Kalenderdialog 1.2.1: Aktuelle News 1.2.2: 3D-Drucker		Grundursache, Qualität, Qualitätssicherung, Testen, Validierung, Verifizierung

Erläuterungen

Softwaretesten ist ein eigenständiger Prozess, der neben dem Entwicklungsprozess, dem Projektmanagement, der Anforderungsanalyse und anderen Prozessen gleichberechtigt in IT-Projekte integriert ist. Der zertifizierte Softwaretester kennt diesen Prozess und kann ihn in IT-Projekten geeignet anwenden, sowie seine Rolle innerhalb der Qualitätssicherung gegenüber anderen Projektbeteiligten angemessen vertreten.

Um die Notwendigkeit von Softwaretests zu motivieren, genügt ein Blick in die einschlägigen tagesaktuellen IT-News, in denen täglich von neuen Datenschutz-Verstößen, Sicherheitslücken oder dringend empfohlenen Systemupdates berichtet wird. Heutzutage zweifelt niemand mehr an, dass das Testen von Software vor der Inbetriebnahme oder Vermarktung eines Produkts absolut notwendig ist. Umso wichtiger ist es, Softwaretests strukturiert und verlässlich durchzuführen, eine einheitliche, standardisierte Terminologie zu verwenden, und den Zweck und die Ziele des Softwaretestens zu formulieren.

Testen ist als Teil der **Qualitätssicherung** eine der Maßnahmen zur Steuerung der Produktqualität einer zu entwickelnden Software. Qualitätssicherung geht jedoch weit über das reine Softwaretesten hinaus und behandelt z.B. auch die **Grundursachenanalysen** oder Prozessverbesserungen in IT-Projekten sowie die Produktentwicklung außerhalb der IT.

Für den Tester ist es wichtig, seine Kompetenzen nicht nur für die Durchführung von Softwaretests an einem fertigen Softwareprodukt einzusetzen, sondern darüber hinaus auch qualitativ Einfluss auf andere Disziplinen zu nehmen, etwa durch folgende Beiträge [Lehrplan, Kapitel 1.2.1]:

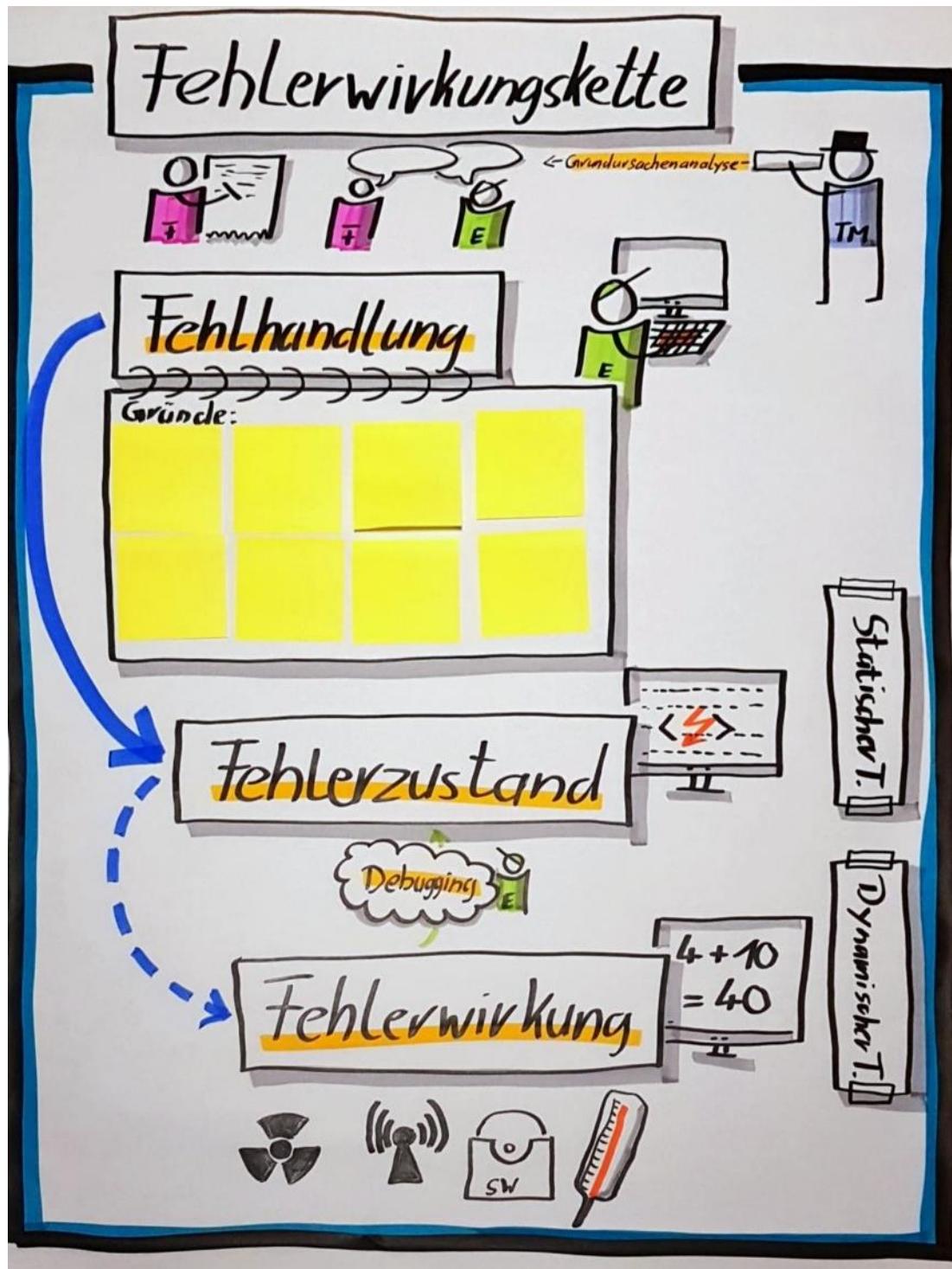
- *Teilnahme an Anforderungsreviews und Aufdecken von Fehlerzuständen*
- *Enge Zusammenarbeit mit Systementwicklern während des Systementwurfs und Einwirken auf die Möglichkeit, das System zu testen*
- *Enge Zusammenarbeit mit Entwicklern während der Erstellung des Codes zur Reduzierung von Fehlerzuständen*
- **Verifizierung und Validierung** der Software vor der Freigabe zum Aufdecken von Fehlerwirkungen und zur Erhöhung der Wahrscheinlichkeit, dass die Software den Bedürfnissen der Stakeholder entspricht

Die beschriebenen Aspekte werden in den Zielen des Testens recht anschaulich zusammengefasst. Eine besondere Bedeutung hat dabei die **Testbarkeit** der beschriebenen Anforderungen oder User-Storys. Tester können einen wertvollen Beitrag liefern, um Fehlerzustände aufzudecken, und damit das Risiko der Entwicklung fehlerhafter oder schlecht testbarer Funktionalitäten reduzieren.

- Die Testziele aus dem Lehrplan, Abschnitt 1.1.1, werden in der Schulung besprochen und können hier nachgetragen werden.

1.2 Warum ist Testen notwendig?

- FL-1.1.2 (K2) Testen von Debugging unterscheiden können
- FL-1.2.3 (K2) Zwischen Fehlhandlung, Fehlerzustand und Fehlerwirkung unterscheiden können
- FL-1.2.4 (K2) Zwischen der Grundursache eines Fehlerzustands und seinen Auswirkungen unterscheiden können



Beispiele	Übungen	Schlüsselbegriffe
1.1.2: Kalenderdialog 1.2.3 und 1.2.4: Fehlerwirkungskette		Debugging, Fehlerwirkung, Fehlerzustand, Fehlhandlung

Erläuterungen

Wir unterteilen den recht allgemeinen Begriff des Fehlers hier in drei voneinander abzugrenzende Fehlerbegriffe, um zu verdeutlichen, wie Fehler entstehen, wie sie aufgedeckt werden, und wie man der Entstehung vorbeugen kann.

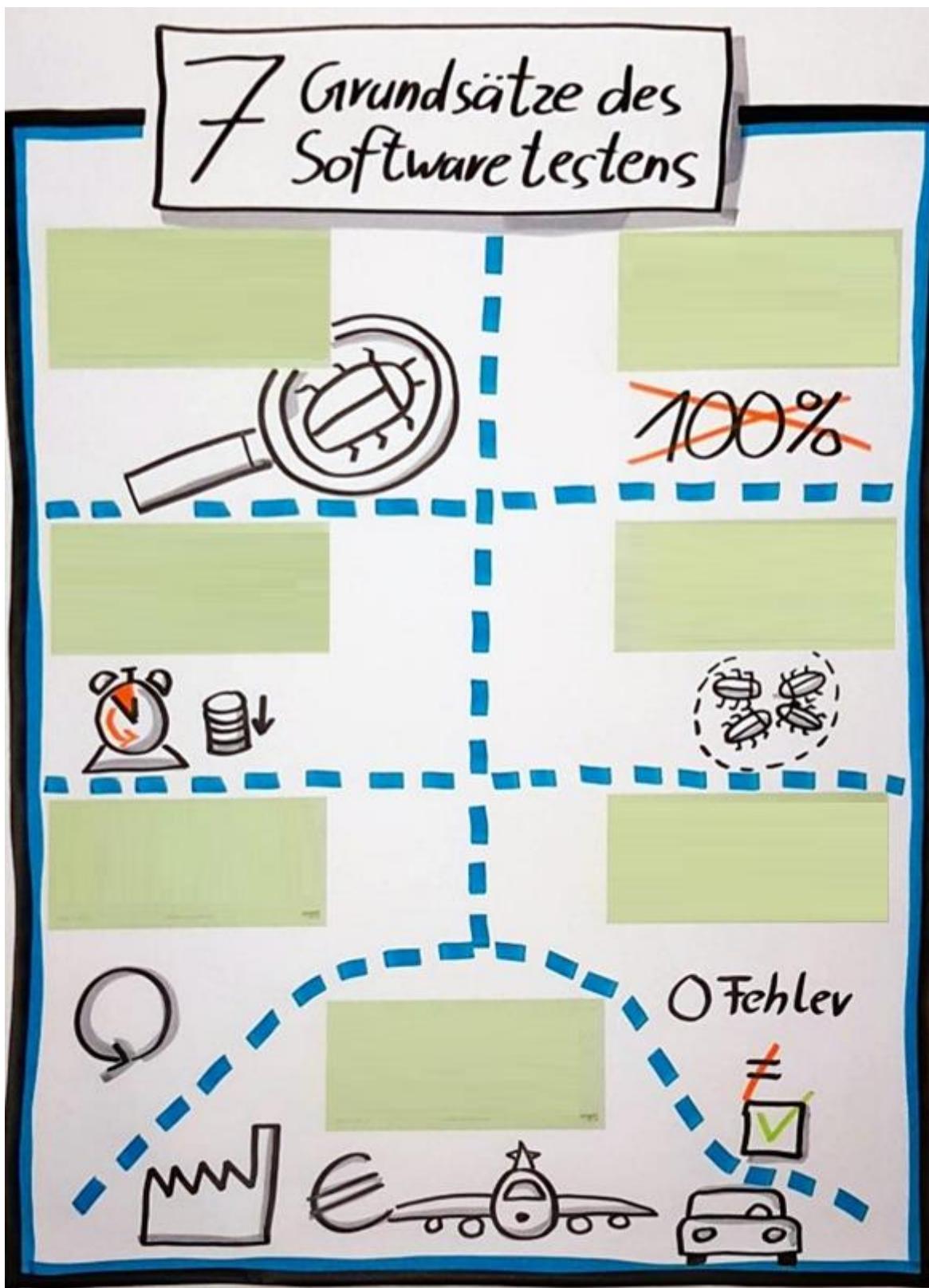
Eine **Fehlhandlung** führt zu einem **Fehlerzustand**, der eine **Fehlerwirkung** hervorrufen kann. Oder anders herum ausgedrückt: Die Ursache für eine Fehlerwirkung kann ein Fehlerzustand sein, der wiederum durch eine Fehlhandlung verursacht wurde. Hierbei ist es wichtig zu erkennen, dass ein Fehlerzustand zwar immer eine Fehlhandlung voraussetzt, eine Fehlerwirkung jedoch nicht unbedingt auftreten muss. Außerdem ist eine Fehlerwirkung, wenn sie auftritt, auch nicht in jedem Fall auf einen Fehlerzustand zurückzuführen, sondern kann auch durch Nebeneffekte hervorgerufen werden.

→ Gründe für Fehlhandlungen gemäß Lehrplan, Abschnitt 1.2.3, können hier ergänzt werden.

Debugging ist am anschaulichsten an Hand der Fehlerwirkungskette zu veranschaulichen. Der Entwickler nutzt Debugging, um Fehlerzustände zu lokalisieren. Dies kann einerseits während der Entwicklung geschehen, um Fehlerzustände aufzudecken, bevor Software in den dynamischen Test gegeben wird, und somit Fehlerwirkungen vorbeugen. Andererseits können durch den dynamischen Test aufgedeckte Fehlerwirkungen durch Debugging auf Fehlerzustände zurückgeführt werden.

1.3 Sieben Grundsätze des Testens

FL-1.3.1 (K2) Die sieben Grundsätze des Softwaretestens erklären können



Beispiele	Übungen	Schlüsselbegriffe
1.3.1: Sieben Grundsätze		

Erläuterungen

Die Grundsätze werden im Lehrplan, Abschnitt 1.3, im Detail erläutert. Sie bieten einen guten Überblick über den Sinn und Zweck des Testens, aber warnen auch vor typischen Fehlern, die in IT-Projekten häufig gemacht werden, wenn das Testen missverstanden oder Ergebnisse falsch gedeutet werden.

Viele speziellere Aspekte des Softwaretestens lassen sich auf die 7 Grundsätze zurückführen, so dass man immer wieder hierauf zurück verweisen kann.

- ➔ Die 7 Grundsätze des Softwaretestens gemäß Lehrplan, Abschnitt 1.3, können hier ergänzt werden.

1.4 Testprozess

-
- FL-1.4.1 (K2) Die Auswirkungen des Kontexts auf den Testprozess erklären können
- FL-1.4.2 (K2) Die Testaktivitäten und zugehörigen Aufgaben innerhalb des Testprozesses beschreiben können
- FL-1.4.3 (K2) Arbeitsergebnisse unterscheiden können, die den Testprozess unterstützen
- FL-1.4.4 (K2) Die Bedeutung der Pflege der Verfolgbarkeit zwischen Testbasis und Testarbeitsergebnissen erklären können
-

Beispiele	Übungen	Schlüsselbegriffe
1.4.1: Auswirkungen des Kontexts 1.4.2: Testaktivitäten 1.4.3: Arbeitsergebnisse 1.4.4: Verfolgbarkeit		Verfolgbarkeit, Testablauf, Testabschluss, Testanalyse, Testausführungsplan, Testbasis, Testbedingung, Testdurchführung, Testdaten, Testentwurf, Testfall, Testmittel, Testobjekt, Testorakel, Testplanung, Testrealisierung, Teststeuerung, Testsuite, Testüberwachung, Testziel, Überdeckung

Erläuterungen

Der Testprozess beschreibt die im Rahmen eines Softwareentwicklungsprojekts notwendigen Testaktivitäten, die Aufgaben der beteiligten Rollen, sowie die Arbeitsergebnisse, die während der Testaktivitäten entstehen.

Wie die genannten Aspekte des Testprozesses in einem konkreten Projekt an die Anforderungen einer Organisation anzupassen sind, bestimmen z.B. das Softwareentwicklungslebenszyklus-Modell, der Geschäftsbereich, die Festlegung von Teststufen und Testarten, die Behandlung von Projekt- und Produktrisiken, Richtlinien, geforderte Standards sowie betriebliche Beschränkungen [Lehrplan Abschnitt 1.4.1].

Ein besonderes Merkmal aller Testprozesse ist die herzustellende **Verfolgbarkeit** zwischen **Testbasis** und Testarbeitsergebnissen. Diese erleichtert die Bewertung der **Testüberdeckung**, und unterstützt außerdem [Lehrplan, Abschnitt 1.4.4]:

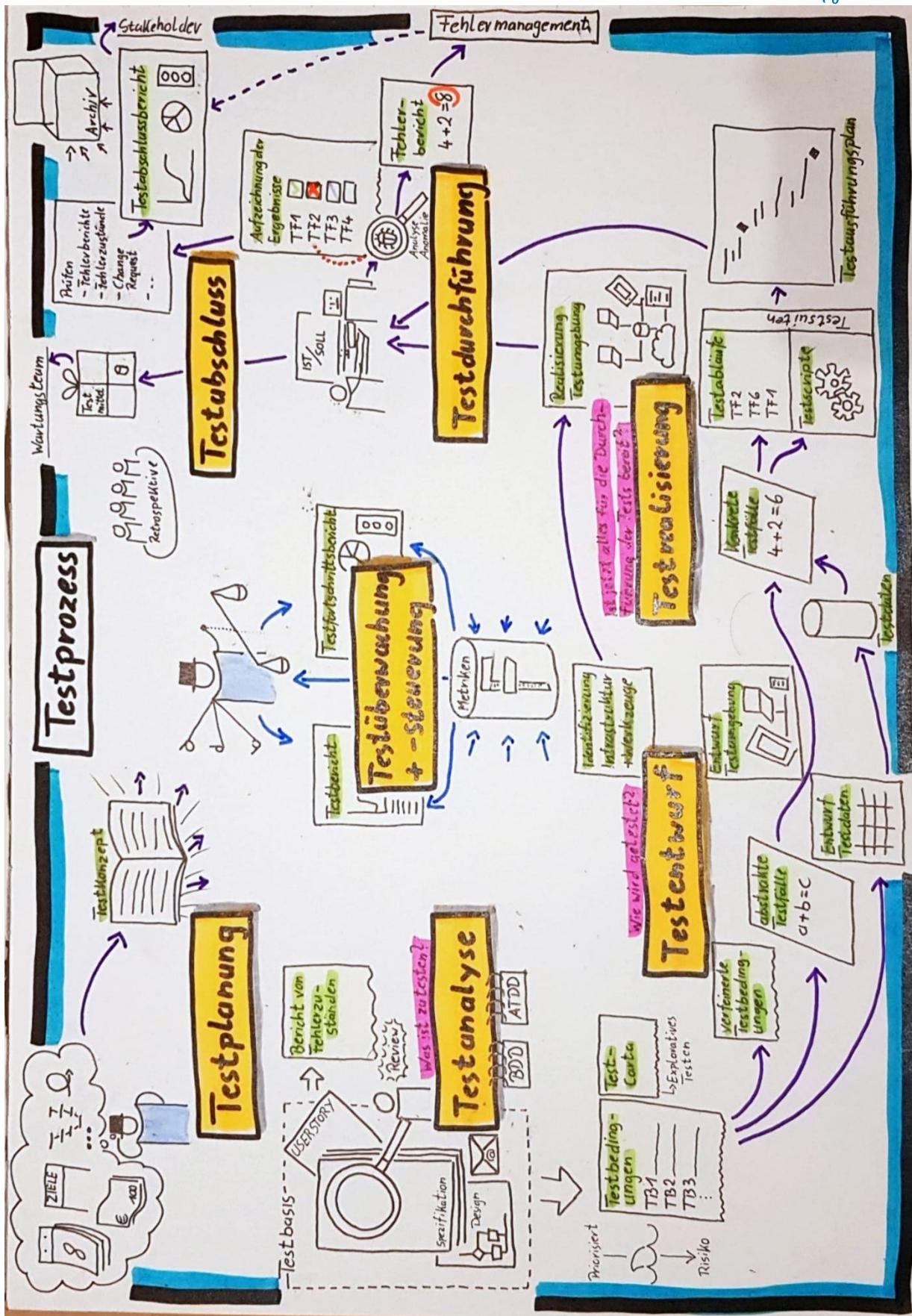
- *Die Auswirkungsanalyse von Änderungen*
- *Testen nachvollziehbarer zu machen*
- *IT-Governance-Kriterien zu erfüllen*
- *Die Verständlichkeit von Testfortschrittsberichten und Testabschlussberichten zu verbessern, um den Status der Elemente der Testbasis einzubeziehen*
- *Bericht über die technischen Aspekte des Testens an die Stakeholder in einer Art und Weise, die sie verstehen können*

- Weitergabe von Informationen zur Beurteilung von Produktqualität, Prozessfähigkeit und Projektfortschritt gegenüber Geschäftszielen

Die im Lehrplan, Abschnitte 1.4.2 und 1.4.3 im Detail beschriebenen Aktivitäten und Arbeitsergebnisse werden hier tabellarisch zusammengefasst, sowie auf dem zugehörigen Flipchart auch grafisch veranschaulicht.

Hauptgruppe von Aktivitäten	Hauptaktivitäten und Einzelaufgaben	Arbeitsergebnisse
Testplanung	<ul style="list-style-type: none"> • Testziele definieren • Teststrategie bestimmen • Testkonzept erstellen 	<ul style="list-style-type: none"> • Testkonzept(e) • Endekriterien
Testüberwachung und -steuerung	<ul style="list-style-type: none"> • Vergleich des Fortschritts mit dem Testplan unter Nutzung von Überwachungsmetriken • Ergreifen notwendiger Maßnahmen • Bewertung von Endekriterien (Definition-of-Done) 	<ul style="list-style-type: none"> • Testfortschrittsberichte • Testabschlussberichte
Testanalyse <i>Was ist zu testen?</i>	<ul style="list-style-type: none"> • Analyse der Testbasis für die in Betracht gezogene Teststufe • Bewertung der Testbasis und der Testelemente, insbesondere im Hinblick auf Testbarkeit • Identifikation von Features und Feature-Sets • Definition und Priorisierung von Testbedingungen für jedes Feature • Erfassung bidirektionaler Verfolgbarkeit zwischen jedem Element der Testbasis und den zugehörigen Testbedingungen 	<ul style="list-style-type: none"> • Priorisierte Testbedingungen inkl. Verfolgbarkeit • Test Chartas • Fehlerzustände in der Testbasis
Testentwurf <i>Wie wird getestet?</i>	<ul style="list-style-type: none"> • Entwurf und Priorisierung von Testfällen und Sets an Testfällen • Identifizierung von notwendigen Testdaten zur Unterstützung der Testbedingungen und Testfälle • Entwurf der Testumgebung und Identifizierung benötigter Infrastruktur und Werkzeuge • Erfassung der bidirektionalen Verfolgbarkeit zwischen der Testbasis, den Testbedingungen, den Testfällen und den Testabläufen 	<ul style="list-style-type: none"> • (abstrakte) Testfälle und Sets von Testfällen, um Testbedingungen abzudecken • Entwurf/Identifizierung der notwendigen Testdaten • Weiter spezifizierte Testbedingungen

Hauptgruppe von Aktivitäten	Hauptaktivitäten und Einzelaufgaben	Arbeitsergebnisse
Testrealisierung <i>Ist alles bereit?</i>	<ul style="list-style-type: none"> Entwicklung und Priorisierung von Testabläufen Erstellung automatisierter Testskripte Erstellen von Testsuiten Anordnen der Testsuiten innerhalb eines Testausführungsplans Aufbau der Testumgebung und Verifizierung Vorbereitung von Testdaten Verifizierung und Aktualisierung der bidirektionalen Verfolgbarkeit 	<ul style="list-style-type: none"> Testabläufe und die Aneinanderreihung dieser Testabläufe Testsuiten Testausführungsplan Testdaten Weiter verfeinerte Testbedingungen
Testdurchführung	<ul style="list-style-type: none"> Aufzeichnung der IDs und Versionen des Testelements oder des Testobjekts, des Testwerkzeugs und Testmittel Durchführung der Tests Vergleich der Istergebnisse mir den erwarteten Ergebnissen Analyse der Anomalien zur Feststellung ihrer wahrscheinlichen Ursachen Bericht über Fehlerzustände auf Grundlage der beobachteten Fehlerwirkungen Aufzeichnung der Ergebnisse der Testdurchführung Wiederholung der Testaktivitäten Verifizierung und Aktualisierung der bidirektionalen Verfolgbarkeit zwischen der Testbasis, den Testbedingungen, den Testfällen, den Testabläufen und den Testergebnissen 	<ul style="list-style-type: none"> Dokumentation des Status individueller Testfälle und Testabläufe Fehlerbericht Dokumentation darüber, welche Testelemente, Testwerkzeuge und Testmittel für die Tests benutzt wurden
Testabschluss	<ul style="list-style-type: none"> Prüfen, ob alle Fehlerberichte geschlossen sind, eintragen von Change Requests oder Product-Backlog-Elementen für Fehlerzustände, die am Ende der Testdurchführung weiterhin nicht geschlossen sind Erstellen eines Testabschlussberichts Finalisieren und Archivieren der Testumgebung, der Testdaten, der Testinfrastruktur und anderer Testmittel Übergabe der Testmittel an die Wartungsteams Analyse der gewonnenen Erkenntnisse aus den abgeschlossenen Testaktivitäten Nutzung der gesammelten Informationen zur Verbesserung der Testprozessreife 	<ul style="list-style-type: none"> Testabschlussberichte Offene Punkte zur Verbesserung in nachfolgenden Projekten Change Requests Product-Backlog-Elemente Finalisierte Testmittel



1.5 Die Psychologie des Testens

FL-1.5.1 (K1) Die psychologischen Faktoren identifizieren können, die den Erfolg des Testens beeinflussen

FL-1.5.2 (K2) Den Unterschied zwischen der für Testaktivitäten erforderlichen Denkweise und der für Entwicklungsaktivitäten erforderlichen Denkweise erklären können

Kein Flipchart

Beispiele	Übungen	Schlüsselbegriffe
1.5.2: Tester und Entwickler		

Erläuterungen

Der Beruf des Softwaretesters erfordert es, im Hinblick auf die Qualitätssicherung eines Softwareprodukts, dass der Tester seine Kompetenzen auch in andere Disziplinen einbringt. Durch die Aufdeckung von Fehlerzuständen bewertet der Tester die Arbeit anderer Projektbeteiligter. Die Sichtweise eines Testers unterscheidet sich von der eines Entwicklers oder Architekten dahingehend, dass letztere davon ausgehen korrekte Ergebnisse zu liefern, der Tester jedoch davon ausgehen muss, Fehler zu finden. Der Lehrplan, Abschnitt 1.5.1 erläutert in diesem Zusammenhang den Begriff des „Bestätigungsfehlers“ aus der Humanpsychologie.

Gute soziale Kompetenzen in diesem Spannungsfeld sind für den Softwaretester essentiell. Die Kommunikation von Befunden, die aus Sicht anderer Projektbeteiligter zunächst negativ wahrgenommen werden, muss als positiv für die Produktqualität verkauft werden. Der Lehrplan, Abschnitt 1.5.1, nennt die folgenden Beispiele für gute Kommunikation:

- *Beginne mit Zusammenarbeit statt mit Streit. Erinnere jeden Beteiligten an das gemeinsame Ziel eines Systems von höherer Qualität.*
- *Betone den Nutzen des Testens. Zum Beispiel können den Autoren Informationen über Fehlerzustände helfen, ihre Arbeitsergebnisse und ihre Fähigkeiten zu verbessern. Während des Testens gefundene und behobene Fehlerzustände ersparen dem Unternehmen Zeit und Geld und reduzieren das allgemeine Risiko für die Produktqualität.*
- *Kommuniziere Testergebnisse und andere Erkenntnisse in einer neutralen, faktenorientierten Weise, ohne denjenigen zu kritisieren, der das defekte Element erstellt hat. Schreibe objektive und tatsächenbasierte Fehlerberichte und Reviewbefunde.*
- *Versuche zu verstehen, wie die andere Person sich fühlt, und die Gründe dafür nachzuvozziehen, warum sie negativ auf die Information reagieren könnte.*
- *Lasse dir bestätigen, was die andere Person verstanden hat, was gesagt wurde und umgekehrt.*

Insbesondere das Spannungsfeld zwischen Testern und Entwicklern bedarf hier detaillierteren Beispielen, da diese beiden Rollen in Projekten am häufigsten und intensiveren zusammen arbeiten sollten, grundsätzlich jedoch vollkommen unterschiedliche Sicht- und Denkweisen haben.

2. Testen im Softwareentwicklungslebenszyklus

Flipchart-Bilder

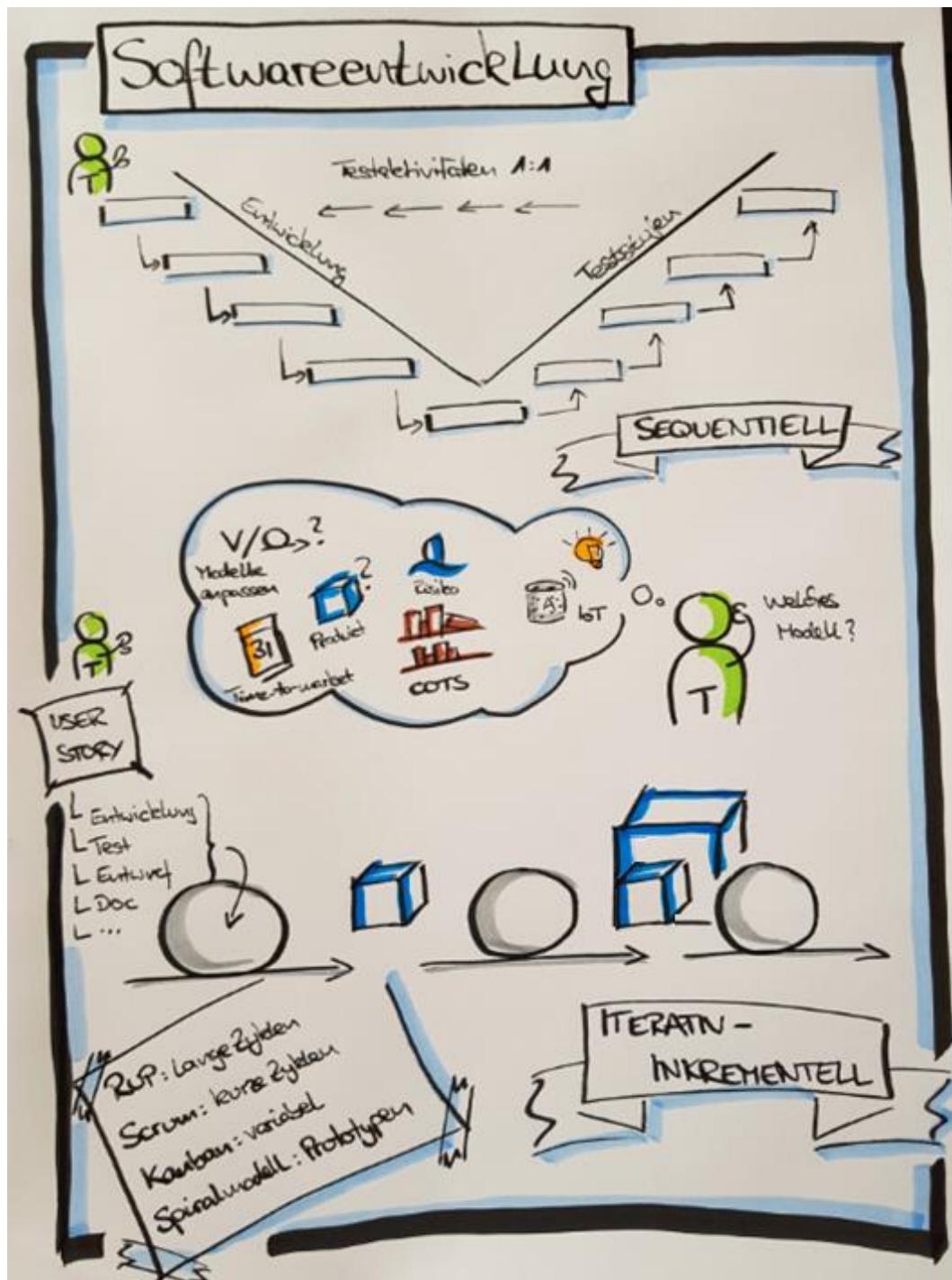
- Softwareentwicklung
- Teststufen
- Testarten
- Wartungstest

Schlüsselbegriffe im Kapitel

Abnahmetest, änderungsbezogenes Testen, Alpha-Test, Auswirkungsanalyse, Benutzerabnahmetest, Beta-Test, betrieblicher Abnahmetest, Fehlernachtest, funktionaler Test, Integrationstest, kommerzielle Standardsoftware, Komponentenintegrationstest, Komponententest, nicht-funktionaler Test, Regressionstest, regulatorischer Abnahmetest, sequenzielles Entwicklungsmodell, Systemintegrationstest, Systemtest, Teststart, Testbasis, Testfall, Testobjekt, Teststufe, Testumgebung, Testziel, vertraglicher Abnahmetest, Wartungstest, White-Box-Test

2.1 Softwareentwicklungslebenszyklus-Modelle

- FL-2.1.1 (K2) Die Beziehungen zwischen Softwareentwicklungsaktivitäten und Testaktivitäten im Softwareentwicklungslebenszyklus erklären können
- FL-2.1.2 (K1) Gründe identifizieren können, warum Softwareentwicklungslebenszyklus-Modelle an den Kontext des Projekts und an die Produktmerkmale angepasst werden müssen



Beispiele	Übungen	Schlüsselbegriffe
2.1.1: Komposit-V Standardsoftware		kommerzielle Standardsoftware, sequenzielles Entwicklungsmodell

Erläuterungen

Die Softwareentwicklung im Rahmen eines IT-Projekts folgt einem bestimmten Softwareentwicklungslebenszyklusmodell, d.h. die Entstehung eines Softwareprodukts durchläuft bestimmte vordefinierte Phasen. Der Lehrplan unterscheidet zwischen **sequenziellen** und **iterativ-inkrementellen** Entwicklungsmodellen, die jedoch in Bezug auf das Softwaretesten gemeinsame Merkmale aufweisen [Lehrplan, Abschnitt 2.1.1]:

- Für jede Entwicklungsaktivität gibt es eine zugehörige Testaktivität.
- Jede Teststufe hat ihre stufenspezifischen Ziele für den Test.
- Für eine vorgegebene Teststufe beginnen Testanalyse und Testentwurf bereits während der zugehörigen Entwicklungsaktivität.
- Tester nehmen an Diskussionen zur Definition und Verfeinerung von Anforderungen und des Entwurfs teil. Darüber hinaus sind sie am Review von Arbeitsergebnissen (z.B. Anforderungen, Architekturdesign, User-Stories usw.) beteiligt, sobald erste Entwürfe dafür vorliegen.

Ergänzend zur Beschreibung der Modelle im Lehrplan, Abschnitt 2.1.1, listen wir hier kurz die typischen Merkmale der einzelnen Modelle auf.

Modell	Typ	Merkmale
Wasserfallmodell	Sequentiell	<ul style="list-style-type: none"> • Entwicklungsaktivitäten werden nacheinander abgeschlossen • Testen findet erst statt, wenn alle anderen Entwicklungsaktivitäten abgeschlossen sind
V-Modell	Sequentiell, Testen ist in den Entwicklungsprozess integriert	<ul style="list-style-type: none"> • Teststufen stehen in Bezug zu jeder Entwicklungsphase • Sequentielle Durchführung der Tests, die zur jeweiligen Entwicklungsstufe gehören
z.B. RUP, Scrum, Kanban, Spiralmodell	Inkrementell	<ul style="list-style-type: none"> • Festlegung von Anforderungen, Entwurf, Implementierung und Test des Systems in Teilen • Software-Features wachsen inkrementell • Größe der Inkremeante variiert
	Iterativ	<ul style="list-style-type: none"> • Gruppen von Features werden in Zyklen spezifiziert, entworfen, implementiert und getestet • Jede Iteration liefert eine lauffähige Software

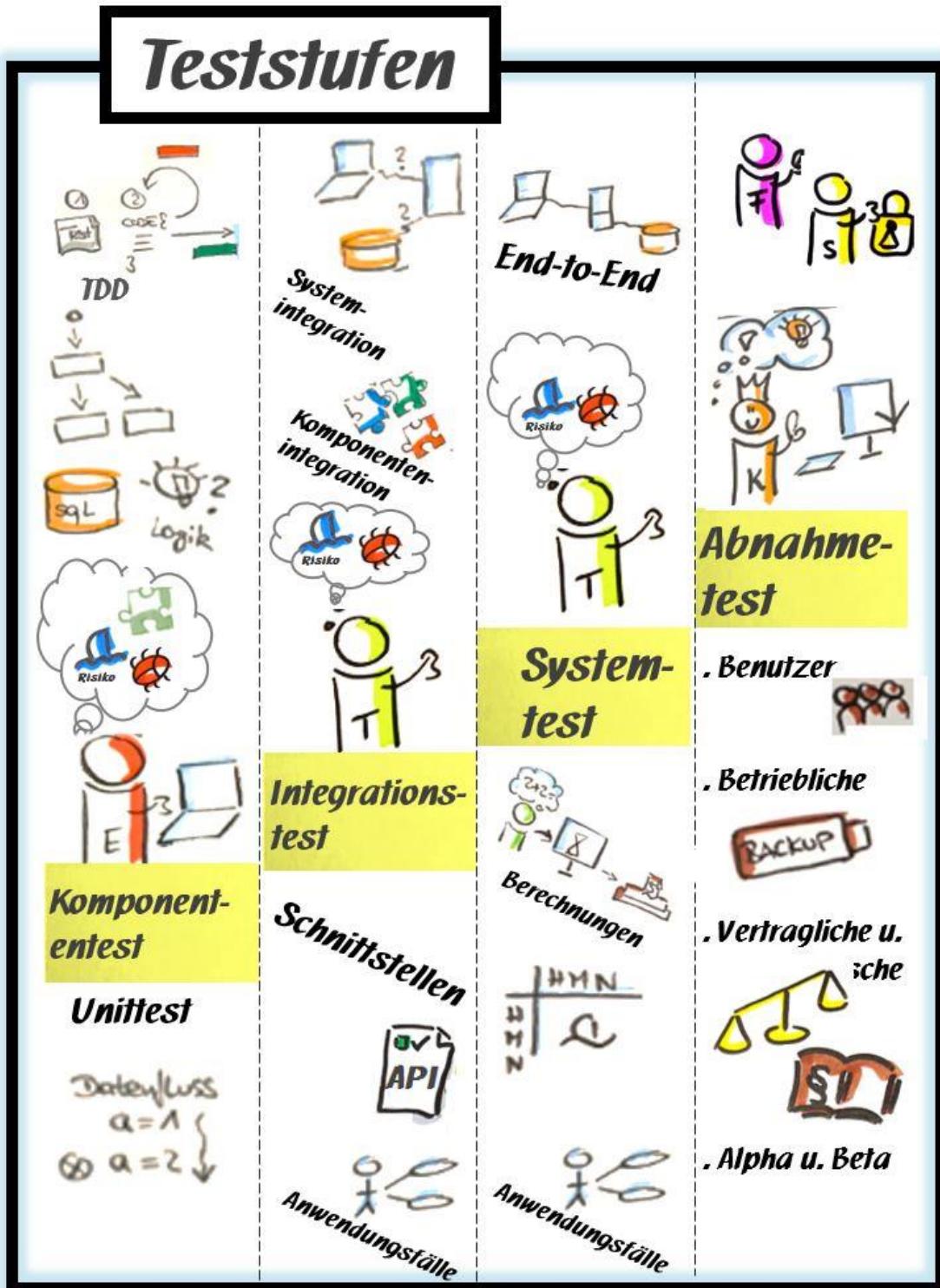
Besonders die iterativ-inkrementellen Modelle erfordern von Softwaretestern häufig eine Automatisierung der Tests, vor allem falls die Software kontinuierlich bereitstellt werden soll („continuous integration“, „continuous delivery“, „continuous deployment“). Dadurch erfolgt die Lieferung eines potentiell nutzbaren Softwareprodukts auch nicht – wie bei sequentiellen Projekten – am Ende des Projekts, sondern laufend innerhalb von Tagen oder wenigen Wochen.

Regressionstests (siehe Kapitel 2.3) sind besonders in der agilen Softwareentwicklung von besonderer Bedeutung, da die Software mit jedem Inkrement wächst, und sichergestellt werden soll, dass Dinge, die bisher funktioniert haben, auch nach der iterativ-inkrementellen Weiterentwicklung weiterhin funktionieren. Durch die häufige Wiederholung dieser Tests eignen sich Regressionstests besonders zur Testautomatisierung (siehe Kapitel 6.1).

Die Anpassung der Softwaretests an den Kontext eines Projekts und damit an den verwendeten Softwareentwicklungszyklus ist eine wichtige Aufgabe bei der Planung der erforderlichen Testaktivitäten. Häufig werden in IT-Projekten auch verschiedene Modelle miteinander kombiniert, oder an spezifische Bedingungen innerhalb eines Unternehmens, einer Branche, oder eines bestimmten Typs von Produkten angepasst, z.B. **kommerzielle Standardsoftware**, commercial off-the-shelf (COTS), Internet of Things (IoT), u.a..

2.2 Teststufen

- FL-2.2.1 (K2) Die unterschiedlichen Teststufen unter den Aspekten der Testziele, Testbasis, Testobjekte, typischen Fehlerzustände und Fehlerwirkungen sowie der Testvorgehensweise und Verantwortlichkeiten vergleichen können



Beispiele	Übungen	Schlüsselbegriffe
2.2.1: Teststufen		Abnahmetest, Alpha-Test, Benutzerabnahmetest, Beta-Test, betrieblicher Abnahmetest, Integrationstest, Komponentenintegrationstest, Komponententest, regulatorischer Abnahmetest, Systemintegrationstest, Systemtest, Testbasis, Testfall, Testobjekt, Teststufe, Testumgebung, Testziel, vertraglicher Abnahmetest

Erläuterungen

Das Flipchart stellt tabellarisch die Ziele, Testbasis, Testobjekte sowie gängige Fehlerzustände und Fehlerwirkungen der einzelnen **Teststufen**, wie im Lehrplan, Abschnitt 2.2, beschrieben, dar.

Wie im vorherigen Abschnitt beschrieben, müssen die Teststufen nicht unbedingt sequentiell ablaufen, sondern können in iterativ-inkrementellen Softwareentwicklungslebenszyklen durchaus auch parallel, z.B. für jedes Feature oder jede User Story, durchgeführt werden. Ebenso ist eine Überlappung der Teststufen möglich, auch zusätzliche Teststufen sind in der Praxis häufig anzutreffen, z.B. der **Systemintegrationstest** als Integrationstest von bereits fertig integrierten Teilsystemen.

Die genannten Eigenschaften der Teststufen sind dabei jedoch, unabhängig vom Entwicklungsmodell, in jedem Fall vertreten.

Zusätzlich charakteristisch für eine Teststufe ist, dass die Tests in einer der Teststufe angepassten **Testumgebung** durchgeführt werden sollten.

Im **Komponententest** ist dies in der Regel der PC des Entwicklers selbst. Der Entwickler kann die Testgetriebene Entwicklung anwenden und die Werkzeuge sind in einer Integrierten Entwicklungsumgebung (IDE) lokal verfügbar.

Im **Integrationstest** werden die unterschiedlichen Komponenten der Entwickler zusammen mit Platzhaltern (Stubs, Driver, Mocks) für noch nicht vorhandene Systembestandteile integriert. Häufig verwendet man das Prinzip der kontinuierlichen Integration, bei dem die integrierte Software automatisiert integriert, getestet und kompiliert wird.

Der **Systemtest** sollte in einer produktionsähnlichen Umgebung stattfinden.

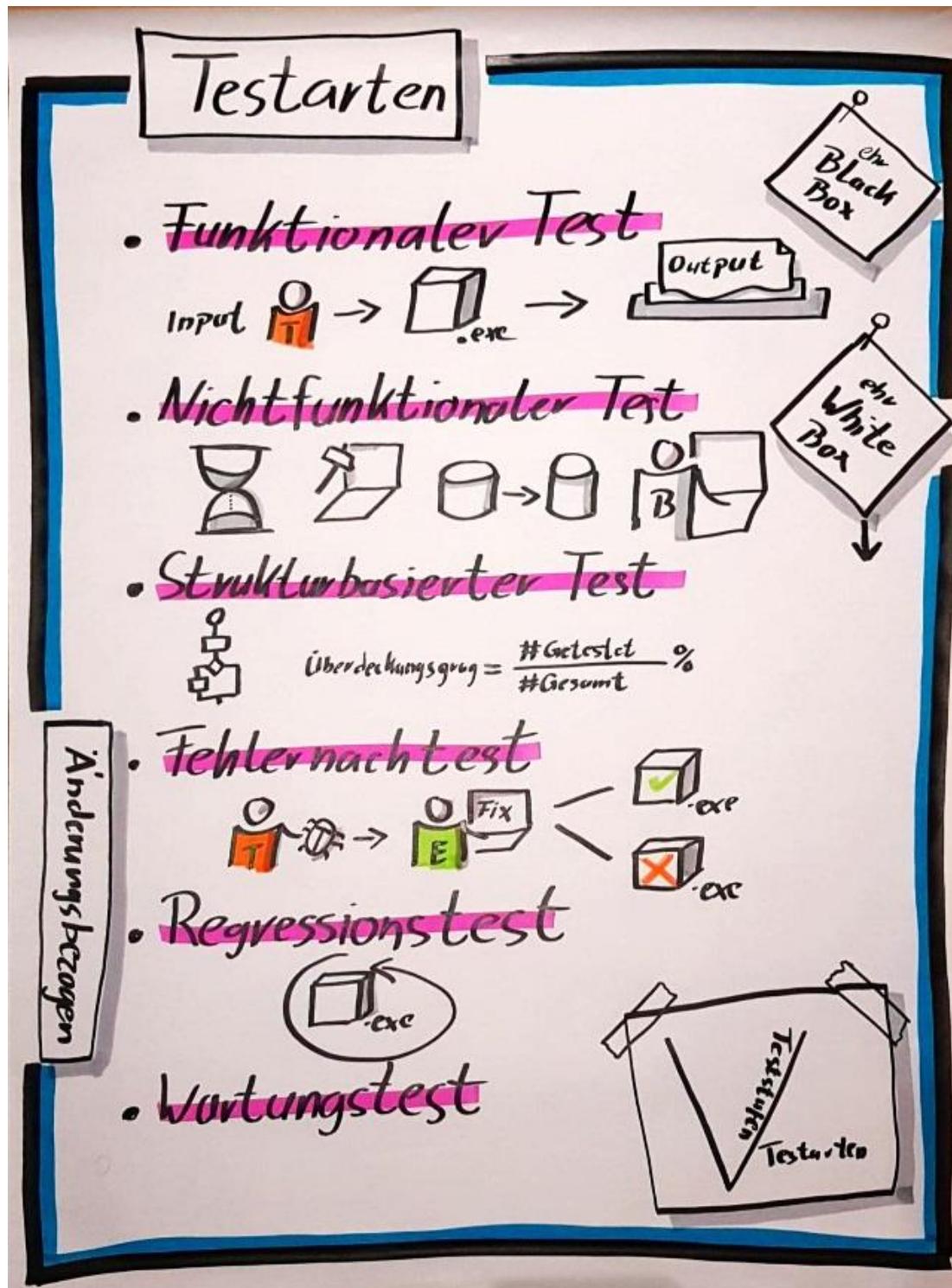
Beim **Abnahmetest** unterscheidet man zwischen dem Alpha-Test in einer Umgebung beim Hersteller und dem Beta-Test (Feldtest) am Kundenstandort, aber auch zwischen Benutzerabnahmetest, betrieblichem Abnahmetest sowie vertraglichem und regulatorischem Abnahmetest.

Die folgende Tabelle bietet noch einmal eine zusammenfassende Übersicht. Die vier Teststufen bieten hier natürlich nur eine mögliche Klassifizierung, in der Praxis können die unterschiedlichen Teststufen durchaus auch differenzierter ausgeprägt sein.

Beispiele	Komponententest	Integrationstest	Systemtest	Abnahmetest
Ziele	<ul style="list-style-type: none"> Risikoreduktion Verifizieren der Spezifikation Schaffen von Vertrauen in die Qualität der Komponente Finden von Fehlerzuständen in der Komponente Weitergeben von Fehlerzuständen verhindern 	<ul style="list-style-type: none"> Risikoreduktion Verifizieren der Spezifikation Vertrauen in die Qualität des Systems als Ganzes schaffen Finden von Fehlerzuständen Weitergeben von Fehlerzuständen verhindern 	<ul style="list-style-type: none"> Risikoreduktion Vertrauen in die Qualität des Systems als Ganzes schaffen Finden von Fehlerzuständen Weitergeben von Fehlerzuständen verhindern 	<ul style="list-style-type: none"> Verifizieren der Spezifikation Vertrauen in die Qualität des Systems als Ganzes schaffen Finden von Fehlerzuständen Weitergeben von Fehlerzuständen verhindern
Testbasis	<ul style="list-style-type: none"> Feinentwurf Code Datenmodelle Komponentenspezifikation 	<ul style="list-style-type: none"> Software- und Systementwurf Sequenzdiagramme Spezifikationen von Schnittstellen und Protokollen Anwendungsfälle Architektur auf Komponenten- oder Systemebene Workflows Ext. Schnittstellendefinitionen 	<ul style="list-style-type: none"> Anforderungsspezifikationen Risikoanalyseberichte Anwendungsfälle Epics und User-Stories Modelle des Systemverhaltens Zustandsdiagramme System- und Benutzeranleitungen 	<ul style="list-style-type: none"> Geschäftsprozesse Anforderungen, Use-Cases Vorschriften, Verträge, Standards Dokumentation Installationsverfahren Risikoanalyseberichte Backup & Restore-Verfahren Disaster-Recovery-Verfahren
Testobjekte	<ul style="list-style-type: none"> Komponenten, Units, Module Code und Datenstrukturen Klassen Datenbankmodule 	<ul style="list-style-type: none"> Subsysteme Datenbanken Infrastruktur Schnittstellen APIs Microservices 	<ul style="list-style-type: none"> Anwendungen Hardware/Softwaresysteme Betriebssysteme Systeme unter Test (SUT) Systemkonfiguration und Konfigurationsdaten 	<ul style="list-style-type: none"> System unter Test (SUT) Systemkonfigurationen Geschäftsprozesse Wiederherstellungssysteme Betriebs- und Wartungsprozesse Formulare und Berichte Produktionsdaten
Fehlerzustände und Fehlerwirkungen	<ul style="list-style-type: none"> Nicht korrekte Funktionalität Datenflussprobleme Nicht korrekter Code und nicht korrekte Logik 	<ul style="list-style-type: none"> Falsche Daten, fehlende Daten, falsche Datenverschlüsselung Falsche Abfolge von Schnittstellenaufrufen Fehler in der Kommunikation zwischen Komponenten Falsche Bedeutung, Einheiten oder Grenzen der Daten Inkonsistente Strukturen zwischen Systemen Fehlende Konformität mit erforderlichen Richtlinien 	<ul style="list-style-type: none"> Falsche Berechnungen Falsche oder unerwartete funktionale oder nicht-funktionale Systemverhaltensweisen Falsche Kontroll- und/oder Datenflüsse Versagen bei der Ausführung von End-to-End-Aufgaben Versagen des Systems in der Produktivumgebung Das System funktioniert nicht wie in den Anleitungen beschrieben 	<ul style="list-style-type: none"> Systemworkflows erfüllen nicht die Anforderungen Geschäftsregeln wurden nicht korrekt umgesetzt Nichterfüllen der vertraglichen oder regulatorischen Anforderungen Sicherheitschwachstellen, nicht angemessene Performanz, nicht ordnungsgemäßer Betrieb auf einer unterstützten Plattform
Spezifische Ansätze und Verantwortlichkeiten	<ul style="list-style-type: none"> Entwicklertests Testgetriebene Entwicklung Refactoring 	<ul style="list-style-type: none"> Komponentenintegration (Entwickler) Systemintegration (Tester) Integrationsstrategien (Top-down, Bottom-up) Kontinuierliche Integration 	<ul style="list-style-type: none"> Unabhängige Tester End-to-End-Verhalten 	<ul style="list-style-type: none"> Kunde, Fachanwender, Product Owner, Betreiber Benutzeraufnahme / Betriebliche Abnahme / Vertragliche oder regulatorische Abnahme Alpha- und Betaetest

2.3 Testarten

- FL-2.3.1 (K2) Funktionale, nicht-funktionale und White-Box-Tests vergleichen können
- FL-2.3.2 (K1) Erkennen können, dass funktionale, nicht-funktionale und White-Box-Tests auf jeder Teststufe eingesetzt werden können
- FL-2.3.3 (K2) Den Zweck von Fehlernachtests und Regressionstests vergleichen können



Beispiele	Übungen	Schlüsselbegriffe
2.3.1: Testarten 2.3.3: Fehlernachttest und Regressionstest		Fehlernachttest, funktionaler Test, nicht-funktionaler Test, Regressionstest, Testart, White-Box-Test

Erläuterungen

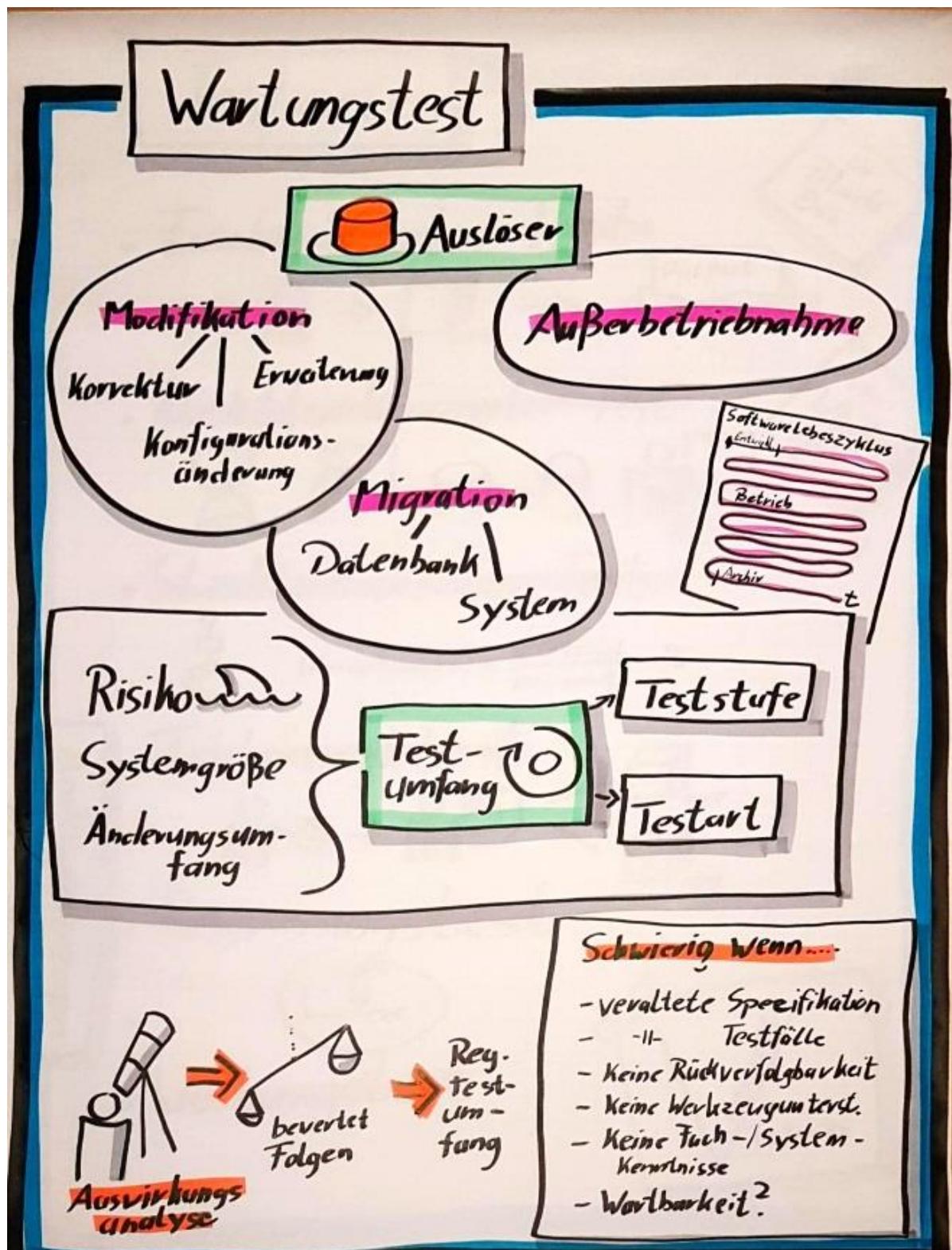
Während Teststufen sich auf die zeitlichen bzw. organisatorischen Zusammenhänge zwischen Testaktivitäten beziehen, hat eine **Testart** den Zweck, ein System im Hinblick auf bestimmte Qualitätsmerkmale oder spezifische Testziele hin zu testen. Dementsprechend gliedern sich die Testarten in funktionale Tests, nicht-funktionale Tests, White-Box Tests und Änderungsbezogene Tests.

Grundsätzlich können alle Testarten in allen Teststufen durchgeführt werden. Allerdings werden in den niedrigeren Teststufen **Komponententest** und **Integrationstests** vermehrt White-Box-Techniken angewandt, die Wissen über die interne Struktur der Software nutzen. In den höheren Teststufen **Systemtest** und **Abnahmetest** verwendet man dagegen eher die spezifikationsorientierten Black-Box-Testverfahren.

- **Funktionalität** testet das Ein- / Ausgabeverhalten der Software. Bricht man ein Produkt auf einzelne funktionale Bestandteile herunter, bemisst sich die funktionale Überdeckung als der Anteil der getesteten Funktionalitäten an allen Funktionalitäten. Das Erstellen funktionaler Tests folgt bestimmten Testentwurfsverfahren und erfordert häufig neben den Kompetenzen eines Softwaretesters auch Fachkenntnisse der spezifischen Anwendungen
- Mit **nicht-funktionalen Tests** werden Qualitätsmerkmale einer Software überprüft, die sich auf die Gebrauchstauglichkeit, Performanz oder Informationssicherheit beziehen. Auch hier lässt sich die Überdeckung der Tests messen als der Anteil getesterter Elemente an allen Elementen eines Testobjekts, und auch für die nicht-funktionalen Tests macht die Verwendung von Testentwurfsverfahren Sinn.
- **White-Box-Tests** (oder Strukturbasierte Tests) setzen Kenntnisse über die innere Struktur der Software oder einzelner Komponenten voraus, und nutzen diese Kenntnisse für den Testentwurf. Überdeckungsmaße messen die Anzahl getester Strukturelemente (Codezeilen, Komponenten, Schritte eines Geschäftsprozesses) als Anteil aller Strukturelemente. Dabei kann es sich auch um eine Menüstruktur oder Architekturkomponenten handeln. Der Testentwurf für White-Box-Tests verwendet spezielle Verfahren, die sich von denen funktionaler Tests unterscheiden.
- Änderungen an der Software werden durchgeführt bei der Fehlerbehebung, während der regulären Weiterentwicklung, sowie bei der Wartung bereits in Betrieb befindlicher Software.
 - **Fehlernachtests** haben den Zweck zu bestätigen, dass ein Fehler behoben wurde, d.h. es sind mindestens die Testfälle erneut zu testen, die aufgrund des behobenen Fehlers fehlgeschlagen sind.
 - **Regressionstests** sollen sicherstellen, dass bei Änderungen an der Software keine neuen Fehler an bereits bestehenden Funktionalitäten eingebracht wurden. Es wird also von Regressionstests, im Gegensatz zu anderen Testarten, erwartet, dass sie keine neuen Fehler aufdecken. Hierbei ist allerdings insbesondere das Pestizid-Paradoxon zu beachten.
 - **Wartungstests** werden separat im folgenden Abschnitt beschrieben.

2.4 Wartungstest

- FL-2.4.1 (K2) Auslöser für Wartungstests zusammenfassen können
- FL-2.4.2 (K2) Den Einsatz der Auswirkungsanalyse im Wartungstest beschreiben können



Beispiele	Übungen	Schlüsselbegriffe
2.4.1: Wartungsanlässe 2.4.2: Auswirkungsanalyse		Auswirkungsanalyse, Wartungstest Auswirkungsanalyse, Wartungstest

Erläuterungen

Wartung eines Systems setzt voraus, dass dieses bereits produktiv eingesetzt wird. Insofern spielen Wartungstests eine besondere Rolle, da die Geschäftsfähigkeit eines Unternehmens, echte Kundendaten, die IT-Sicherheit und der Arbeitsalltag der mit der Software arbeiten Mitarbeiter oder Kunden von einer Änderung unmittelbar betroffen sein könnten.

Der Lehrplan, Abschnitt 2.4 nennt drei Faktoren, die den Umfang von Wartungstests bestimmen:

- Die Risikohöhe der Änderung
- Die Größe des bestehenden Systems
- Der Größe der Änderung

Außerdem werden die folgenden Wartungsanlässe (Auslöser) genannt:

- Modifikation, wie geplante Verbesserungen, korrigierende Änderungen, Notfalländerungen, Änderungen der betrieblichen Umgebung, Aktualisierungen von Standardsoftware und Patches für Fehlerzustände und Schwachstellen.
- Migration, wie von einer Plattform zu einer anderen oder Tests der Datenkonvertierung
- Außerbetriebnahme

Aufgrund des in der Regel hohen Risikos eines Wartungsreleases und potenziell hohen Auswirkungen entstehender Schäden, wird für Wartungstests eine **Auswirkungsanalyse** durchgeführt. Dabei wird die Veränderung an der Software analysiert, mögliche Nebeneffekte, auch auf bestehende Tests, analysiert, um neben den Notwendigen Anpassungen auch den Umfang nötiger Regressionstests zu bestimmen.

3. Statischer Test

Flipchart-Bilder

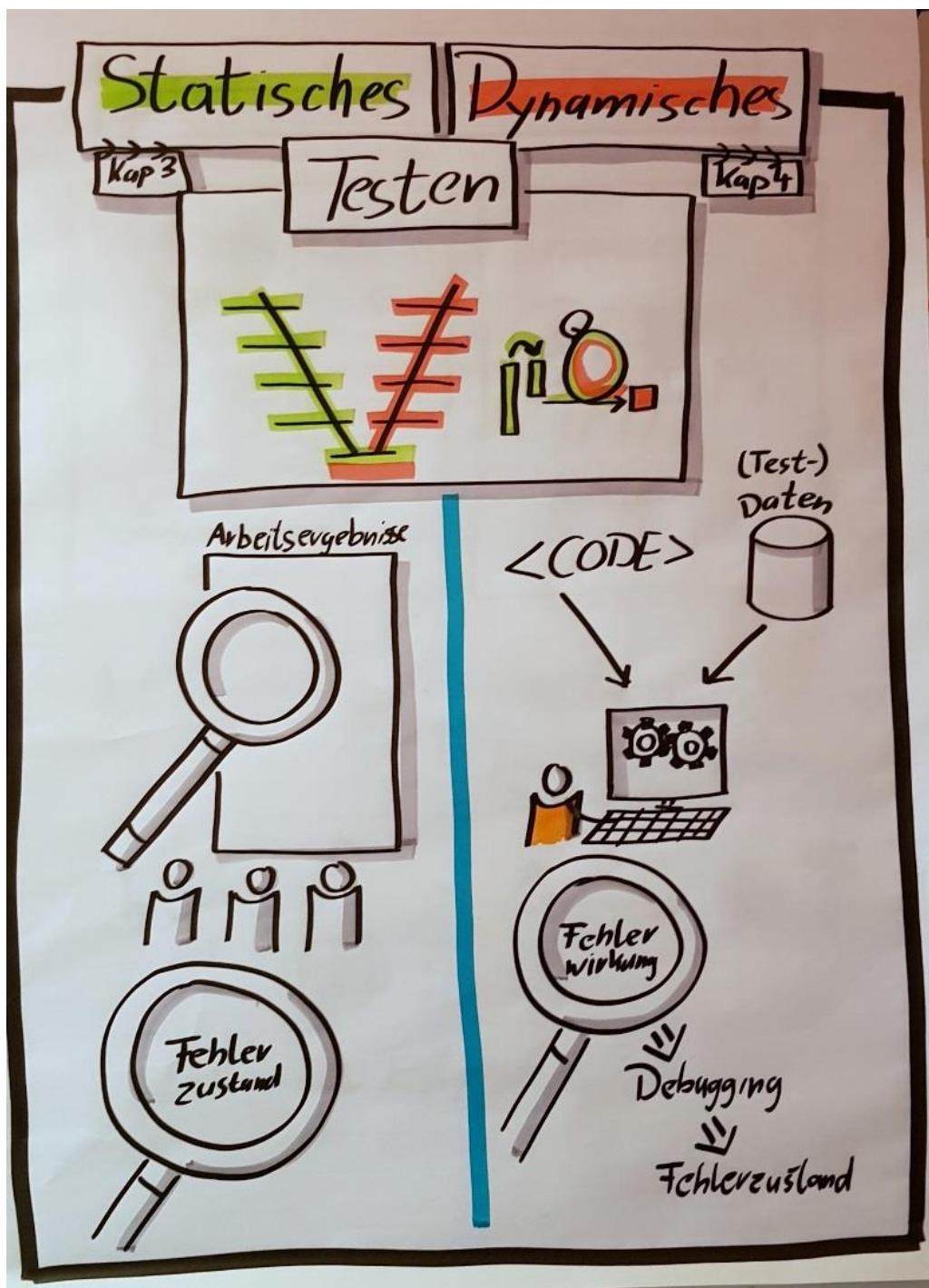
- Statisches vs. Dynamisches Testen
- Review-Prozess
- Review-Rollen
- Review-Arten

Schlüsselbegriffe im Kapitel

Ad-hoc-Review, checklistenbasiertes Review, dynamischer Test, formales Review, informelles Review, Inspektion, perspektivisches Lesen, Review, rollenbasiertes Review, statische Analyse, statischer Test, szenariobasiertes Review, technisches Review, Walkthrough

3.1 Grundlagen des statischen Tests

- FL-3.1.1 (K1) Arten von Softwarearbeitsergebnissen erkennen können, die durch die verschiedenen statischen Testverfahren geprüft werden können
- FL-3.1.2 (K2) Beispiele nennen können, um den Wert des statischen Tests zu beschreiben
- FL-3.1.3 (K2) Den Unterschied zwischen statischen und dynamischen Verfahren unter Berücksichtigung der Ziele, der zu identifizierenden Fehlerzustände und der Rolle dieser Verfahren innerhalb des Softwarelebenszyklus erklären können



Beispiele	Übungen	Schlüsselbegriffe
3.1.2: Wert des statischen Tests 3.1.3: Statisch oder dynamisch?		dynamischer Test, Review, statische Analyse, statischer Test

Erläuterungen

Statische Tests überprüfen die Arbeitsergebnisse, die während der Entwicklung eines Softwareprodukts entstehen. **Dynamische Tests** dagegen erfordern immer die Ausführung des Testobjekts, also der zu testenden Software.

Statische Tests unterteilt man weiter in **Reviews** und **statische Analysen**, wobei aus Sicht des Softwaretesters vor allem die Reviews interessant sind. Dabei werden insbesondere Dokumente nach festgelegten Verfahren und nach einem festgelegten Prozess manuell geprüft.

Statische Analysen dagegen prüfen Code oder auch maschinell lesbare Dokumente automatisiert und unter Verwendung bestimmter Entwicklungswerkzeuge. Daher ist die Tätigkeit der statischen Analysen in der Regel den Entwicklern überlassen.

Beide Arten von Tests haben vor allem den Vorteil, Fehlerzustände frühzeitig aufzudecken, und dadurch zu verhindern, dass diese in den dynamischen Test weitergereicht werden.

Der Lehrplan, Abschnitt 3.1.2, listet eine Reihe weiterer Vorteile:

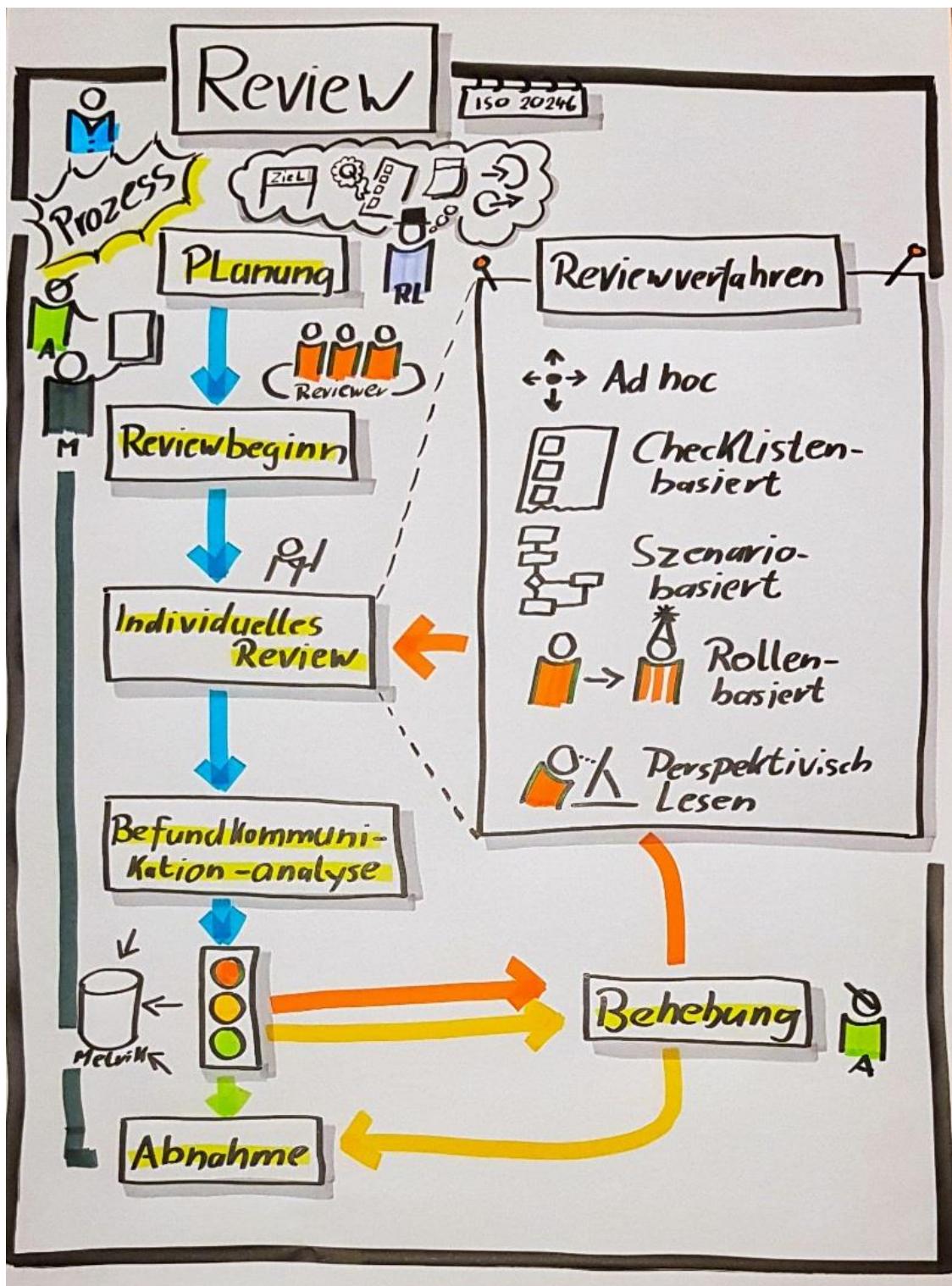
- *Effizienteres Erkennen und Korrigieren von Fehlerzuständen schon vor der Durchführung dynamischer Test*
- *Identifizieren von Fehlerzuständen, die in dynamischen Tests nicht leicht zu finden sind*
- *Verhindern von Fehlerzuständen im Entwurf oder der Kodierung durch das Aufdecken von Inkonsistenzen, Mehrdeutigkeiten, Widersprüchen, Auslassungen, Ungenauigkeiten und Redundanzen in Anforderungen*
- *Erhöhen der Entwicklungsproduktivität (z.B. durch verbesserte Entwürfe, mehr wartungsfähigen Code)*
- *Reduzieren von Entwicklungskosten und -zeit*
- *Reduzieren von Testkosten und -zeit*
- *Reduzieren der Gesamtkosten der Qualität über die Lebenszeit der Software hinweg, aufgrund von weniger Fehlerwirkungen zu einem späteren Zeitpunkt im Lebenszyklus oder nach Auslieferung in die Produktion*
- *Verbesserte Kommunikation zwischen Teammitgliedern durch die Teilnahme an Reviews*

Zudem finden Statische Tests andere Arten von Fehlerzuständen als dynamische Tests, und sind insofern eine sinnvolle Ergänzung. Der Lehrplan, Abschnitt 3.1.3 nennt hier explizit *Anforderungsfehler, Entwurfsfehler, Programmierfehler, Abweichungen von Standards, falsche Schnittstellenspezifikationen, Schwachstellen in der Zugriffssicherheit und Lücken oder Ungenauigkeiten in der Verfolgbarkeit oder der Überdeckung der Testbasis*.

Die genannten Vorteile zielen u.a. auch darauf ab, durch vorbeugende Maßnahmen in der Analysephase des Testprozesses die Testbarkeit herzustellen. So kann in einem Testbarkeitsreview besonders Wert gelegt werden auf Anforderungen und Entwürfe, die konkret belastbare Spezifikationen oder quantifizierbare Werteangaben, z.B. Zeitangaben für Performanzests, enthalten.

3.2 Reviewprozess

- FL-3.2.1 (K2) Die Aktivitäten des Reviewprozesses für Arbeitsergebnisse zusammenfassen können
- FL-3.2.4 (K3) Ein Reviewverfahren auf ein Arbeitsergebnis anwenden können, um Fehlerzustände zu finden



Beispiele	Übungen	Schlüsselbegriffe
3.2.1: Reviewprozess	Übung 1: Reviewprozess	Ad-hoc-Review, checklisten-basiertes Review, perspektivisches Lesen, rollenbasiertes Review, szenariobasiertes Review

Erläuterungen

Der bei Dokumentenreviews durchzuführende Prozess umfasst 5 Schritte, die im Flipchart in einer sinnvollen Reihenfolge beschrieben werden. Natürlich macht eine Abnahme des Arbeitsergebnisses nur dann Sinn, wenn vorher festgelegte Endekriterien erreicht sind, andernfalls sind Reviewbefunde zunächst zu beheben und das Dokument muss einem erneuten Review unterzogen werden.

Die folgende Tabelle gibt einen Überblick über die Aktivitäten im Review-Prozess [Zitat Lehrplan, Abschnitt 3.2.1]:

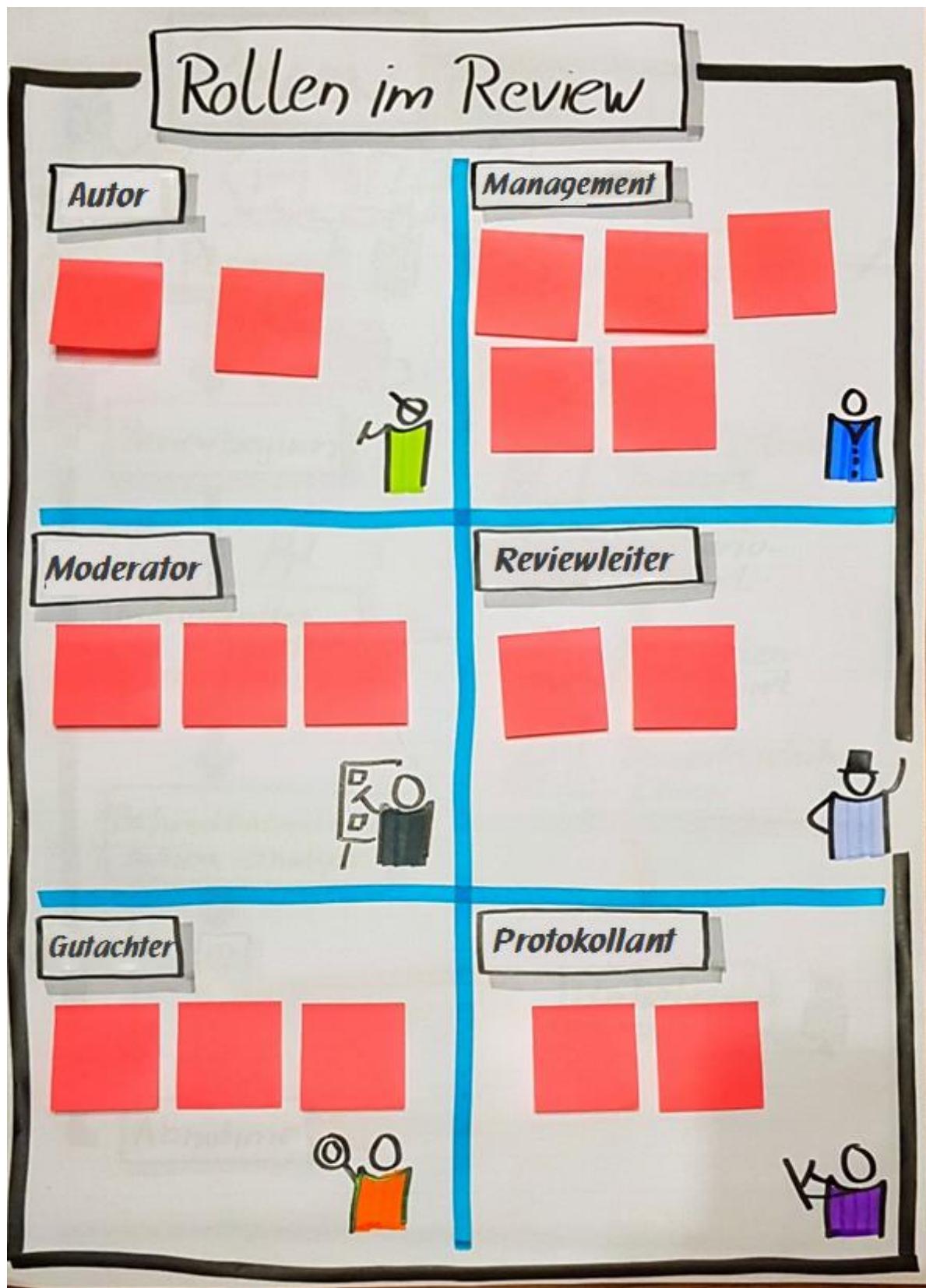
Hauptaktivität	Aktivitäten
Planung	<ul style="list-style-type: none"> • Definition des Umfangs inkl. des Zwecks des Reviews, welche Dokumente oder Teile von Dokumenten im Review einbezogen werden sollen und der Qualitätsmerkmale, die bewertet werden sollen • Schätzung von Aufwand und Zeitbedarf • Identifizieren von Revieweigenschaften wie der Reviewart mit Rollen, Aktivitäten und Checklisten • Auswahl der Personen, die am Review teilnehmen sollen, und Zuordnung der Rollen • Definition der Eingangs- und Endekriterien für formalere Reviewarten (z.B. Inspektionen) • Prüfung, ob Eingangskriterien erfüllt sind (für formalere Reviewarten)
Reviewbeginn	<ul style="list-style-type: none"> • Verteilen des Arbeitsergebnisses (physisch oder elektronisch) und anderer Materialien, wie Befund-Template, Checklisten und zugehörige Arbeitsergebnisse • Erläutern des Umfangs, der Ziele, des Prozesses, der Rollen und der Arbeitsergebnisse gegenüber den Teilnehmern • Beantwortung von Fragen, die die Teilnehmer zum Review haben könnten
Individuelles Review	<ul style="list-style-type: none"> • Review des gesamten oder von Teilen des Arbeitsergebnisses • Aufzeichnung potenzieller Fehlerzustände, Empfehlungen und Fragen
Befundkommunikation und -analyse	<ul style="list-style-type: none"> • Kommunikation identifizierter potenzieller Fehlerzustände (z.B. in einer Reviewsitzung) • Analyse potenzieller Fehlerzustände, Zuweisung von Zuständigkeit und Status • Bewertung und Dokumentation von Qualitätsmerkmalen • Bewertung der Reviewbefunde gegenüber den Endekriterien, um eine Reviewentscheidung zu treffen (ablehnen, umfangreiche Änderungen notwendig, annehmen, vielleicht mit geringfügigen Änderungen)

Hauptaktivität	Aktivitäten
Fehlerbehebung und Bericht	<ul style="list-style-type: none"> • Für die Befunde, die Änderungen erfordern, Fehlerberichte erstellen • Beheben von im geprüften Arbeitsergebnis gefundenen Fehlerzuständen (üblicherweise durch den Autor) • Kommunikation von Fehlerzuständen an die zuständige Person oder das zuständige Team (wenn sie in einem Arbeitsergebnis gefunden wurden, das zu dem Arbeitsergebnis, welches geprüft wurde, in Beziehung steht) • Aufzeichnung des aktualisierten Status der Fehlerzustände (in formalen Reviews), potenziell auch mit der Zustimmung der Person, die den Befund erstellte • Sammeln von Metriken (für formalere Reviewarten) • Prüfen, dass Endekriterien erfüllt sind (für formalere Reviewarten) • Abnahme des Arbeitsergebnisses, wenn die Endekriterien erreicht sind

Die Erkennung von Fehlerzuständen im individuellen Review wird durch eine Reihe von Reviewverfahren unterstützt, die auf dem Flipchart gelistet und im Lehrplan, Abschnitt 3.2.4, erläutert sind. Die Verfahren basieren auf einem systematischen Vorgehen, lassen den Gutachter ein Dokument aus unterschiedlichen Rollen oder Perspektiven betrachten, oder spielen Probeläufe oder Szenarien exemplarisch durch.

FL-3.2.2 (K1) Die unterschiedlichen Rollen und Verantwortlichkeiten in einem formalen Review erkennen können

FL-3.2.5 (K2) Die Faktoren erklären können, die zu einem erfolgreichen Review beitragen



Beispiele	Übungen	Schlüsselbegriffe
3.2.5: Erfolgreiche Reviews		

Erläuterungen

Das Flipchart stellt die Rollen und deren Aufgaben im Review-Prozess dar. Je nach Grad der Formalität in einem Review ist die Besetzung einer Rolle mehr oder weniger wichtig. Findet in einem informellen Review z.B. keine Reviewsitzung statt, sind in der Regel auch kein Moderator und kein Protokollant erforderlich, bzw. können deren Aufgaben auch durch den Autor oder Gutachter übernommen werden.

Dagegen ist es in formaleren Reviews umso wichtiger, den Rollen unterschiedliche Personen zuzuweisen, um z.B. die Neutralität eines Moderators gewährleisten zu können.

Die Auswahl eines geeigneten Review-Verfahrens und einer geeigneten Review-Art gehören zu den Erfolgsfaktoren, die das Ergebnis des Reviews beeinflussen. Sowohl die Organisation, als auch die Auswahl der Personen sowie psychologische Aspekte spielen dabei eine Rolle, der Lehrplan, Abschnitt 3.2.5, listet eine Reihe von Erfolgsfaktoren auf.

- ➔ Die Aufgaben der einzelnen Rollen im Review-Prozess gemäß Lehrplan, Abschnitt 3.2.2 können hier wie in der Schulung besprochen eingetragen werden.

FL-3.2.3 (K2) Die Unterschiede zwischen den unterschiedlichen Reviewarten erklären können: informelles Review, Walkthrough, technisches Review und Inspektion



Beispiele	Übungen	Schlüsselbegriffe
3.2.3: Reviewarten		formales Review, informelles Review, Inspektion, technisches Review, Walkthrough

Erläuterungen

Die für ein Review verwendete Review-Art bestimmt den Grad der Formalität eines Reviews, sowie die mit dem Review verbundenen Eigenschaften und Ziele. Die Review-Arten unterscheiden sich z.B. in der Konsequenz, mit der der Review-Prozess verfolgt wird, in der Rollenaufteilung, in der Art der verwendeten Verfahren, im prinzipiellen Vorgehen und der Art der Dokumentation. Der Lehrplan, Abschnitt 3.2.3, bietet eine gute Übersicht über die Eigenschaften der vier Review-Arten.

Wird ein Review nur unter gleichberechtigten Kollegen durchgeführt, nennt man es Peer-Review.

Ein Arbeitsergebnis kann den Review-Prozess zudem auch mehrfach durchlaufen, unter Verwendung unterschiedlicher Review-Arten und Review-Verfahren. Dadurch ist es zum einen möglich, auch unterschiedliche Arten von Fehlerzuständen aufzudecken, zum anderen aber auch das Review an weitere Anforderungen eines Projekts und des Unternehmens anzupassen.

4. Testverfahren

Flipchart-Bilder

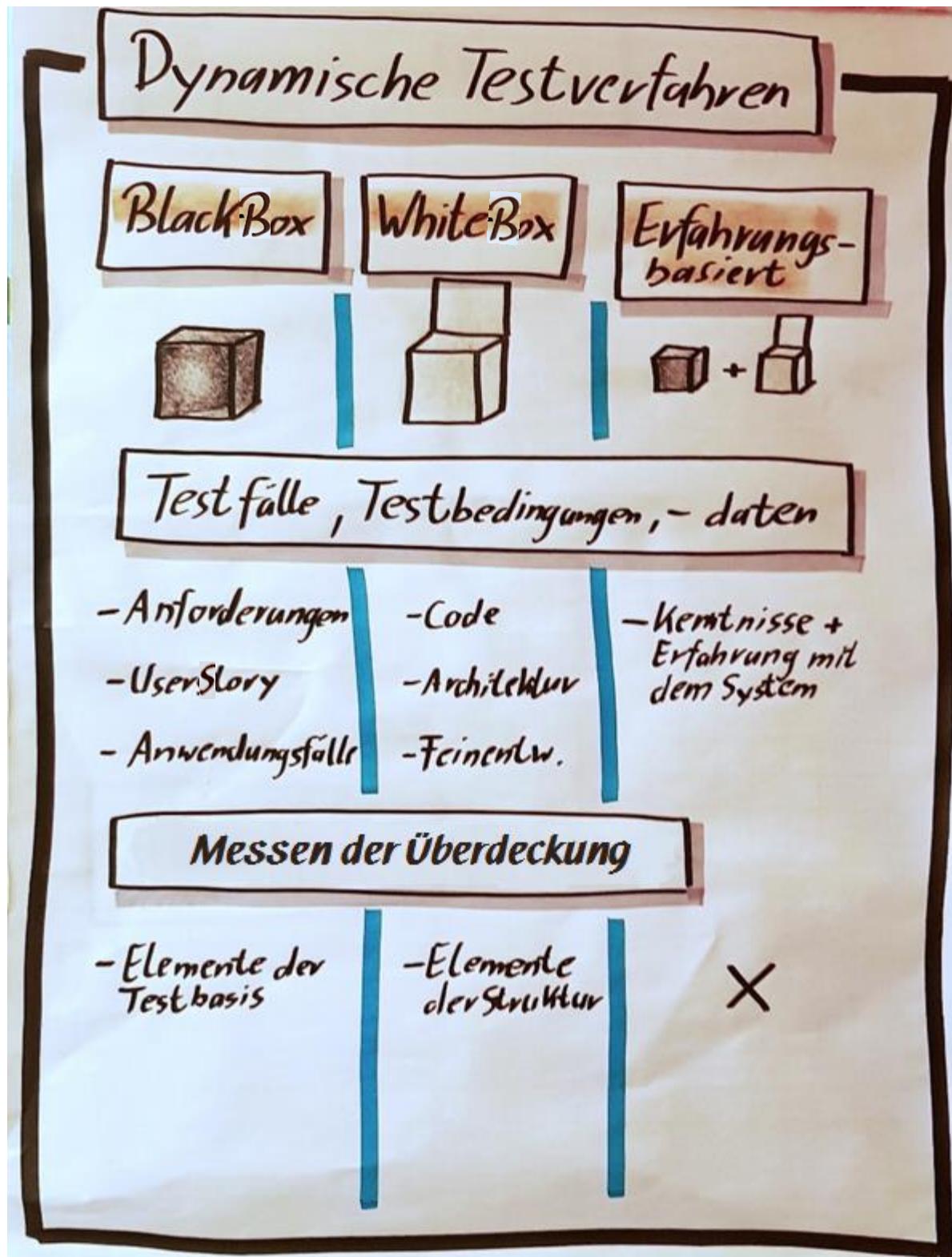
- Dynamische Testverfahren
- Äquivalenzklassen
- Grenzwertanalysen
- Entscheidungstabellen
- Zustandsübergangstest
- White-Box Tests

Schlüsselbegriffe im Kapitel

Anweisungsüberdeckung, Anwendungsfallbasiertes Testen, Äquivalenzklassenbildung, Black-Box-Testverfahren, checklistenbasiertes Testen, Entscheidungstabellentest, Entscheidungsüberdeckung, erfahrungsbasierte Testverfahren, exploratives Testen, Grenzwertanalyse, intuitive Testfallermittlung, Testverfahren, Überdeckung, White-Box-Testverfahren, Zustandsübergangstest

4.1 Kategorien von Testverfahren

- FL-4.1.1 (K2) Die Eigenschaften, Gemeinsamkeiten und Unterschiede zwischen Black-Box-Testverfahren, White-Box-Testverfahren und erfahrungsbasierten Testverfahren erklären können



Beispiele	Übungen	Schlüsselbegriffe
4.1.1: Black or White?		Testverfahren, Überdeckung, Black-Box- Testverfahren, White-Box- Testverfahren

Erläuterungen

Dynamisches Testen benötigt im Unterschied zum Statischen Test ein ausführbares Testobjekt. Testverfahren beschreiben den Testentwicklungsprozess, d.h. das Ableiten von Testbedingungen aus Testbasisdokumenten, das Erstellen von Testfällen, Testszenarien und das ermitteln von Testdaten.

Die Bestimmung, welches Testverfahren verwendet werden soll, wird vor allem durch die Testvorgehensweise festgelegt, in der eine Reihe von Faktoren zu berücksichtigen sind, der Lehrplan, Abschnitt 4.1.1, nennt folgende Beispiele:

- Die Komplexität der Komponente oder des Systems
- Regulatorische Standards
- Kundenanforderungen oder vertragliche Anforderungen
- Risikostufen
- Risikoarten
- Verfügbare Dokumentation
- Kenntnisse und Fähigkeiten des Testers
- Verfügbare Werkzeuge
- Zeit und Budget
- Softwareentwicklungslebenszyklus-Modell
- Die Arten von Fehlerzuständen, die in der Komponente oder dem System erwartet werden

Wir kategorisieren Testverfahren in **Black-Box**, **White-Box** und **erfahrungsisierte Testverfahren**, die sowohl als Black-Box-, als auch White-Box-Testverfahren ausgeprägt sein können. Neben den im Lehrplan, Abschnitt 4.1.1, genannten Merkmalen ist es vor allem der Einblick in die inneren Strukturen der Software (Code, Architektur), der die Verwendung von White-Box Techniken ermöglicht. Fehlt dieser Einblick, und Testfälle können nur auf der Basis von Spezifikationen, Anwendungsfällen oder User Stories entwickelt werden, verwendet man Black-Box Techniken.

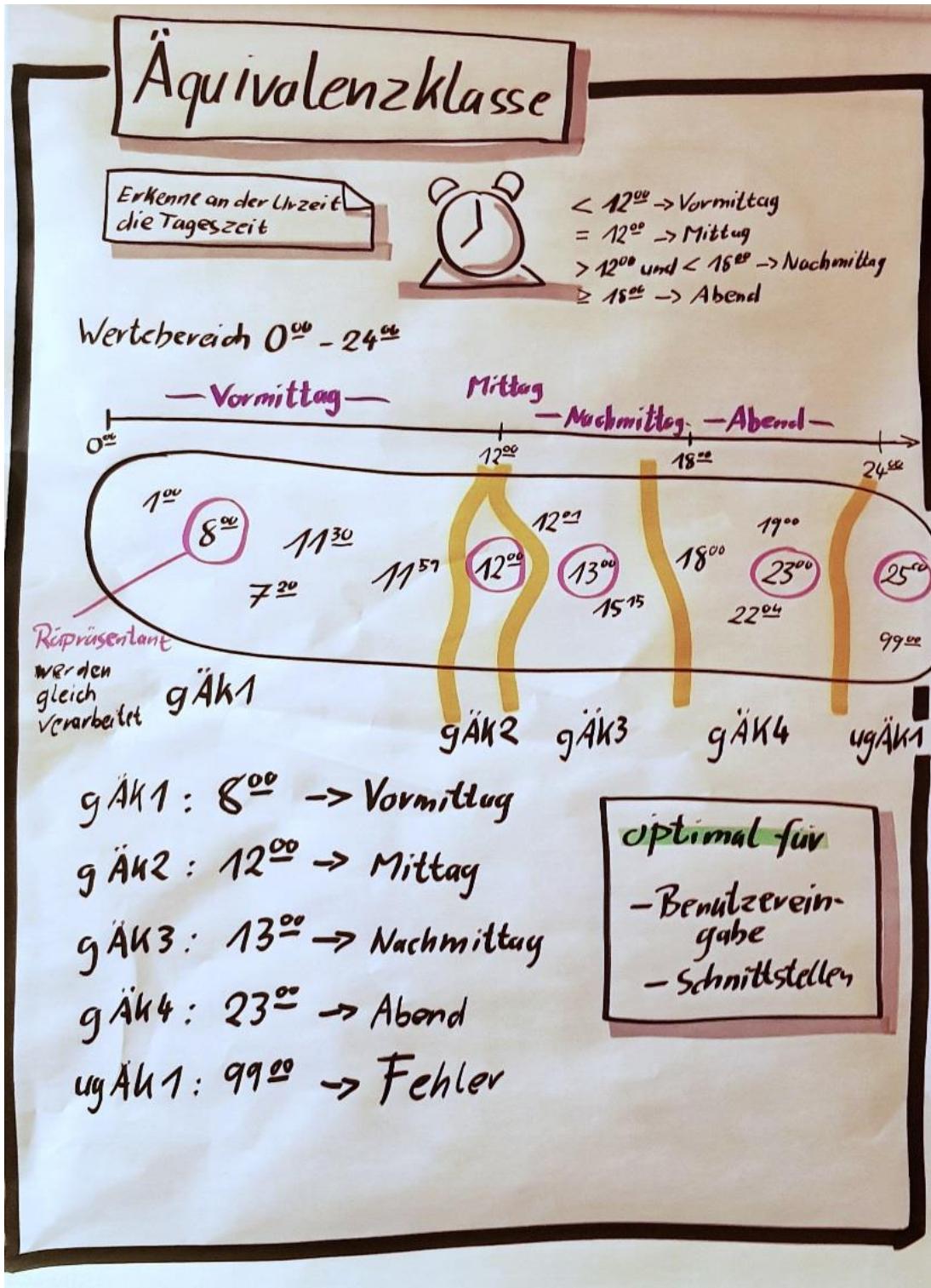
Sowohl bei Black-Box-, als auch bei White-Box-Testverfahren kann die **Überdeckung** der Testbasis durch Tests, also die Testintensität, auf verschiedene Arten gemessen werden.

Erfahrungsisierte Verfahren bilden zudem eine gute Ergänzung, denn hierbei können auch Aspekte der Nutzung, berechtigte Erwartungen an die Software, vermutete Fehlerzustände u.a. überprüft werden. Da sich diese jedoch schlecht quantifizieren lassen, ist eine Überdeckungsmessung im Gegensatz zu Black-Box und White-Box Tests in der Regel nicht möglich.

4.2 Black-Box-Testverfahren

4.2.1 Äquivalenzklassenbildung

FL-4.2.1 (K3) Die Äquivalenzklassenbildung anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten



Beispiele	Übungen	Schlüsselbegriffe
	Übung 2: Äquivalenzklassenbildung	Äquivalenzklassenbildung

Erläuterungen

Im Funktionalen Testen betrachten wir ein System als Black-Box, die auf der Basis einer Eingabe eine Berechnung durchführt, und eine Ausgabe produziert. Alle Eingabewerte, die auf dieselbe Weise verarbeitet werden, bilden eine **Äquivalenzklasse**, was man in der Regel daran erkennt, dass die Elemente einer Äquivalenzklasse von Eingabewerten dieselbe Ausgabe ergeben.

Eine gültige Äquivalenzklasse besteht aus Werten, die vom System verarbeitet (akzeptiert) werden. Lehnt das System die Verarbeitung ab, z.B. bei ungültigen Eingaben, sprechen wir von einer ungültigen Äquivalenzklasse.

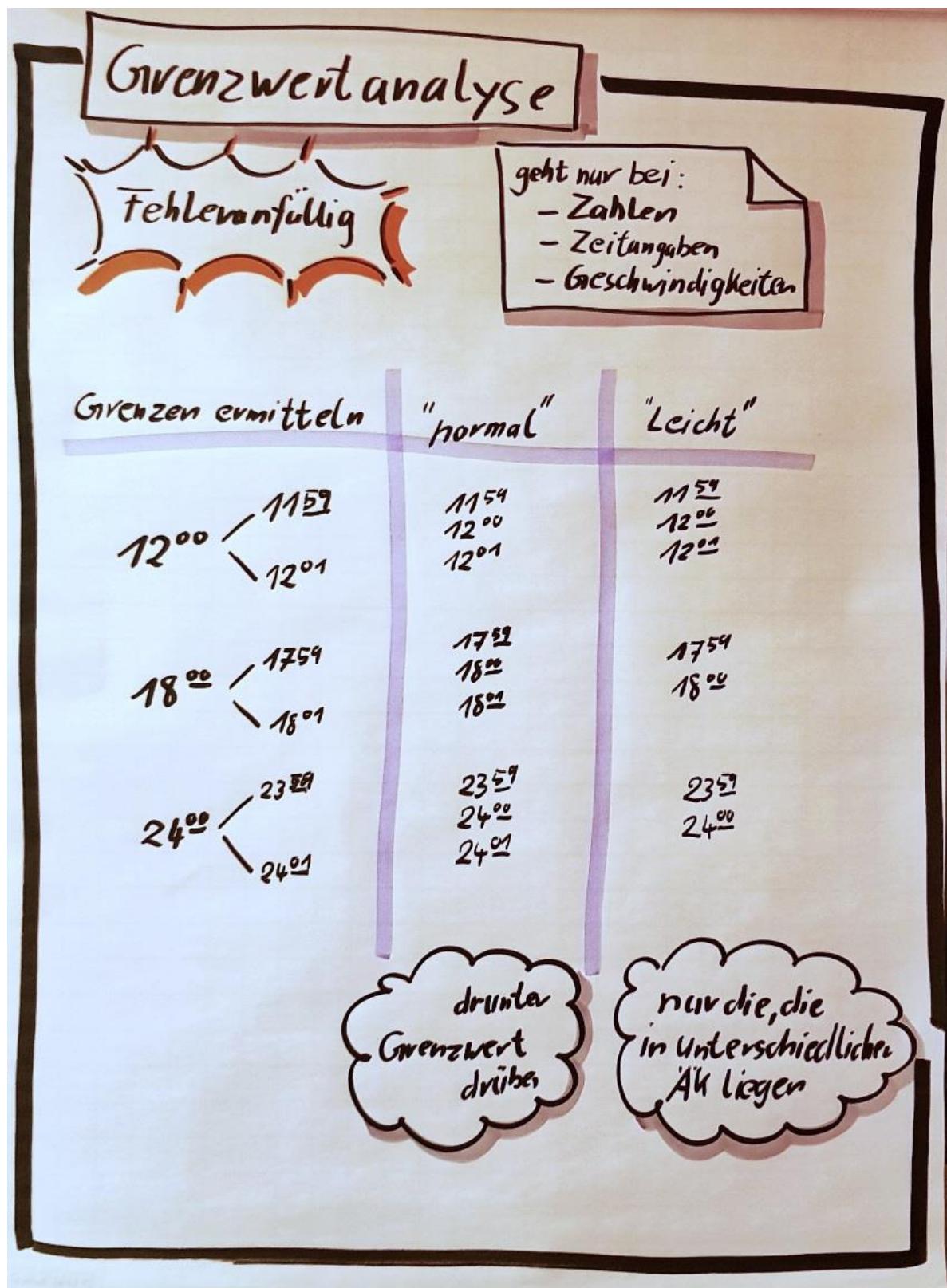
Eingabewerte müssen nicht unbedingt manuell aus Endbenutzer-Sicht in das System eingegeben werden, so dass Äquivalenzklassen z.B. auch für interne Schnittstellenparameter oder Wertbelegungen interner Variablen gebildet werden können. Auch können Äquivalenzklassen weiter in Unterklassen unterteilt werden.

Bei einem System mit mehreren Eingabeparametern sind Äquivalenzklassen für jeden Parameter aufzustellen. Ein Testfall kombiniert dann Werte aus den Äquivalenzklassen aller Eingabeparameter. Falls ungültige Eingabeparameter getestet werden, sollte in einem Testfall immer nur ein einzelner Parameter ungültig sein, um zu verhindern dass Fehlerwirkungen maskiert bleiben, d.h. es muss klar sein, dass dieser eine ungültige Parameter zur Fehlermeldung geführt hat.

Die Überdeckung der Äquivalenzklassenbildung misst die Anzahl der getesteten Äquivalenzklassen im Verhältnis zu allen gebildeten Äquivalenzklassen.

4.2.2 Äquivalenzklassenbildung und Grenzwertanalyse

FL-4.2.2 (K3) Die Grenzwertanalyse anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten



Beispiele	Übungen	Schlüsselbegriffe
	Übung 3: Grenzwertanalysen	Grenzwertanalyse

Erläuterungen

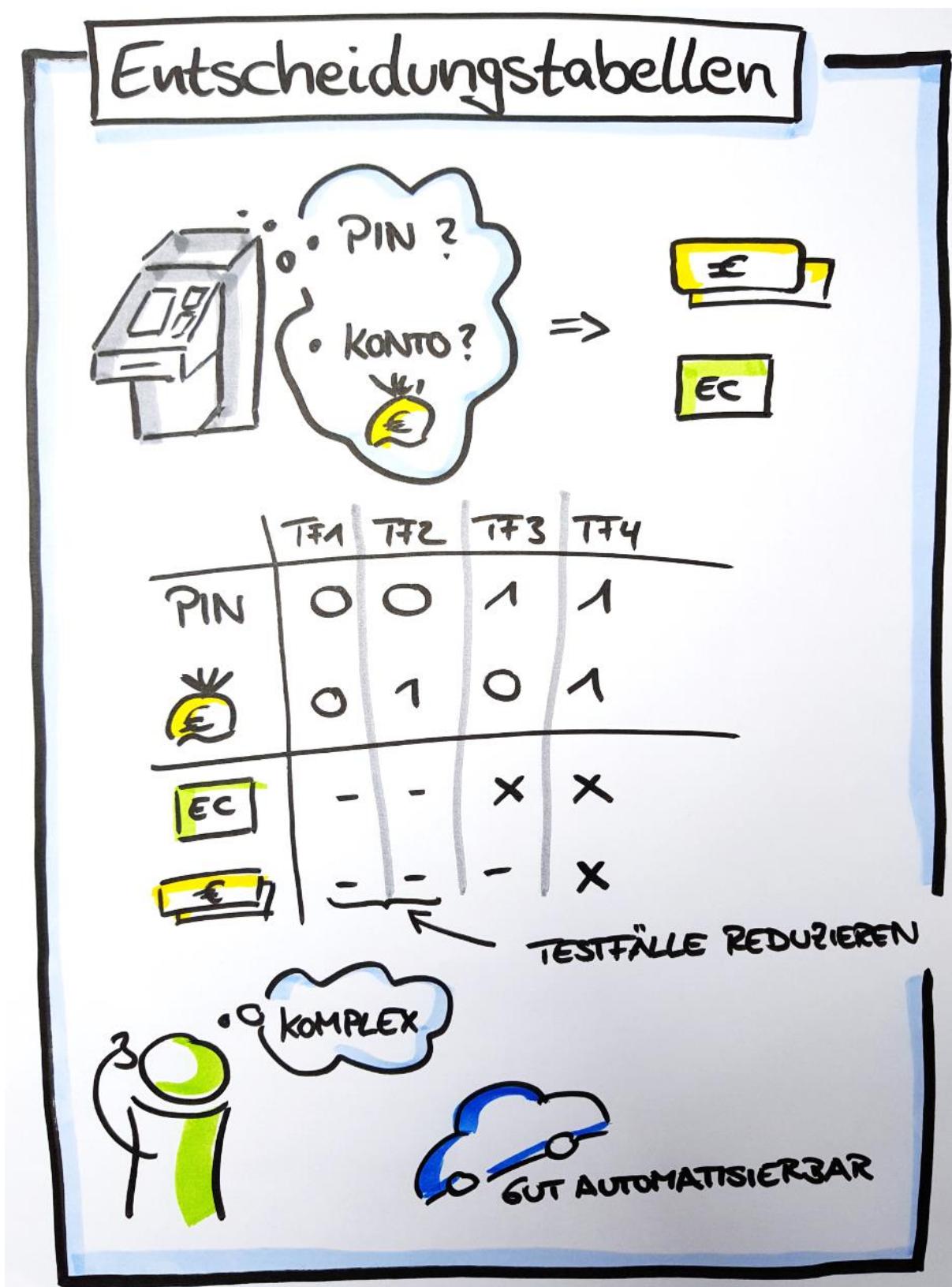
Grenzwertanalysen erweitern die Äquivalenzklassenbildung um eine Betrachtung der Randbereiche der Äquivalenzklassen. Dazu ist es natürlich erforderlich, dass für die Eingabewerte eine Ordnung/Sortierung existiert, wie z.B. bei numerischen Werten, Datums-/Zeitangaben usw.. Ferner lassen sich Grenzwerte auch für nicht-funktionale Tests bilden, z.B. für Performanzwerte.

Gerade diese Randbereiche sind besonders fehleranfällig (Entwickler implementieren z.B. ein $<=$ statt ein $>=$). Je nachdem, wie die Grenzen zwischen Äquivalenzklassen definiert sind, macht es Sinn, zwei oder drei Werte pro Grenze zu testen.

Die Überdeckung der Grenzwertanalyse misst die Anzahl der getesteten Grenzwerte im Verhältnis zu allen gebildeten Grenzwerten.

4.2.3 Entscheidungstabellentests

FL-4.2.3 (K3) Entscheidungstabellentests anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten



Beispiele	Übungen	Schlüsselbegriffe
	Übung 4: Entscheidungstabellentests	Entscheidungstabellentest

Erläuterungen

Eine **Entscheidungstabelle** listet Eingaben (Bedingungen) und Ausgaben (Aktionen) in den Zeilen einer Tabelle auf. Testfälle können dann einfach in den Spalten der Tabelle gebildet werden, indem alle Kombinationsmöglichkeiten der Bedingungen abgedeckt werden, und die eintretenden Aktionen in der entsprechenden Spalte markiert werden.

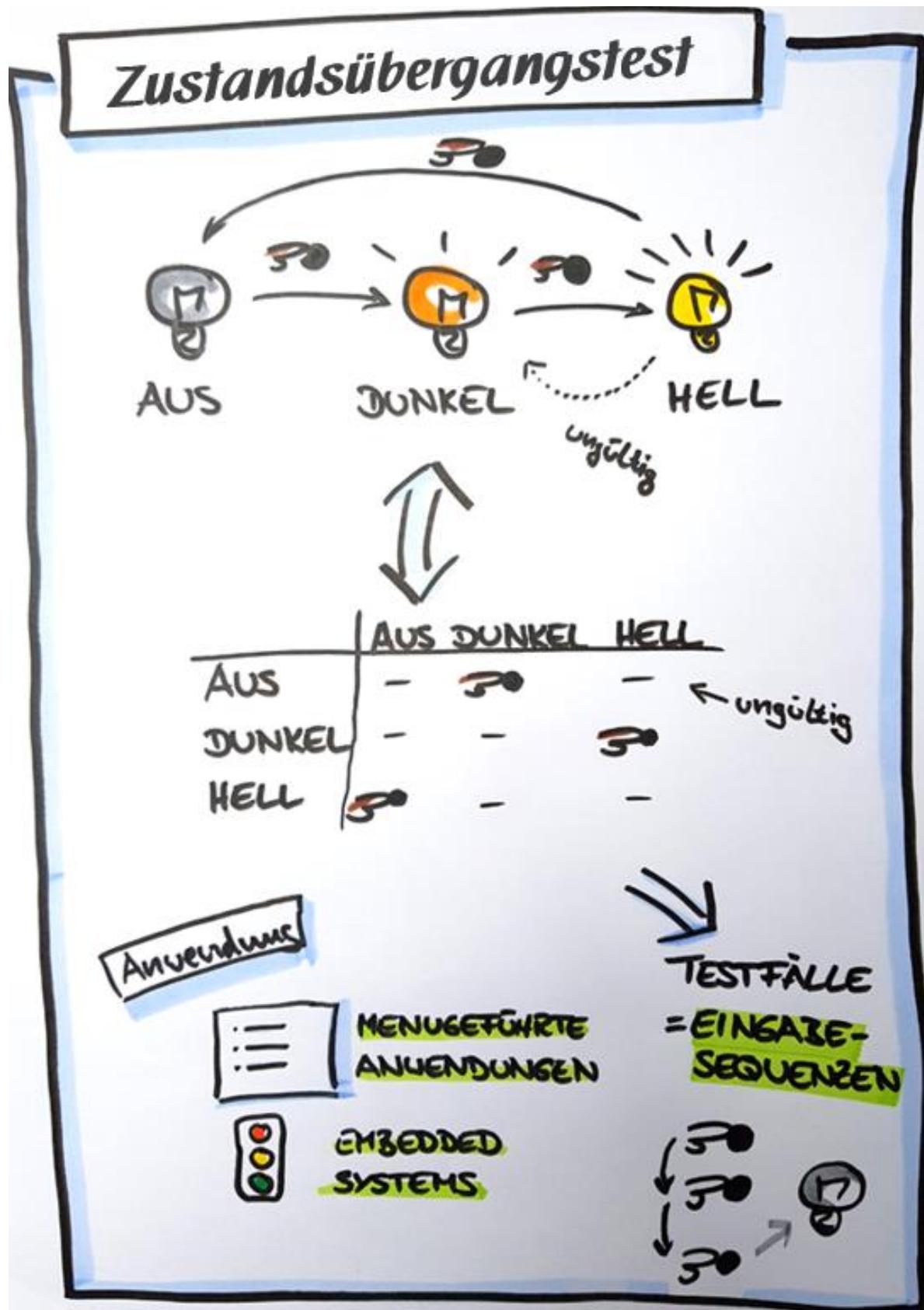
Leicht sieht man, dass die Anzahl der Kombinationsmöglichkeiten mit steigender Anzahl der Eingabeparameter exponentiell steigt. Daher sollten Entscheidungstabellen optimiert werden, indem unmögliche oder praktisch nicht durchführbare Kombinationen gestrichen werden. Auch Testfälle mit Bedingungen, deren Belegung keinen Einfluss auf das Ergebnis hat, können gelöscht werden.

Der Entscheidungstabellentest erscheint zunächst sehr komplex, ist jedoch aufgrund der sehr strukturierten Vorgehensweise einfach erfassbar, und lässt sich daher auch gut automatisieren.

Die Überdeckung bemisst sich als Anzahl der getesteten Kombinationen an allen möglichen.

4.2.4 Zustandsübergangstest

FL-4.2.4 (K3) Zustandsübergangstestanwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten



Beispiele	Übungen	Schlüsselbegriffe
	Übung 5: Zustandsübergangstest	Zustandsübergangstest

Erläuterungen

Bei zustandsbasierten Systemen hängt der Zustand, in dem sich ein solches System zu einem bestimmten Zeitpunkt befindet, vom vorherigen Zustand des Systems und einem Ereignis (Eingabe), ggf. auch noch von zusätzlichen Schutzbedingungen, ab.

Solche Systeme werden durch Zustandsübergangsdiagramme (deutsch: State Charts) dargestellt, aus denen Abfolgen von Zuständen mit den jeweiligen Aktionen, die Zustandsübergänge auslösen, direkt abgelesen werden können. Weiterhin können beim Zustandsübergang auch bestimmte Ereignisse eintreten.

Zustandsübergangstabellen sind eine alternative Darstellung solcher Diagramme.

Testfälle im **Zustandsübergangstest** beschreiben Eingabesequenzen als Abfolge von Zuständen oder Zustandsübergängen. Je nach Komplexität eines Zustandsgraphen können hierbei sehr komplexe Szenarien entstehen.

Zur Überdeckungsmessung lassen sich die Anzahl der getesteten Zustände im Verhältnis zur Anzahl aller Zustände zählen, oder auch die Anzahl der getesteten Zustandsübergänge im Verhältnis zu allen Zustandsübergängen.

Zusätzlich lässt sich die Intensität der Tests weiter steigern, indem Folgen von Zuständen oder Zustandsübergängen betrachtet werden.

4.2.5 Anwendungsfallbasierter Test

FL-4.2.5 (K2) Erklären können, wie man Testfälle aus einem Anwendungsfall ableitet

Kein Flipchart

Beispiele	Übungen	Schlüsselbegriffe
4.2.5: Anwendungsfalltest		Anwendungsfallbasierter Test

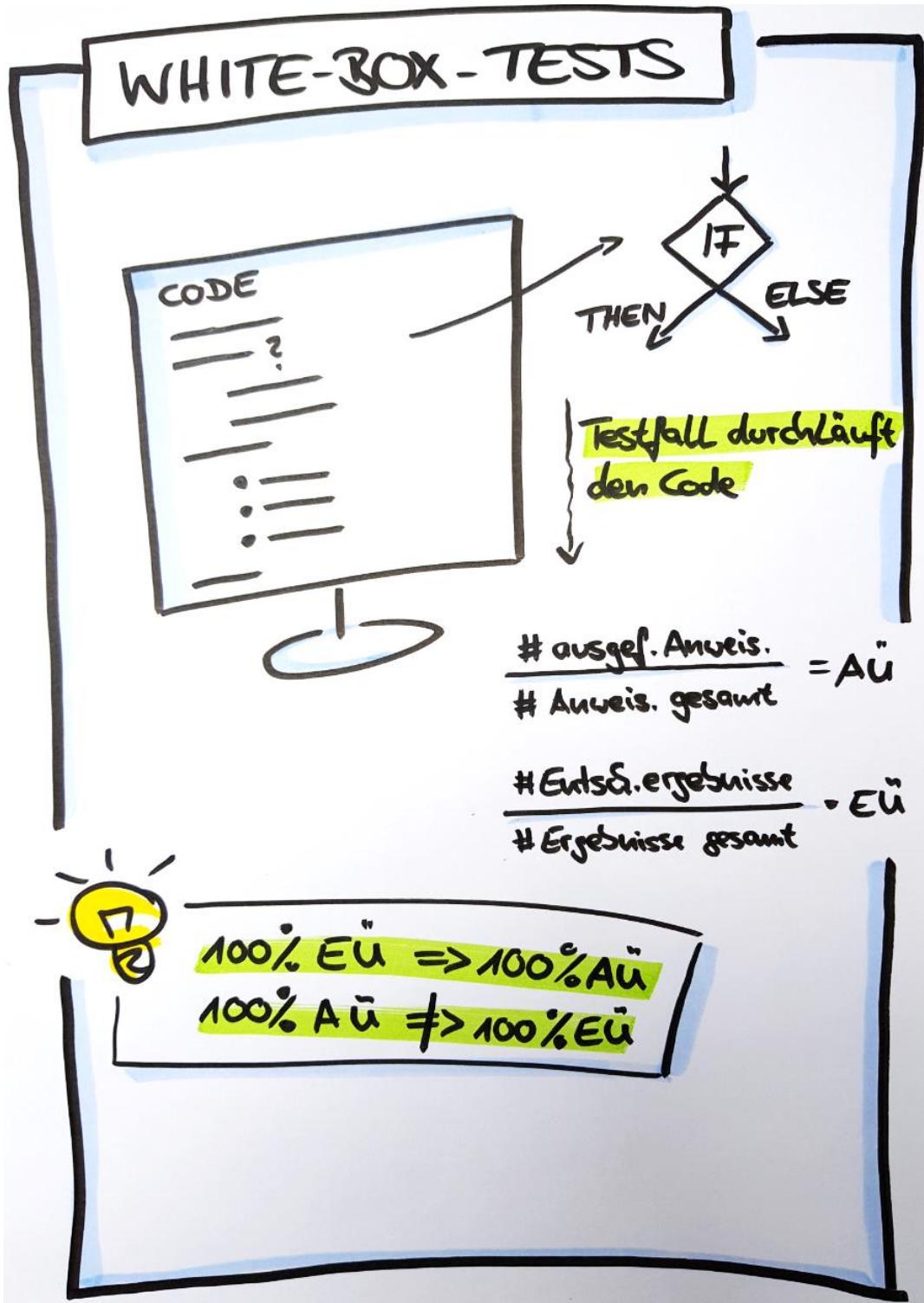
Erläuterungen

Ein Anwendungsfall (Use-Case) beschreibt die Interaktion eines Benutzers (Akteur) mit dem System selbst. Wie auch Zustandsautomaten werden Anwendungsfälle häufig in Form von UML-Diagrammen dargestellt, sollten jedoch zusätzlich auch textuell beschrieben sein. Dadurch werden für jede Variante (jedes Szenario) eines Anwendungsfalls Vorbedingungen, Nachbedingungen, Fehlerbehandlungen, alternative Szenarien usw. beschrieben.

Testfälle lassen sich aus Anwendungsfallbeschreibungen 1:1 ablesen, wobei die Überdeckungsmessung im **Anwendungsfallbasierten Test** alle Varianten eines Anwendungsfalls berücksichtigen muss.

4.3 White-Box-Testverfahren

- FL-4.3.1 (K2) Anweisungsüberdeckung erklären können
- FL-4.3.2 (K2) Entscheidungsüberdeckung erklären können
- FL-4.3.3 (K2) Die Bedeutung von Anweisungs- und Entscheidungsüberdeckung erklären können



Beispiele	Übungen	Schlüsselbegriffe
4.3.1: Anweisungstest 4.3.2: Entscheidungstest 4.3.3: Anweisungs- und Entscheidungsüberdeckung		Anweisungsüberdeckung, Entscheidungsüberdeckung

Erläuterungen

Interne Abläufe in Systemen werden häufig an Hand von Kontrollflüssen beschrieben. Die Anweisungen werden dabei sequentiell durchlaufen, und ein Durchlauf durch den Kontrollfluss bildet einen Testfall.

Ziel der Tests bei der **Anweisungstests** ist es nun, möglichst jede Anweisung im Code einmal auszuführen. Dadurch lässt sich neben der Prüfung auf Korrektheit von Berechnungen z.B. auch nicht erreichbarer Code („dead code“) identifizieren.

Spezielle Anweisungen im Code führen zu Verzweigungen oder Schleifen, die wir Entscheidungen nennen. Code innerhalb einer Schleife oder Code hinter einer IF-Anweisung wird in Abhängigkeit der Entscheidung ausgeführt.

Beim **Entscheidungstest** sollte jeder Ausgang der Entscheidung einmal getestet werden. Daher messen wir die Überdeckung als Anzahl der getesteten Entscheidungsergebnisse im Verhältnis zu allen möglichen Entscheidungsergebnissen.

100% **Entscheidungsüberdeckung** beinhaltet 100% **Anweisungsüberdeckung**.

4.4 Erfahrungsbasierte Testverfahren

FL-4.4.1 (K2) Die intuitive Testfallermittlung erklären können

FL-4.4.2 (K2) Exploratives Testen erklären können

FL-4.4.3 (K2) Checklistenbasiertes Testen erklären können

Kein Flipchart,

Beispiele	Übungen	Schlüsselbegriffe
4.4.1: Intuitiv Testen 4.4.2: Explorativ Testen 4.4.3: Mit Checklisten Testen Sogeti-Template für eine Test Charta		checklistenbasiertes Testen, erfahrungsbasierte Testverfahren, exploratives Testen, intuitive Testfallermittlung

Erläuterungen

Erfahrungsbasierte Verfahren ergänzen die Black-Box- und White-Box-Testverfahren, und ermöglichen so das Finden von Fehlern, die durch systematischere Verfahren nicht aufgedeckt worden wären.

Das Vorgehen erfolgt auch bei den erfahrungsbasierten Verfahren nach einer gewissen Methodik, ist also nicht etwa ein zufälliges herumklicken in der Anwendung.

Die **Intuitive Testfallermittlung** basiert darauf, „*Fehlhandlungen, Fehlerzustände und Fehlerwirkungen aufgrund des Wissens des Testers*“ zu vermuten, u.a. [Zitat Lehrplan Abschnitt 4.4.1]

- Wie hat die Anwendung früher funktioniert?
- Welche Fehlhandlungen machen die Entwickler üblicherweise?
- Fehlerwirkungen, die in anderen Anwendungen aufgetreten sind

Auf dieser Grundlage erstellt der Tester beim Testentwurf eine Checkliste möglicher Fehler, und leitet Testfälle her, die zielgerichtet solche vermuteten Fehler aufdecken können.

Explorative Tests werden während der Testdurchführung entworfen, indem Testergebnisse genutzt werden, um die Erfahrungen wieder zu verwenden, z.B. wenn sich Fehler in einem bestimmten Teil des Systems häufen. Die methodische Grundlage für eine explorative Testsitzung ist eine Test-Charta oder ein Sitzungsblatt (Session-Sheet), womit die durchgeführten Tests sowie die Ergebnisse protokolliert werden. Wichtig ist auch, eine Testsitzung auf ein fest definiertes Zeitfenster zu beschränken, in der Regel 1-2 Stunden.

Beim **Checklistenbasierten Testen** werden die Testbedingungen zunächst auf einer Checkliste erfasst, und entweder direkt zum Testen verwendet, oder aber im Testentwurf in eine neue Checkliste überführt. Die Checklisten entstehen dabei nicht formal anhand bestimmter Black-Box oder White-Box Techniken, sondern auf „*Grundlage von Erfahrungen, Wissen über das Wesentliche in der Nutzung oder einem guten Verständnis darüber, warum und wie die Software fehlschlägt*“.[Zitat Lehrplan, Abschnitt 4.4.3]

5. Testmanagement

Flipchart-Bilder

- Unabhängigkeit
- Eingangs-/Endekriterien
- Testaufwand
- Metriken und Berichte
- Produkt- und Projektrisiken
- Risikobasiertes Testen
- Fehlerberichte

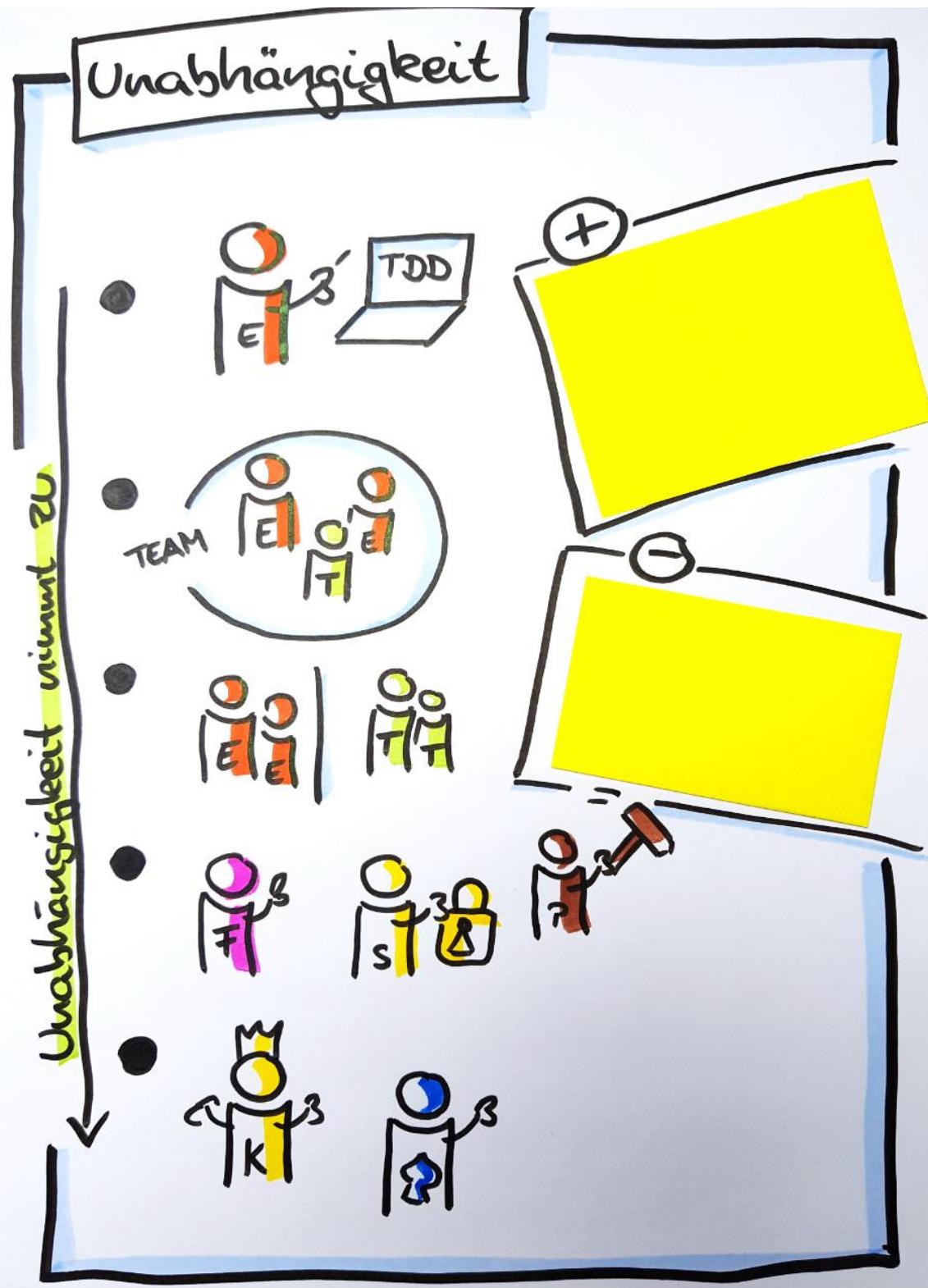
Schlüsselbegriffe im Kapitel

Eingangskriterien, Endekriterien, Fehlerbericht, Fehlermanagement, Konfigurationsmanagement, Produktrisiko, Projektrisiko, Risiko, risikobasiertes Testen, Risikostufe, Testabschlussbericht, Tester, Testfortschrittsbericht, Testkonzept, Testmanager, Testplanung, Testschätzung, Teststeuerung, Teststrategie, Testüberwachung, Testvorgehensweise

5.1 Testorganisation

FL-5.1.1 (K2) Vor- und Nachteile unabhängigen Testens erklären können

FL-5.1.2 (K1) Die Aufgaben eines Testmanagers und eines Testers benennen können



Beispiele	Übungen	Schlüsselbegriffe
5.1.1: Unabhängiges Testen		Tester, Testmanager

Erläuterungen

Unter dem Begriff „Unabhängigkeit des Testens“ werden verschiedene Organisationsformen des Testens in einem Unternehmen oder einem Projekt zusammengefasst. Wir kategorisieren fünf unterschiedliche Stufen der Unabhängigkeit, wobei durchaus auch Mischformen oder mehrere Formen der Testorganisation gleichzeitig vorkommen können.

Es empfiehlt sich, die Unabhängigkeit umso größer zu wählen, je näher man dem Produktivbetrieb einer Software kommt – bei Komponententests auf Entwicklerseite ist es dagegen durchaus üblich, die niedrigste Form der Unabhängigkeit zu wählen.

In einem sequentiellen Entwicklungsprozess genügt es dazu, mit jeder Teststufe die Stufe der Unabhängigkeit zu erhöhen. In iterativen Modellen dagegen werden häufiger interdisziplinäre Teams gebildet, aus Entwicklern, Testern und Fachexperten, die während der gesamten Entwicklungszeit zusammen arbeiten.

- ➔ Vorteile und Nachteile des unabhängigen Testens wurden in der Schulung gesammelt, und können hier nachgetragen werden.

Wir unterscheiden hier nur die Rollen des **Testmanagers** und des **Testers**. Allerdings ist es wichtig zu wissen, dass in der Praxis auch detailliertere Rollen existieren, die bestimmte Aktivitäten übernehmen oder bestimmte Fähigkeiten benötigen, z.B. Testdatenmanager, Testautomatisierer, Sicherheitstester u.a.

- ➔ Die typischen Aufgaben eines Testmanagers und Testers wurden an Hand von Moderationskarten besprochen, und können im Bild zu Kapitel 1.4 nachgetragen werden.

5.2 Testplanung und -schätzung

-
- FL-5.2.1 (K2) Den Zweck und Inhalt eines Testkonzepts zusammenfassen können
 FL-5.2.2 (K2) Zwischen verschiedenen Teststrategien unterscheiden können
 FL-5.2.4 (K3) Wissen über Priorisierung sowie technische und logische Abhängigkeiten anwenden können, um die Testdurchführung für ein gegebenes Testfallset zu planen
-

Kein Flipchart

Beispiele	Übungen	Schlüsselbegriffe
5.2.1: Testkonzept 5.2.2: Teststrategien	Übung 6: Testausführungsplan	Testkonzept, Testplanung, Testschätzung, Teststeuerung, Teststrategie, Testvorgehensweise

Erläuterungen

Die Inhalte eines **Testkonzepts** werden z.B. in der ISO-Norm 29113-3 behandelt. In der Praxis sollte jedoch jedes Unternehmen sein eigenes Template für Testkonzepte entwickeln und für die Projekte bereitstellen. Die Inhalte wurden bereits in Abschnitt 1.4 erläutert, und im dortigen Bild dargestellt.

Das Testkonzept als Dokument fasst die Ergebnisse der **Testplanung** zusammen und führt die Testplanung kontinuierlich durch den gesamten Produktlebenszyklus fort, d.h. das Testkonzept ist ein lebendes, fortlaufend zu aktualisierendes Dokument.

Wesentlicher Teil des Testkonzeptes ist die Festlegung der **Teststrategie** bzw. der **Testvorgehensweise**. Diese legt dar, wie auf Grundlage der Testbasis sowie der Rahmenbedingungen für das Projekt die Testfälle zu erstellen, zu priorisieren und durchzuführen sind.

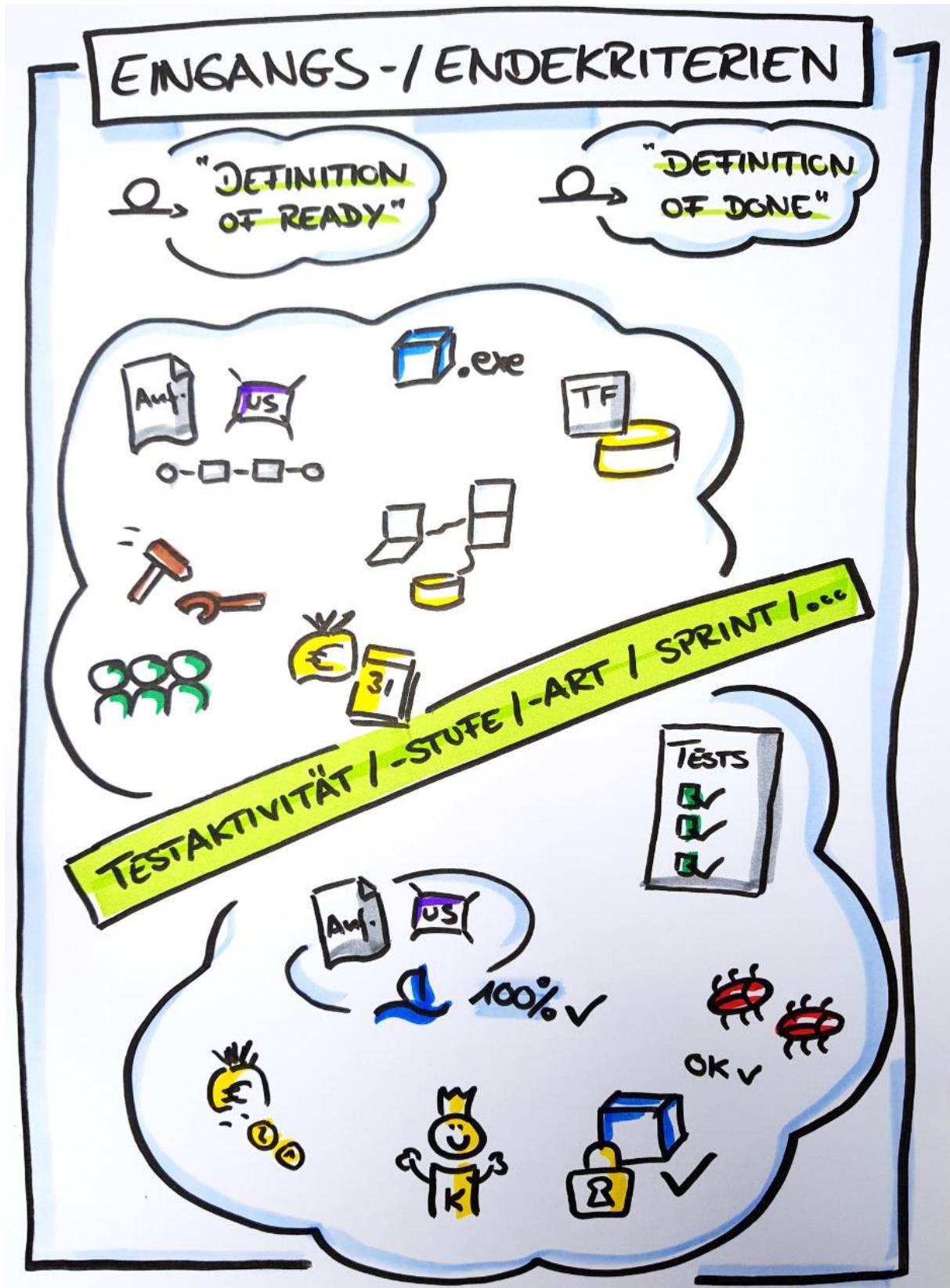
Die untenstehende Tabelle gibt einen Überblick über die im Lehrplan, Abschnitt 5.2.2 genannten Teststrategien. Ein Projekt sollte sich nicht unbedingt auf eine Strategie festlegen, sondern eine Kombination der Strategien wählen. Eine Zuordnung einer gegebenen Strategie in eine der genannten Kategorien ist somit in der Regel auch nicht eindeutig.

Nach der Erstellung von Testfällen ist festzulegen, wann welche Testfälle zur Ausführung kommen sollen. In dem dabei zu erstellenden **Testausführungsplan** sind die Testfälle entsprechend ihrer Priorität, ihrer Abhängigkeiten untereinander sowie vom Testobjekt an sich, sowie in Abhängigkeit von Planungsaspekten anzugeben, die Ausführungszeiten sind festzulegen und Tester zuzuweisen.

Die Ausführungsreihenfolge der Testfälle kann zum einen durch die Teststrategie bestimmt werden, zum anderen aber auch die Effizienz und Effektivität einer gewählten Teststrategie beeinflussen.

Teststrategie	Grundlage	Beispiel
Analytisch	Analyse eines Faktors, z.B. Anforderung oder Risiko	Risikobasiertes Testen
Modellbasiert	Modell eines geforderten Produktaspekts, einer Funktion, eines Geschäftsprozesses, einer internen Struktur oder nichtfunktionalen Eigenschaft	Geschäftsprozessmodell Zustandsmodell Zuverlässigkeitswachstumsmodell
Methodisch	Systematische Nutzung vordefinierter Sets von Tests oder Testbedingungen	Look-and-Feel-Standards
Prozesskonform	Analyse, Entwurf und Realisierung von Tests auf der Grundlage von externen Vorschriften und Standards, Branchenstandards, Prozessdokumentation	Prozesse und Standards, die für das oder vom Unternehmen aufgestellt worden sind
Angeleitet / beratend	Beratung, Anleitung, Anweisungen von Stakeholdern, Fachexperten	Technologieexperten kommen von außerhalb des Unternehmens
Regressionsvermeidend	Wunsch, Rückgang bei den vorhandenen Leistungsfähigkeiten zu vermeiden	Wiederverwendung vorhandener Testmittel Automatisierung von Regressionstests Nutzung von Standardtestsuiten
Reaktiv	Reaktion auf die während der Testdurchführung auftretenden Ereignisse	Exploratives Testen

FL-5.2.3 (K2) Beispiele für mögliche Eingangs- und Endekriterien geben können



Beispiele	Übungen	Schlüsselbegriffe
5.2.3: Eingangs- und Endekriterien		Eingangskriterien, Endekriterien

Erläuterungen

Kriterien, wann eine Testaktivität beginnen sollte, und wann sie abgeschlossen ist, sollten bereits während der Testplanung definiert werden. Die Sicherstellung der Kriterien, bzw. die Überprüfung vor und nach einer Aktivität stellen die Verlässlichkeit der Aktivitäten sicher oder weisen bei Nichterfüllen darauf hin, dass „*sich die Aktivität als schwieriger, zeitaufwändiger, kostspieliger und risikoreicher erweisen wird*“ [Zitat Lehrplan Kapitel 5.2.3].

- ➔ Die in der Schulung erläuterten Kriterien können auf dem Flipchart entsprechend ergänzt werden.

Praktische Hinweise

- Frühzeitige Planung der Testumgebung ist entscheidend, um das Eingangskriterium der Testbereitschaft auch rechtzeitig zur Testdurchführung erfüllen zu können.
- Nichterfüllte Eingangskriterien sind vermeidbare Projektrisiken, vor allem wenn es um die Testbarkeit einer Anforderung, eines Entwurfs oder einer User Story geht, denn diese Aspekte sollten im Review lange vor Beginn der Entwicklung geklärt werden.

FL-5.2.5 (K1) Faktoren benennen können, die den Testaufwand beeinflussen

FL-5.2.6 (K2) Den Unterschied zwischen zwei Schätzverfahren erklären können: das metrikbasierte Verfahren und das expertenbasierte Verfahren



Beispiele	Übungen	Schlüsselbegriffe
5.2.5: Metriken- oder Expertenbasiert?		

Erläuterungen

Eine wichtige Aktivität im Zusammenhang mit der Planung ist die Testaufwandsschätzung. Dabei sind alle Aktivitäten, im Zusammenhang mit dem Testen, sowie Faktoren, die den Testaufwand beeinflussen, zu ermitteln und zu bemessen. Die Einflüsse werden auf dem Flipchart, sowie im Lehrplan, Abschnitt 5.2.5, auch in Form von Stichpunkten dargelegt.

Um den Aufwand zu bestimmen, unterscheidet der Lehrplan in Abschnitt 5.2.6 zwei Verfahren:

- Das metrikbasierte Verfahren: Schätzung des Testaufwands auf Basis der Metriken früherer ähnlicher Projekte oder auf Basis von typischen Werten
- Das expertenbasierte Verfahren: Schätzung des Testaufwands basierend auf der Erfahrung der für die Testaufgaben zuständigen Person oder von Experten

Auch wenn agile Projekte scheinbar eher auf die vermeintlich leichtgewichtigeren, expertenbasierten Verfahren setzen, z.B. den bekannten Planungspoker, kommen auch hier durchaus formalere metrikenbasierte Vorgehen zum Einsatz, etwa das Messen der Velocity, die Verwendung von Burndown-Charts und die Vergabe von Story Points für User Stories.

Praktische Hinweise

- Um den zu erwartenden Zeitbedarf für ein Arbeitspaket herauszufinden, bedarf es einer geschickten Kommunikationsstrategie, denn es kommt darauf an, wen man um eine Schätzung bittet (erfahrene Tester, Fachseite, Entwickler), aber auch wann man fragt (kurz vor Feierabend, früh morgens, in Stresssituationen, nach negativen Erfahrungen).

5.3 Testüberwachung und -steuerung

FL-5.3.1 (K1) Testmetriken wiedergeben können

FL-5.3.2 (K2) Zweck, Inhalte und Zielgruppen für Testberichte zusammenfassen können



Beispiele	Übungen	Schlüsselbegriffe
5.3.2: Testberichte		Testabschlussbericht, Testfortschrittsbericht, Testüberwachung

Erläuterungen

Zur **Überwachung** und Steuerung der Testaktivitäten ist es unerlässlich, bestimmte Metriken zu erheben. Die gesammelten Daten können dann ausgewertet werden, um Maßnahmen einzuleiten, wenn z.B. von der Planung abgewichen wird, oder sich die Voraussetzung für den Test ändern.

Beispiele für Metriken lassen sich in die vier Kategorien Fortschrittsmetriken (bezogen auf Zeitplan und Budget), Produktmetriken (Qualität des Testobjekts), Testmetriken (Angemessenheit der Testvorgehensweise) und Effektivität der Testaktivitäten in Bezug auf die Testziele unterteilen.

Zwar lässt sich nur das sinnvoll kontrollieren, was auch gemessen wird. Anders herum sollte im Verlauf eines Projektes aber auch nur das gemessen werden, was auch benötigt wird. Die erhobenen Daten werden dann aufbereitet in Form von Statistiken, Tabellen, Diagrammen, und in Testberichten dargestellt.

Ein **Testfortschrittsbericht**, der während einer Testaktivität erstellt wird, liefert laufend aktuelle Informationen zum Verlauf des Projekts. Ein **Testabschlussbericht** fasst die Aktivitäten und den Status zum Ende einer Aktivität wie auch am Ende des Projekts zusammen, und bewertet auch das Erreichen der Endekriterien.

- Die Inhalte eines Fortschrittsberichts und Abschlussberichts gemäß Lehrplan, Abschnitt 5.3.2, können hier nachgetragen werden.

Praktische Hinweise

- Art und Aufbereitung haben Einfluss auf den praktischen Nutzen
- Automatisierte Berichterstattung kann den Zeitraum deutlich verringern.

5.4 Konfigurationsmanagement

FL-5.4.1 (K2) Zusammenfassen können, wie Konfigurationsmanagement das Testen unterstützt

Kein Flipchart

Beispiele	Übungen	Schlüsselbegriffe
5.4.1: Konfigurationsmanagemen t		Konfigurationsmanagement

Erläuterungen

Konfigurationsmanagement dient der Versionierung von Testmitteln und zur Verwaltung der Beziehungen von Artefakten untereinander. Häufig verwenden Entwickler bereits ein Konfigurationsmanagement-Werkzeug, so dass es Sinn macht, während der Testplanung eine Mitbenutzung durch die Tester zu planen.

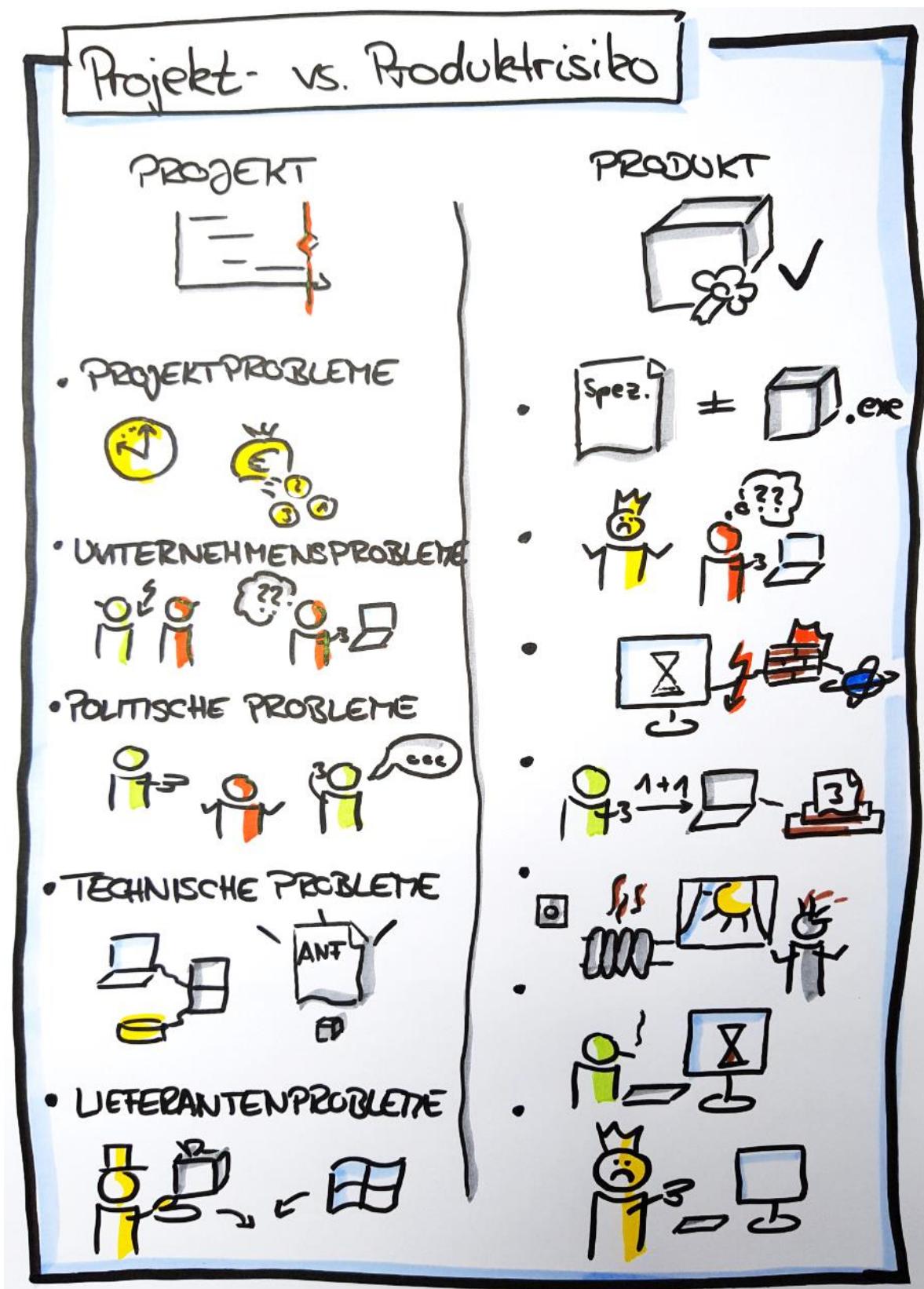
Das Konfigurationsmanagement muss folgendes sicherstellen [Zitat Lehrplan Kapitel 5.4]:

- Alle Testelemente sind eindeutig identifiziert, versionskontrolliert, werden in ihren Änderungen nachverfolgt und stehen in Verbindung zueinander.
- Alle Elemente der Testmittel sind eindeutig identifiziert, versionskontrolliert, werden in ihren Änderungen nachverfolgt, stehen in Verbindung zueinander und in Verbindung zu den Versionen der Testelemente, so dass die Verfolgbarkeit im Testprozess gewährleistet werden kann.
- Alle identifizierten Dokumente und Softwareelemente sind unmissverständlich in der Testdokumentation benannt.

Insbesondere die Wartbarkeit der Testmittel wird durch ein Konfigurationsmanagement unterstützt. Wartbare Testfälle zu erstellen ist besonders im Hinblick auf die Erhaltung der Verfolgbarkeit bei Änderungen von Anforderungen oder der Behebung von Fehlern wichtig, aber .B. auch bei Weiterentwicklung bestehender Testsuiten in der iterativen Softwareentwicklung sowie bei der Übertragung manueller Testfälle in automatisierte Testskripte.

5.5 Risiken und Testen

FL-5.5.2 (K2) Zwischen Projekt- und Produktrisiken unterscheiden können



Beispiele	Übungen	Schlüsselbegriffe
5.5.2: Projekt- oder Produktrisiko?		Produktrisiko, Projektrisiko

Erläuterungen

Ein Risiko ist ein möglicherweise vorkommendes Ereignis in der Zukunft, das negative Auswirkungen hat. Die Höhe des Risikos wird durch die Eintrittswahrscheinlichkeit des Ereignisses und dessen Wirkung (der Schadenhöhe) bestimmt [Zitat Lehrplan, Abschnitt 5.5.1].

Wir unterscheiden zwischen Projekt- und Produktrisiken.

Projektrisiko ist ein Risiko, das den Projekterfolg beeinträchtigt. Ein Produktrisiko ist ein Risiko, das die Qualität eines Produkts beeinträchtigt. [Zitat Glossar, Version 3.2]

Projektrisiken führen dazu, dass das Projekt in Gefahr gerät, in der Regel dadurch, dass es nicht mehr in der Lage ist, das Produkt oder Teile davon im geplanten Zeit- und Kostenrahmen liefern zu können. Ist das Projekt beendet, sind die Projektrisiken nicht mehr relevant.

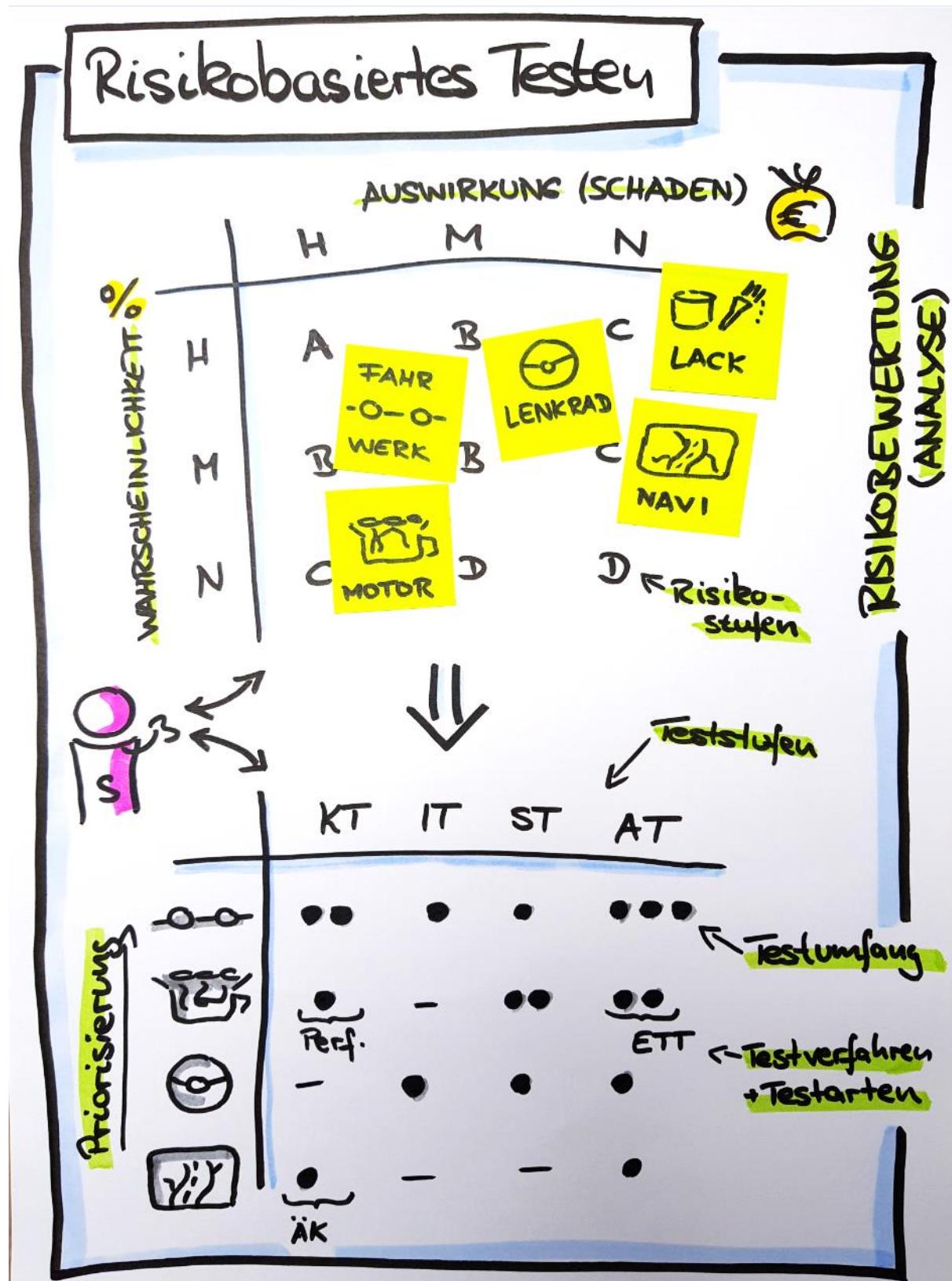
Produktrisiken bleiben auch nach Projektende im Produkt bestehen, denn sie sind eng mit dem Produkt selbst und seiner Gebrauchstauglichkeit verbunden. Insofern besteht zwischen den Produktrisiken und der Produktqualität ein unmittelbarer Zusammenhang.

Ergänzend zum Flipchart listet der Lehrplan, Abschnitt 5.5.2, die dargestellten Punkte in Textform.

Praktische Hinweise

- Zwischen dem Risikobasierten Testen (siehe nächstes Kapitel) und den Produktrisiken besteht ein unmittelbarer Zusammenhang, denn ein strukturierter, sorgfältig ausgeführter Risikomanagement-Prozess erleichtert das Verständnis für Risiken.
- Ausreichende Kommunikation über die ermittelten Risiken fördert zudem den Projekterfolg, insbesondere das Schaffen von Bewusstsein für Risiken bei den Stakeholdern eines Projekts.

- FL-5.5.1 (K1) Risikostufe anhand der Wahrscheinlichkeit (des Eintritts) und Auswirkung (im Schadensfall) definieren können
- FL-5.5.3 (K2) Anhand von Beispielen beschreiben können, wie die Produktrisikoanalyse Intensität und Umfang des Testens beeinflussen kann



Beispiele	Übungen	Schlüsselbegriffe
5.5.3: Produktrisikoanalyse		Risiko, risikobasiertes Testen, Risikostufe

Erläuterungen

Beim **risikobasierten Testen** werden die Testverfahren, Testarten, der Testumfang (Testintensität) und die Priorisierung der Testfälle entsprechend der ermittelten Risiken bestimmt. Das Ziel dieser Vorgehensweise ist es, die Produktqualität zu erhöhen durch Aufdecken von Fehlern in den Bereichen, wo die größten Produktrisiken zu erwarten sind.

Ein guter Ansatz hierfür ist z.B. ein Herunterbrechen des Produkts in kleinere Bestandteile (oder des Testobjekts in Testelemente), für die dann eine Risikoidentifizierung durchgeführt wird (welcher Schaden kann mit welcher Wahrscheinlichkeit entstehen?)

Im Rahmen einer Risikobewertung kann dann für jedes Teilprodukt (Testelement) eine **Risikostufe** vergeben werden.

Die darauf folgenden Maßnahmen zur Risikominderung sind im einfachsten Fall die Priorisierung der Testfälle, so dass am höchsten priorisierte Funktionalität zuerst getestet wird.

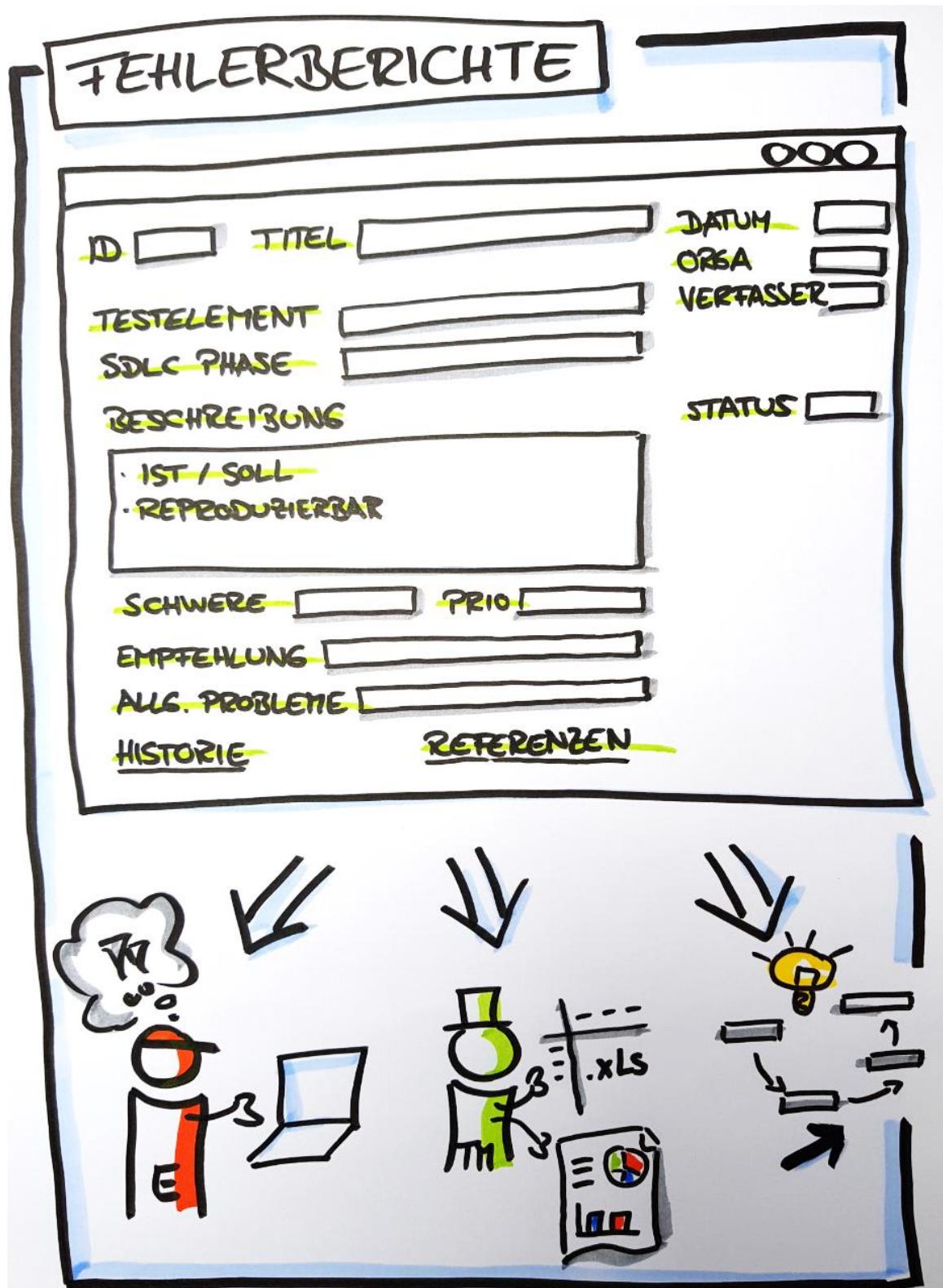
Im Rahmen von Testplanung, -analyse und -entwurf lässt sich die Risikoanalyse jedoch auch dazu verwenden, bei einer risikobasierten Testvorgehensweise Testintensitäten über die Teststufen und Testarten zu verteilen; außerdem können Testverfahren dem Risiko entsprechend ausgewählt, und Überdeckungsmaße definiert werden.

Praktische Hinweise

- Fehleranalysen aus vorherigen oder vergleichbaren Projekten können helfen, eine bessere Einschätzung von Risiken vornehmen zu können.

5.6 Fehlermanagement

FL-5.6.1 (K3) Einen Fehlerbericht schreiben können, der einen während des Testens gefundene Fehler enthält



Beispiele	Übungen	Schlüsselbegriffe
	Übung 7: Fehlerbericht schreiben	Fehlermanagement

Erläuterungen

Zur Behandlung der im Test identifizierten Fehlerwirkungen und Fehlerzustände wird ein **Fehlermanagement**prozess definiert. Der Lebenszyklus eines Fehlers beginnt mit dessen Erstellung in Form eines Fehlerberichts, wie etwa in ISO 29119-3 oder auch im Lehrplan, Abschnitt 5.6, definiert.

Fehler können nach bestimmten Kriterien kategorisiert werden, z.B. auch an Hand von Standards wie der IEEE-1044. Die Fehlertaxonomie ist „*eine systematische Liste von Fehlerarten mit ihrer hierarchischen Gliederung in Fehlerkategorien. Sie dient der Klassifikation von Fehlerzuständen*“ [Zitat Glossar Version 3.2, CTAL-TM].

Vorteile der systematischen Erfassung von Fehlerberichten sind die Möglichkeiten der Auswertung im Hinblick auf die Bewertung der Produktqualität, aber auch im Hinblick darauf, die Testqualität zu verbessern durch Analyse der falsch positiven (Fehler, die keine waren), oder falsch negativen Ergebnisse (Fehler, die man nicht bzw. zu spät gefunden hat).

Die Ziele von Fehlerberichten sind im Lehrplan, Abschnitt 5.6, wie folgt dargestellt:

- *Entwickeln und anderen Beteiligten Informationen über jedes aufgetretene unerwünschte Ereignis zur Verfügung zu stellen, um sie in die Lage zu versetzen, spezifische Auswirkungen zu identifizieren, das Problem mit minimalem Test zu reproduzieren und zu isolieren und die möglichen Fehlerzustände wie erforderlich zu korrigieren oder anderweitig das Problem zu lösen*
- *Testmanagern eine Möglichkeit zu geben, die Qualität der Arbeitsergebnisse und die Auswirkungen auf das Testen nachzuverfolgen (z.B. falls eine große Anzahl von Fehlerzuständen gemeldet wird, haben die Tester mehr Zeit damit verbracht, die Fehlerzustände zu melden, statt Tests durchzuführen, und es werden mehr Fehlernachtests notwendig sein)*
- *Ideen liefern für die Verbesserung der Entwicklung und des Testprozesses*

6. Werkzeugunterstützung für das Testen

Flipchart-Bilder

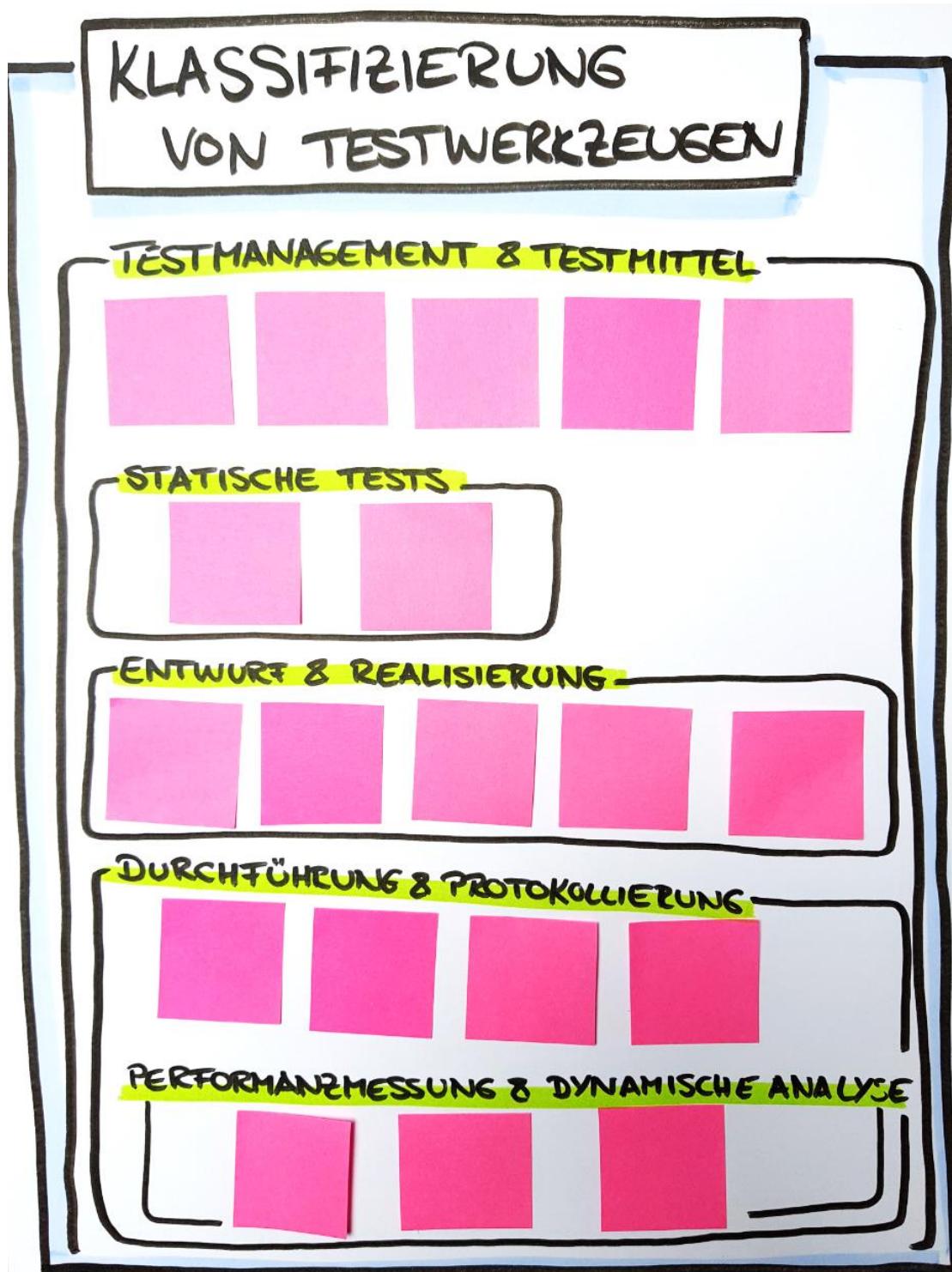
- Werkzeugtypen
- Testautomatisierung
- Werkzeugeinführung

Schlüsselbegriffe im Kapitel

Datengetriebener Test, schlüsselwortgetriebener Test, Testautomatisierung, Testausführungswerkzeug, Testmanagementwerkzeug

6.1 Überlegungen zu Testwerkzeugen

- FL-6.1.1 (K2) Testwerkzeuge gemäß ihrem Zweck und den Testaktivitäten, die sie unterstützen, klassifizieren können
- FL-6.1.2 (K1) Nutzen und Risiken der Testautomatisierung identifizieren können
- FL-6.1.3 (K1) Sich an besondere Gesichtspunkte von Testdurchführungs- und Testmanagementwerkzeugen erinnern können



Beispiele	Übungen	Schlüsselbegriffe
6.1.1: Werkzeugtypen		Testausführungswerkzeug, Testmanagementwerkzeug

Erläuterungen

Werkzeugunterstützung ist in jeder Projektaktivität notwendig zur Verwaltung der anfallenden Informationen und zur Dokumentierung der Arbeitsergebnisse.

Testwerkzeuge werden gemäß Lehrplan, Abschnitt 6.1.1, in sechs Gruppen kategorisiert, die die Testaktivität festlegen, in der ein Werkzeug verwendet wird.

Eine besondere Rolle nehmen dabei die **Testausführungswerkzeuge** ein, die mehrere Disziplinen unterstützen müssen, etwa unterschiedliche Testarten, sowie auch die Testautomatisierung.

Testmanagementwerkzeuge sind häufig universeller gestaltet und erlauben die Integration aller anderen Werkzeuge zur Aufbereitung von Informationen, Sicherstellung der Verfolgbarkeit und Verbindung zwischen angrenzenden Disziplinen wie dem Konfigurationsmanagement, dem Release- und Änderungsmanagement, dem Projektmanagement.

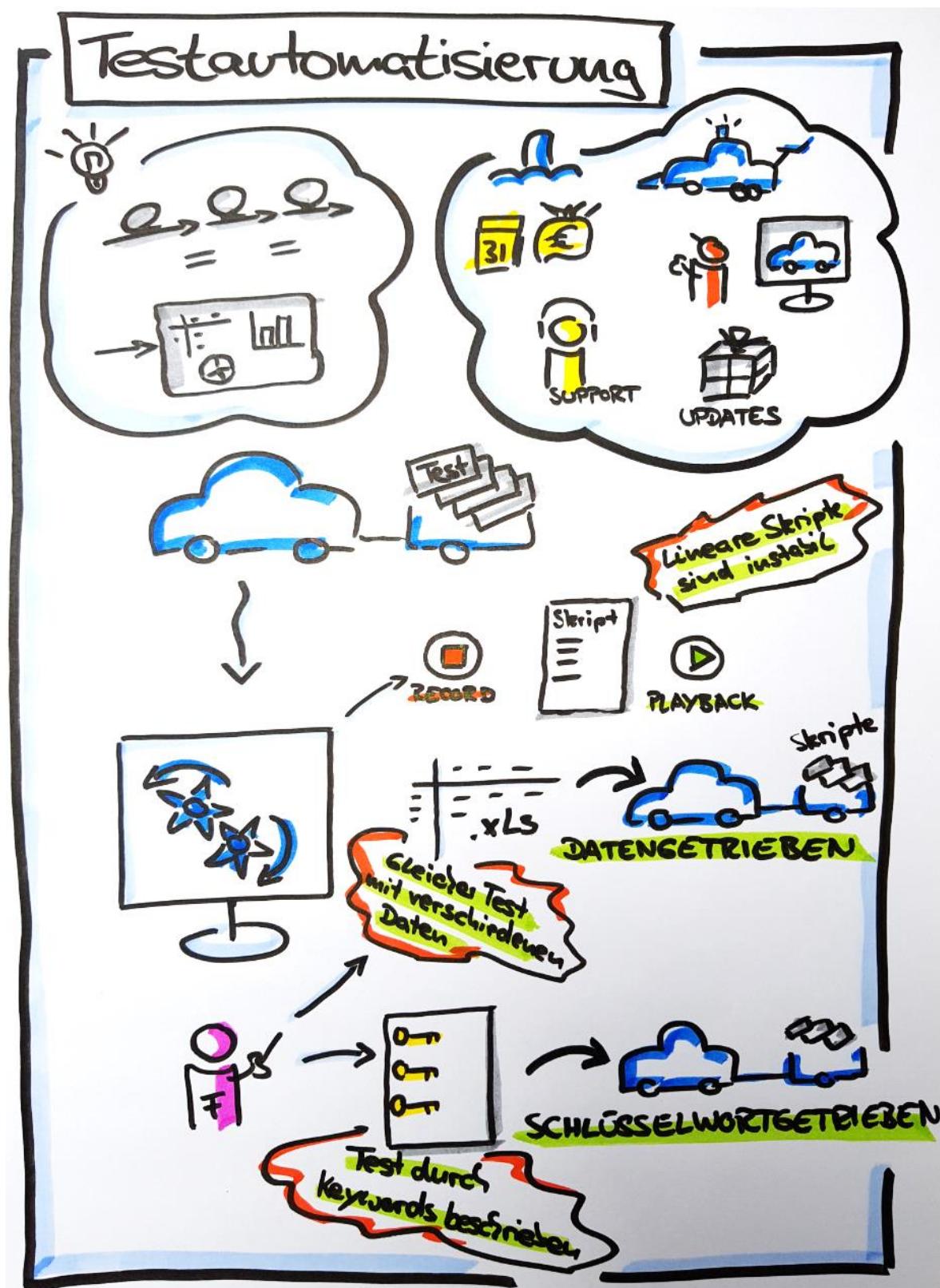
Für spezielle Testbedürfnisse existieren zusätzlich zu der im Flipchart gezeigten Kategorisierung weitere Werkzeuge für [Zitat Lehrplan Abschnitt 6.1.1, Seite 84]):

- Datenqualitätsbewertung
- Datenkonvertierung und -migration
- Gebrauchstauglichkeitstest
- Testen der Barrierefreiheit
- Softwarelokalisierungstest
- IT-Sicherheitstest
- Portabilitätstest (z.B. Testen von Software über mehrere unterstützte Plattformen)

→ Die Beispiele für Testwerkzeuge können hier in die entsprechenden Gruppen eingeordnet werden, oder auch in das Bild des Testprozesses in Abschnitt 1.4 eingefügt werden.

FL-6.1.2 (K1) Nutzen und Risiken der Testautomatisierung identifizieren können

FL-6.1.3 (K1) Sich an besondere Gesichtspunkte von Testdurchführungs- und Testmanagementwerkzeugen erinnern können



Beispiele	Übungen	Schlüsselbegriffe
		Datengetriebenes Testen, schlüsselwortgetriebener Test, Testautomatisierung

Erläuterungen

Testautomatisierung ist in Projekten interessant, um sich wiederholende, manuelle Arbeit zu übernehmen, für Konsistenz, Wiederholbarkeit und Objektivität zu sorgen, sowie einfacher Informationen über das Testen erheben zu können [Lehrplan, Abschnitt 6.1.2].

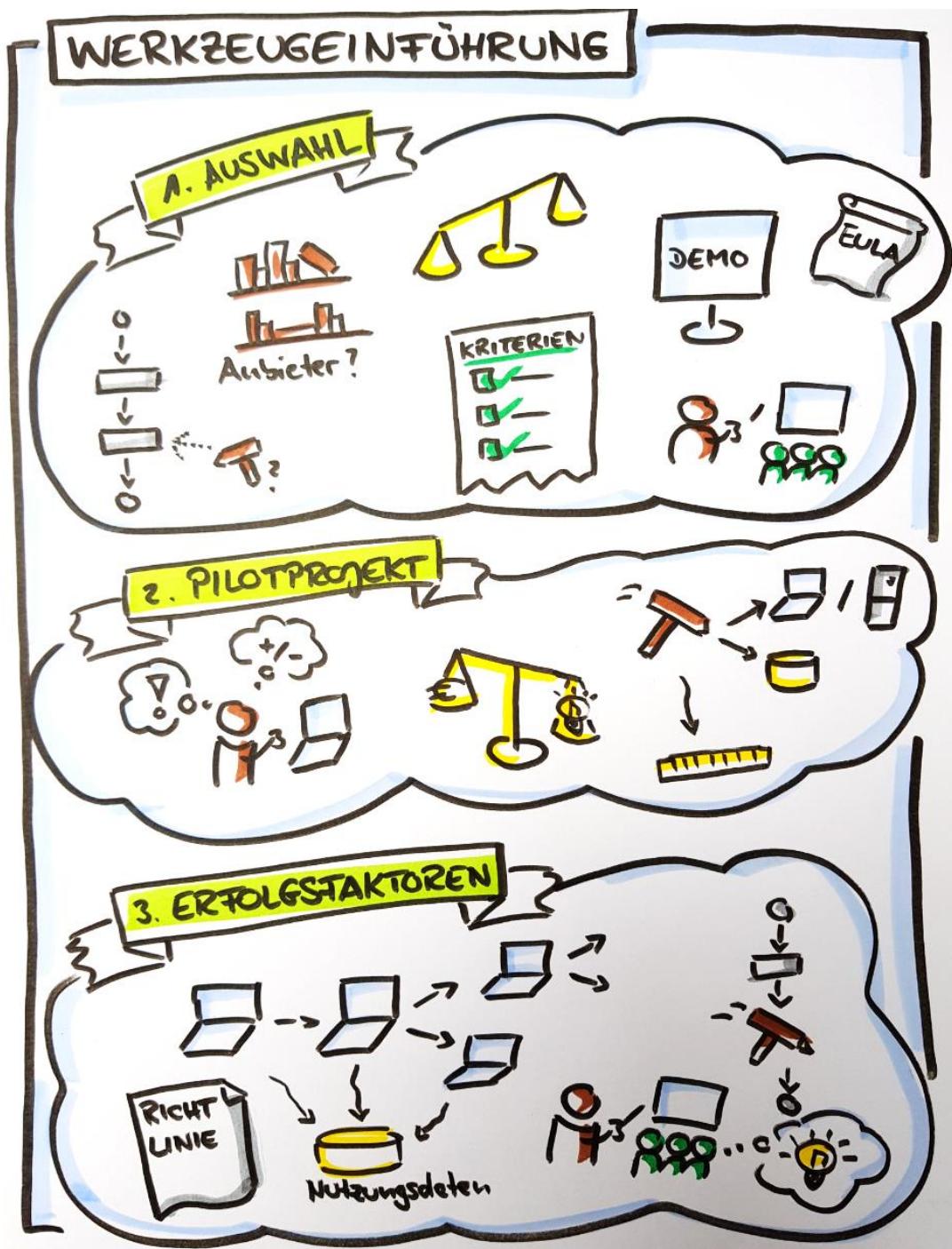
Die Testautomatisierung birgt aber auch eine Anzahl nicht unerheblicher Risiken, die der Lehrplan, Abschnitt 6.1.2, listet.

Zwar lassen sich Testautomatisierungswerkzeuge in allen Aktivitäten des Testentwicklungsprozesses verwenden, häufig besteht aber ein starker Fokus auf die Testausführung. Hierzu verwenden die Werkzeuge automatisierte Skripte, die Aktionen beschreiben, die sonst ein manueller Tester ausführen würde. Einfache, aufgezeichnete oder linear ablaufende Skripte sind nicht sehr robust bei Änderungen am Testobjekt, weshalb häufig **datengetriebene** oder **schlüsselwortgetriebene Tests** verwendet werden.

Ein wesentlicher Vorteil Daten- und Schlüsselwortgetriebener Tests ist, dass die Erstellung der eigentlichen Testfälle losgelöst werden kann von der eher entwicklungsnahen Testautomatisierung, d.h. die Technik zur Ansteuerung des Testobjekts wird von der Logik der Testfallerstellung getrennt.

6.2 Effektive Nutzung von Werkzeugen

- FL-6.2.1 (K1) Die Hauptprinzipien für die Auswahl eines Werkzeugs identifizieren können
- FL-6.2.2 (K1) Sich an Ziele für die Nutzung von Pilotprojekten zur Einführung von Werkzeugen erinnern können
- FL-6.2.3 (K1) Erfolgsfaktoren für die Evaluierung, Implementierung, Bereitstellung und kontinuierliche Unterstützung von Testwerkzeugen in einem Unternehmen identifizieren können



Beispiele	Übungen	Schlüsselbegriffe

Erläuterungen

Bei der Einführung von Testwerkzeugen beginnt ein Unternehmen in der Regel mit einem Auswahlprozess, für den der Lehrplan, Kapitel 6.2.1, eine Reihe von Kriterien listet. Für Produkte, die den Kriterien genügen, lassen sich dann Machbarkeitsstudien (Proof-of-Concept) erstellen.

Hat man sich schließlich auf ein Produkt festgelegt, sollte dies nicht direkt global eingeführt werden, sondern die Auswahl eines Pilotprojekts und die Evaluierung des Produktes in der Praxis sollten einer stufenweisen Einführung vorangehen. Hierzu listet der Lehrplan, Abschnitt 6.2.2, eine Reihe von Zielen für Pilotprojekte.

Der Werkzeugeinsatz ist auch nach einem erfolgreichen Pilotprojekt von weiteren Erfolgsfaktoren abhängig, auch hier listet der Lehrplan, Abschnitt 6.2.3, eine Reihe von Stichpunkten, die das Flipchart bildlich darstellt.

Die kontinuierliche Überwachung des Werkzeugeinsatzes, die vollständige Integration in den Softwareentwicklungslebenszyklus und in die Art der Integration in die Organisation/das Unternehmen an sich sichert die erfolgreiche Nutzung eines Werkzeuges.

Platz für Notizen

About Sogeti

Sogeti is a leading provider of technology and engineering services. Sogeti delivers solutions that enable digital transformation and offers cutting-edge expertise in Cloud, Cybersecurity, Digital Manufacturing, Digital Assurance & Testing, and emerging technologies. Sogeti combines agility and speed of implementation with strong technology supplier partnerships, world class methodologies and its global delivery model, Rightshore®. Sogeti brings together more than 25,000 professionals in 15 countries, based in over 100 locations in Europe, USA and India. Sogeti is a wholly-owned subsidiary of Capgemini SE, listed on the Paris Stock Exchange.

Learn more about us at www.sogeti.com

This document contains information that may be privileged or confidential and is the property of the Sogeti Group.
Copyright © 2019 Sogeti.