ISTQB Gesamtdokumentation

Inhaltsverzeichnis

Grundlagen des Softwaretestens

Warum Software getestet werden muss

Qualitätskontrolle in der digitalen Welt

In der traditionellen Industrie ist Qualitätskontrolle selbstverständlich - Produkte werden systematisch überprüft, bevor sie das Werk verlassen. Software stellt uns jedoch vor besondere Herausforderungen: Sie ist nicht greifbar, nicht sichtbar und lässt sich nicht mit herkömmlichen Methoden prüfen.

> Die immaterielle Natur von Software erfordert völlig neue Ansätze zur Qualitätssicherung.

Fehlerhafte Software kann dramatische Folgen haben: Von finanziellen Verlusten über Reputationsschäden bis hin zu Gefahren für Leib und Leben. Moderne Beispiele wie Fehlfunktionen in Fahrassistenzsystemen oder fehlerhafte Bankensoftware zeigen die weitreichenden Konsequenzen mangelhafter Qualitätssicherung.

Ein Online-Shop mit fehlerhafter Bestellabwicklung kann binnen weniger Stunden erhebliche finanzielle Schäden für alle Beteiligten verursachen und das Kundenvertrauen nachhaltig erschüttern.

Testen als Risikomanagement

Systematisches Testen reduziert Risiken und trägt zum Projekterfolg bei. Es ermöglicht die frühzeitige Erkennung von Problemen, bevor diese in der Produktionsumgebung auftreten. Dabei profitieren nicht nur spezialisierte Testteams - jeder Projektbeteiligte kann und sollte zur Qualitätssicherung beitragen.

Der Testprozess in der Praxis

Mehr als nur Code ausführen

Viele verstehen unter Testen lediglich die Ausführung von Programmen mit Testdaten. In Wirklichkeit umfasst professionelles Testen einen umfassenden Testprozess mit verschiedenen Phasen:

- Testplanung und Strategie: Testaufwand schätzen, Ressourcen planen
- Testanalyse und Design: Testfälle entwickeln, Testdaten definieren

- Testdurchführung: Tests ausführen und dokumentieren
- Bewertung: Ergebnisse analysieren, Qualität beurteilen

Werkzeuge und menschliche Expertise

Moderne Testwerkzeuge automatisieren viele Routineaufgaben und ermöglichen komplexe Testszenarien. Dennoch bleibt Testen eine primär intellektuelle Tätigkeit, die analytisches Denken, Fachwissen und die Fähigkeit zum kritischen Hinterfragen erfordert.

Testarten und Ansätze

Statische versus dynamische Verfahren

Nicht alle Tests erfordern laufende Software. Statische Tests prüfen Dokumente, Spezifikationen und Quellcode bereits vor der ersten Programmausführung. Dynamische Tests hingegen untersuchen das Verhalten der ausführbaren Software.

> Je früher Fehler entdeckt werden, desto kostengünstiger ist deren Behebung.

Verifikation trifft Validierung

Effektives Testen kombiniert zwei Perspektiven: - Verifikation: Entspricht das System den Spezifikationen? - Validierung: Erfüllt das System die tatsächlichen Nutzerbedürfnisse?

Die Nutzerperspektive ist dabei entscheidend. Da echte Anwender oft nicht kontinuierlich verfügbar sind, muss das Testteam deren Sichtweise stellvertretend einnehmen.

Grenzen und Realitäten

Die Illusion der Perfektion

Absolut fehlerfreie Software existiert nicht - zumindest nicht bei Systemen von relevanter Komplexität. Häufige Fehlerquellen sind:

- Übersehene Ausnahmesituationen
- Unberücksichtigte Randbedingungen
- Komplexe Wechselwirkungen zwischen Systemkomponenten

Trotz dieser Einschränkungen arbeiten unzählige Softwaresysteme täglich zuverlässig und erfüllen ihre Aufgaben.

Testen beweist keine Fehlerfreiheit

Selbst umfangreiche Tests können nicht garantieren, dass keine weiteren Fehler existieren. Testen kann die Anwesenheit von Fehlern aufdecken, aber niemals deren vollständige Abwesenheit beweisen.

Diese Erkenntnis führt zu einer pragmatischen Herangehensweise: Software wird produktiv gesetzt, wenn die identifizierten Risiken akzeptabel sind und die erkannten Fehler das Nutzungsziel nicht gefährden.

Fazit

Professionelles Testen ist unverzichtbar für qualitativ hochwertige Software. Es minimiert Risiken, erhöht die Zuverlässigkeit und trägt maßgeblich zum Projekterfolg bei. Obwohl absolute Perfektion unerreichbar bleibt, ermöglicht systematisches Testen die bestmögliche Qualität unter den gegebenen Umständen.

Der Schlüssel liegt in der Kombination verschiedener Testansätze, der Integration aller Projektbeteiligten und einem realistischen Verständnis der Möglichkeiten und Grenzen des Testens.

Fehlerbegriffe und Fehleranalyse im Softwaretest

Was macht einen Fehler zum Fehler?

Die Testbasis als Bewertungsgrundlage

Ein Verhalten kann nur dann als fehlerhaft bewertet werden, wenn zuvor definiert wurde, was als korrekt gilt. Diese Referenz bildet die sogenannte Testbasis - eine Sammlung von Anforderungen, Spezifikationen und weiteren Informationen, die als Maßstab für die Bewertung dient.

> Ein Fehler ist die Nichterfüllung einer festgelegten Anforderung - die Abweichung zwischen dem tatsächlichen und dem erwarteten Systemverhalten.

Software altert nicht wie physische Produkte. Jeder Softwarefehler existiert bereits seit dem Zeitpunkt seiner Entstehung im Code und wird erst bei der Ausführung sichtbar.

Die Fehlerkette verstehen

Von der Ursache zur sichtbaren Auswirkung

Fehlerwirkung (Failure): Das nach außen sichtbare, fehlerhafte Verhalten des Systems. Dies kann ein falscher Ausgabewert, ein Systemabsturz oder eine nicht funktionierende Funktion sein.

Fehlerzustand (Fault/Defekt/Bug): Die eigentliche Ursache im Code - beispielsweise eine falsch programmierte Anweisung oder ein vergessener Programmteil.

Fehlhandlung (Error): Die menschliche Handlung, die zum Fehlerzustand führte, wie fehlerhafte Programmierung oder Missverständnisse bei der Anforderungsanalyse.

Komplexe Wechselwirkungen

Fehlermaskierung kann auftreten, wenn sich mehrere Fehlerzustände gegenseitig kompensieren. Eine Fehlerwirkung wird erst sichtbar, nachdem maskierende Fehler korrigiert wurden. Dies erklärt, warum Fehlerbehebungen manchmal neue Probleme verursachen.

Ein Fehlerzustand muss nicht zwangsläufig zu einer Fehlerwirkung führen. Die Auswirkung kann ausbleiben, sporadisch oder systematisch auftreten und sich weit entfernt von der ursprünglichen Fehlerquelle manifestieren.

Häufige Ursachen für Fehlhandlungen

Menschliche Faktoren

- Zeitdruck - in Softwareprojekten allgegenwärtig
- Komplexität der Aufgaben, Architekturen und Technologien
- Unaufmerksamkeit durch Ablenkung, Konzentrationsmangel oder Müdigkeit

Kommunikation und Verständnis

- Missverständnisse bei der Interpretation von Anforderungen
- Unklare Schnittstellen zwischen Systemkomponenten
- Unterschiedliche Auslegungen von Spezifikationen zwischen Projektbeteiligten

Technische Herausforderungen

- Neue Technologien, die noch nicht vollständig verstanden werden
- Unerfahrenheit oder unzureichende Ausbildung
- Komplexe Systeminteraktionen bei großen Anwendungen

Testergebnisse richtig interpretieren

Falsch-positive und falsch-negative Ergebnisse

Nicht jedes unerwartete Testergebnis deutet auf einen echten Softwarefehler hin:

Ergebnistyp	Beschreibung	Konsequenz
Falsch-positiv	Test zeigt Fehler an, obwohl das System korrekt funktioniert	Unnötige Fehlersuche
Falsch-negativ	Test übersieht vorhandenen Fehler	Fehler bleibt unentdeckt
Richtig-positiv	Test deckt tatsächlichen Fehler auf	Erfolgreiche Fehlerfindung
Richtig-negativ	Test bestätigt korrektes Verhalten	Verifikation der Funktionalität

Praktisches Beispiel: Unklare Anforderungen

Ein Finanzierungssystem soll Kreditzinssätze basierend auf Bonitätsprüfungen reduzieren. Die Anforderung verschwieg jedoch, dass diese Reduktion bei Sonderaktionen nicht gilt. Kunden erhielten dadurch initially zu niedrige Zinssätze angeboten, was zu Beschwerden bei der späteren Abrechnung führte.

Dieses Beispiel zeigt, wie unvollständige Spezifikationen zu Fehlhandlungen in der Entwicklung und letztendlich zu Problemen im Produktivbetrieb führen.

Lernen aus Fehlern

Prozessverbesserung durch Fehleranalyse

Die systematische Analyse von Fehlern und deren Ursachen ermöglicht nachhaltigen Lerneffekt. Durch die Untersuchung der Fehlerkette - von der Fehlhandlung über den Fehlerzustand bis zur Fehlerwirkung - können Prozessverbesserungen implementiert werden, die ähnliche Probleme in Zukunft verhindern.

Abgrenzung: Testen versus Debugging

Testen identifiziert Fehlerwirkungen, Debugging lokalisiert und behebt Fehlerzustände. Während Testen primär das Aufdecken von Problemen zum Ziel hat, fokussiert sich Debugging auf die Ursachenanalyse und Problemlösung im Code.

Fazit

Das Verständnis der Fehlerkette ist fundamental für effektives Testen. Die klare Unterscheidung zwischen Fehlhandlung, Fehlerzustand und Fehlerwirkung ermöglicht es, sowohl die unmittelbaren Symptome als auch die zugrundeliegenden Ursachen systematisch anzugehen.

Erfolgreiche Qualitätssicherung erfordert mehr als das reine Aufspüren von Fehlern - sie umfasst die kontinuierliche Verbesserung von Prozessen und die Schaffung von Rahmenbedingungen, die Fehlhandlungen minimieren.

Testbegriff und Testziele

Testen versus Debugging - Eine wichtige Abgrenzung

Klare Aufgabentrennung

Testen und Debugging werden häufig verwechselt, obwohl es sich um völlig unterschiedliche Aktivitäten handelt:

- Testen zielt darauf ab, Fehlerwirkungen gezielt aufzudecken
- Debugging lokalisiert und behebt die zugrundeliegenden Fehlerzustände im Code

  Während Testen Probleme sichtbar macht, löst Debugging diese im Quellcode.

Das Debugging obliegt dem jeweiligen Entwicklungsteam, das für die betroffene Codekomponente verantwortlich ist. Erst wenn eine Fehlerwirkung durch Tests identifiziert wurde, kann die Fehlersuche und -behebung beginnen.

Fehlernachtests - Kontrolle der Korrekturen

Nach der Fehlerbehebung müssen Fehlernachtests durchgeführt werden, um zu verifizieren, dass die Korrekturmaßnahmen erfolgreich waren. Diese Tests führen häufig dieselben Personen durch, die auch die ursprünglichen Fehler entdeckt haben.

Bei automatisierten Tests ist der Fehlernachtest besonders einfach: Der zuvor fehlgeschlagene Test sollte nun erfolgreich durchlaufen.

Seiteneffekte und Regressionen

Fehlerbehebungen können unbeabsichtigt neue Probleme verursachen. Solche Seiteneffekte manifestieren sich oft bei völlig anderen Eingabekonstellationen und erfordern zusätzliche Regressionstests, die über den ursprünglich fehlgeschlagenen Test hinausgehen.

Vielfältige Testziele

Qualitätsbewertung und Vertrauen schaffen

Testen verfolgt mehrere strategische Testziele:

- Qualitative Bewertung von Arbeitsergebnissen wie Spezifikationen, User Stories und Programmcode
- Vollständigkeitsnachweis der Anforderungsumsetzung
- Vertrauensbildung bei Stakeholdern durch fundierte Qualitätsinformationen

Risikominimierung und Compliance

- Risikoreduktion durch Aufdeckung und Behebung von Fehlerwirkungen
- Compliance-Nachweis für vertragliche, rechtliche oder regulatorische Anforderungen
- Integrationsentscheidungen durch Bewertung der Systemteil-Qualität

Technische Zielsetzungen

Überdeckungsmessung: Tests liefern Informationen darüber, ob geforderte Überdeckungsziele erreicht wurden oder zusätzliche Testfälle benötigt werden.

Freigabeentscheidungen: Systematische Bereitstellung von Informationen für fundierte Entscheidungen über Systemfreigaben.

Kontextabhängige Zielsetzungen

Variationen nach Entwicklungsmodell und Teststufe

Die spezifischen Testziele variieren erheblich je nach:

- Entwicklungsmodell (agil versus traditionell)
- Teststufe (Komponenten-, System-, Abnahmetest)
- Projektkontext und Stakeholder-Erwartungen

Beispiele spezifischer Zielsetzungen

Komponententests konzentrieren sich primär auf: - Maximale Fehlerwirkungsaufdeckung - Hohe Codeüberdeckung - Frühzeitige Fehlerzustandsidentifikation

Abnahmetests fokussieren auf: - Nachweis der Gebrauchstauglichkeit - Erfüllung funktionaler und nicht-funktionaler Anforderungen
- Risikobewertung für Freigabeentscheidungen

Systematik der Testbezeichnungen

Kategorien der Testbenennung

Die Vielfalt der Testbegriffe lässt sich systematisch ordnen:

Kategorie	Beschreibung	Beispiele
Testziel	Benennung nach Testzweck	Lasttest, Sicherheitstest
Testmethode	Benennung nach Verfahren	Zustandsbasierter Test
Testobjekt	Benennung nach Prüfgegenstand	GUI-Test, Datenbanktest
Teststufe	Benennung nach Entwicklungsphase	Systemtest, Integrationstest
Testperson	Benennung nach Durchführenden	Entwicklertest, Anwendertest
Testumfang	Benennung nach Abdeckungsgrad	Partieller Regressionstest

Perspektivenwechsel statt neue Testarten

Viele scheinbar unterschiedliche Testarten sind lediglich verschiedene Betrachtungsweisen derselben Testaktivitäten. Die Benennung erfolgt je nach Fokus der jeweiligen Diskussion oder Dokumentation.

Testbasis als Fundament

Die Testbasis bildet das Referenzsystem für alle Testaktivitäten. Sie umfasst sämtliche Informationen, die zur Bestimmung korrekten Systemverhaltens herangezogen werden:

- Anforderungsspezifikationen
- User Stories

- Systemdokumentation
- Standards und Richtlinien

  Ohne klare Testbasis ist keine objektive Fehlerbewertung möglich.

Fazit

Erfolgreiches Testen erfordert klare Zieldefinition und systematisches Vorgehen. Die Unterscheidung zwischen Testen und Debugging, das Verständnis für kontextabhängige Zielsetzungen und die strukturierte Betrachtung verschiedener Testaspekte bilden das Fundament professioneller Qualitätssicherung.

Die Vielfalt der Testbegriffe sollte nicht verwirren, sondern als Werkzeugkasten für präzise Kommunikation über Testaktivitäten verstanden werden.

Testartefakte und ihre Zusammenhänge

Grundlagen der Testdokumentation

Die Testbasis als Fundament

Die Testbasis bildet das Herzstück aller Testaktivitäten. Sie umfasst sämtliche Dokumente und Informationen, die zur Beurteilung herangezogen werden, ob ein Testverhalten fehlerhaft ist oder den Erwartungen entspricht.

> Die Testbasis definiert das Sollverhalten des Testobjekts und ermöglicht objektive Bewertungen von Testergebnissen.

Typische Bestandteile der Testbasis: * Anforderungsdokumente und Spezifikationen * User Stories und Akzeptanzkriterien * Systemdokumentation * Gesunder Menschenverstand und Fachwissen

Von der Testbasis zum Testfall

Testbedingungen als Zwischenschritt

Da die gesamte Testbasis nicht durch einen einzigen Test überprüfbar ist, müssen fokussierte Aspekte definiert werden. Diese Testbedingungen (Test Conditions) repräsentieren spezifische Prüfpunkte, die für das Erreichen der Testziele relevant sind.

Beispiele für Testbedingungen: * Preisberechnungsfunktionen * Kombinationen von Produktkonfigurationen
* Benutzeroberflächen-Eigenschaften (“Look and Feel”) * Systemantwortzeitverhalten

Testelemente definieren

Parallel zur Ableitung von Testbedingungen werden Testelemente identifiziert - die konkreten Teile des Systems, die durch spezifische Testfälle geprüft werden sollen.

Beispiel: Zur Testbedingung “Preisberechnung” gehört das Testelement `calculate_price()` *- eine Methode, die mit entsprechenden Testfällen validiert wird.*

Testfall und Testlauf

Vom statischen Plan zur dynamischen Ausführung

Ein Testfall beschreibt die Prüfung einer oder mehrerer Testbedingungen. Wenn dieser Testfall auf dem Rechner ausgeführt wird - das Testobjekt also mit konkreten Testdaten versehen und gestartet wird - entsteht ein Testlauf.

Der Testlauf umfasst: * Herstellung der erforderlichen Vorbedingungen * Bereitstellung der Testdaten * Ausführung des Tests * Prüfung der Ergebnisse gegen erwartete Sollwerte

> Die Bewertung erfolgt durch Vergleich zwischen Istergebnis und Sollergebnis basierend auf der Testbasis.

Organisation und Strukturierung

Testsuiten und Testausführungspläne

Einzelne Testfälle isoliert auszuführen ist ineffizient. Stattdessen werden sie in Testsuiten gruppiert, die gemeinsam in einem Testzyklus durchlaufen werden.

Der Testausführungsplan legt die zeitliche Sequenz fest, in der verschiedene Testsuiten abgearbeitet werden, und koordiniert die gesamte Testaktivität.

Automatisierung durch Testskripte

Testskripte automatisieren die Ausführung von Testsuiten und enthalten: * Reihenfolge der Testfälle * Aktionen zur Vorbedingungsherstellung
* Aufräumaktivitäten nach Testdurchführung

Bei manuellen Tests werden diese Informationen als Testablauf (Test Procedure) dokumentiert.

Dokumentation und Berichtswesen

Protokollierung und Berichterstattung

Alle Testläufe müssen systematisch protokolliert werden. Die Einzelergebnisse fließen in zusammenfassende Testberichte ein, die Stakeholdern einen Überblick über Testfortschritt und -qualität geben.

Strategische Planung

Das Testkonzept steht am Anfang aller Testüberlegungen und definiert: * Auswahl der Testobjekte und Testverfahren * Festlegung der Testziele
* Koordination der Testaktivitäten * Berichterstattungsverfahren

Der Testzeitplan konkretisiert die zeitliche Dimension und koordiniert alle Testaktivitäten untereinander.

Übersicht der Testartefakte

Artefakt	Beschreibung	Zweck	Erstellungsphase
Testbasis	Gesamtheit aller Referenzdokumente und Informationen	Definiert Sollverhalten und Bewertungsgrundlage	Projektbeginn
Testbedingung	Spezifischer, testbarer Aspekt der Testbasis	Fokussierung auf prüfbare Eigenschaften	Testanalyse

Artefakt	Beschreibung	Zweck	Erstellungsphase
Testelement	Konkreter Systemteil, der getestet wird	Identifikation der Prüfobjekte	Testanalyse
Testfall	Spezifikation einer Prüfung	Beschreibt WAS getestet wird	Testdesign
Testlauf	Ausführung eines Testfalls auf dem System	Erzeugt tatsächliche Testergebnisse	Testdurchführung
Testsuite	Zusammenfassung mehrerer Testfälle	Organisiert Testfälle in logische Gruppen	Testdesign
Testausführungsplan	Zeitliche Ablaufplanung der Testsuiten	Koordiniert Testreihenfolge und -zyklen	Testplanung
Testskript	Automatisierte Ausführungslogik	Ermöglicht wiederholbare, automatisierte Tests	Testimplementierung
Testablauf (Test Procedure)	Manuelle Ausführungsanweisung	Anleitung für manuelle Testdurchführung	Testimplementierung
Testprotokoll	Dokumentation der Testlaufergebnisse	Erfasst alle Testresultate und Beobachtungen	Testdurchführung
Testbericht	Zusammenfassung der Testergebnisse	Informiert Stakeholder über Testqualität und -fortschritt	Testauswertung
Testkonzept	Strategisches Planungsdokument	Definiert Testansatz, -ziele und -organisation	Testplanung

Artefakt	Beschreibung	Zweck	Erstellungsphase
Testzeitplan	Zeitliche Koordination aller Testaktivitäten	Plant Ressourcen und Termine	Testplanung

Zusammenhänge verstehen

Prozessorientierte Betrachtung

Die verschiedenen Artefakte entstehen in unterschiedlichen Phasen des Testprozesses:

1. Testplanung: Testkonzept und Testzeitplan
2. Testanalyse: Testbedingungen und Testelemente

3. Testdesign: Testfälle und Testsuiten
4. Testimplementierung: Testskripte und Testabläufe
5. Testdurchführung: Testläufe und Protokolle
6. Bewertung: Testberichte

Praktische Umsetzung

Die Qualität der Testartefakte bestimmt maßgeblich den Erfolg der gesamten Testaktivität. Unvollständige oder ungenaue Testbasis führt zu unzuverlässigen Testfällen, während schlecht strukturierte Testsuiten die Effizienz der Testdurchführung beeinträchtigen.

> Systematische Dokumentation und klare Strukturierung der Testartefakte sind Voraussetzung für reproduzierbare und aussagekräftige Testergebnisse.

Fazit

Professionelles Testen erfordert eine durchdachte Struktur von Artefakten, die aufeinander aufbauen und sich gegenseitig ergänzen. Von der grundlegenden Testbasis über spezifische Testbedingungen bis hin zu ausführbaren Testskripten bildet jedes Element einen wichtigen Baustein im Gesamtsystem der Qualitätssicherung.

Die systematische Verwaltung dieser Artefakte ermöglicht nicht nur effiziente Testdurchführung, sondern auch Nachvollziehbarkeit, Wiederverwendbarkeit und kontinuierliche Verbesserung der Testprozesse.

Testaufwand und Risikobasiertes Testen

Die Ökonomie des Testens

Testaufwand in der Praxis

Testen beansprucht einen erheblichen Teil des Entwicklungsbudgets, wobei nur ein Bruchteil aller theoretisch möglichen Testfälle durchgeführt werden kann. Eine pauschale Empfehlung für den optimalen Testaufwand existiert nicht - die Investition hängt stark vom Projektcharakter und den spezifischen Risiken ab.

Traditionelle Kennzahlen wie das Verhältnis von Testern zu Entwicklern (von 1:10 bis 3:1) verlieren in agilen Umgebungen an Aussagekraft, da die Rollentrennung aufweicht. Stattdessen müssen Aufgabentypen im Backlog und Budgetverteilungen analysiert werden.

Praktisches Beispiel: Kombinatorische Explosion

Das Pixel Leap Charaktersystem

Stellen wir uns vor, "Pixel Leap" soll ein umfangreiches Charakteranpassungssystem erhalten:

Anpassungsoptionen: * 8 Charaktermodelle × 6 Kostümvarianten * 12 Farbschemata × 4 Effekte (Matt, Glanz, Metallic, Neon)
* 10 Fähigkeiten-Sets × 5 Schwierigkeitsgrade * 20 optionale Accessoires (jeweils an/aus)

Mathematische Realität: - Basis-Kombinationen: 8 × 6 × 12 × 4 × 10 × 5 = 115.200 - Mit Accessoires: 115.200 × 2^{2}□ = 121.006.080.000.000 mögliche Konfigurationen

> Selbst bei nur 1 Sekunde pro Test würden vollständige Tests über 3,8 Millionen Jahre dauern.

Intelligente Teststrategien

Anstatt jede Kombination zu testen, fokussiert sich das Team auf: * Validierung der Regel-Engine für ungültige Kombinationen * Kombinatorisches Testen für repräsentative Stichproben * Risikobewertung verschiedener Konfigurationen

Risikobasierte Testpriorisierung

Schadenspotenzial bestimmt Testintensität

Jerry Weinbergs Kernfrage: "Im Vergleich zu was?" verdeutlicht die Notwendigkeit einer Risiko-Nutzen-Analyse beim Testen.

> Testen ist ökonomisch sinnvoll, solange die Kosten für Fehlerfindung und -behebung niedriger sind als potenzielle Schäden durch unentdeckte Fehler.

Beispiel: Verschiedene Risikostufen in Pixel Leap

Komponente	Risikostufe	Potenzielle Schäden	Testintensität
Speicher/Lade-System	HOCH	Spielfortschritt verloren → Spieler-Abwanderung	Umfangreiche Tests, mehrere Plattformen
Online-Bestenliste	MITTEL	Falsche Rankings → Frustration	Standardtests, Edge-Cases
Kosmetische Effekte	NIEDRIG	Visuelle Glitches → Geringfügige Störung	Grundfunktionalität, Stichproben
In-App-Käufe	SEHR HOCH	Falsche Abrechnungen → Rechtliche Probleme	Vollständige Testabdeckung, Penetrationstests

Katastrophale Beispiele aus der Realität

Das Hitomi-Weltraumteleskop (2016): Softwarefehler führten zu unkontrollierter Rotation und Totalverlust - Schaden: mehrere hundert Millionen Euro. *Für einen solchen Schaden hätten nahezu unbegrenzte Testressourcen gerechtfertigt werden können.*

Risikoanalyse für Spieleentwicklung

Kritische Bereiche identifizieren

Besonders risikobehaftete Spielkomponenten:

- Progression-Systeme: Charakterfortschritt, Level-Freischaltungen
- Monetarisierung: In-App-Käufe, Premium-Inhalte, Abonnements
- Mehrspieler-Synchronisation: Konsistenz zwischen Clients
- Plattform-Integration: Achievement-Systeme, Cloud-Saves

Schadensbewertung in der Gaming-Industrie

Auch scheinbar harmlose Bugs können verheerend sein: Ein fehlerhaftes Speichersystem in "Pixel Leap" könnte zwar keine direkten physischen Schäden verursachen, aber:

- Sofortige Auswirkungen: Negative Reviews, Rückerstattungsforderungen
- Langfristige Folgen: Reputationsschäden, reduzierte Verkäufe zukünftiger Titel
- Marktpositionierung: Verlust der Glaubwürdigkeit als Premium-Entwickler

Strategische Testplanung

Risikomatrix für Testpriorisierung

Schadenshöhe	Eintrittswahrscheinlichkeit	Testaufwand	Beispiel
Hoch/Hoch	Sehr hoch	Maximum	Kaufsystem-Bugs
Hoch/Niedrig	Hoch	Umfangreich	Seltene Speicher-Korruption
Niedrig/Hoch	Mittel	Standard	UI-Anzeigeprobleme

Schadenshöhe	Eintrittswahrscheinlichkeit	Testaufwand	Beispiel
Niedrig/Niedrig	Minimal	Grundlegend	Kosmetische Effekte

Testbudget-Allokation

Empfohlene Verteilung für "Pixel Leap": - 40% - Kern-Gameplay und Speichersystem - 25% - Plattform-Integration und Performance - 20% - Online-Features und Monetarisierung
- 10% - Benutzeroberfläche und Accessibility - 5% - Kosmetische und optionale Features

Internationale Standards und Best Practices

Sicherheitskritische Systeme als Vorbild

Standards wie DO-178C (Luftfahrt) definieren strenge Testanforderungen basierend auf Kritikalitätsstufen. Obwohl Spiele selten lebenskritisch sind, können ähnliche Prinzipien angewendet werden:

- Kritikalitätseinstufung aller Systemkomponenten
- Dokumentierte Risikobewertung für jede Funktion
- Traceability zwischen Risiken und Testmaßnahmen
- Unabhängige Verifikation kritischer Bereiche

Fazit

Effektives Testmanagement erfordert eine durchdachte Balance zwischen Ressourcen und Risiken. Während theoretisch unendliche Testmöglichkeiten existieren, müssen praktische Entscheidungen auf fundierten Risikoanalysen basieren.

Der Schlüssel liegt in der systematischen Bewertung: Welche Komponenten bergen das höchste Schadenspotenzial? Wo ist die Wahrscheinlichkeit von Fehlern am größten? Wie können begrenzte Testressourcen optimal eingesetzt werden?

Für "Pixel Leap" bedeutet dies: Priorität auf spielkritische Systeme wie Speicherfunktionen und Kaufabwicklung, während kosmetische Features mit Grundtests auskommen können. Die Kunst liegt darin, diese Balance kontinuierlich an neue Erkenntnisse und Marktanforderungen anzupassen.

Testwissen frühzeitig und erfolgreich nutzen

Der Shift-Left-Ansatz im Softwaretest

Mit dem Testen - genauer mit den Überlegungen und vorbereitenden Arbeiten zum Testen - soll so früh wie möglich begonnen werden. Dieser Ansatz wird auch als Shift-Left-Ansatz bezeichnet.

> Frühzeitige Integration von Testwissen in alle Entwicklungsphasen reduziert Risiken und verbessert die Softwarequalität nachhaltig.

Vorteile der frühen Tester-Beteiligung

Prüfung der Anforderungen

Beteiligen sich Tester an der Prüfung der Anforderungen oder an der Verfeinerung von User Stories, können Unklarheiten und Fehler in den Arbeitsprodukten aufgedeckt und behoben werden.

Beispiel aus "Pixel Leap": Bei der User Story "Als Spieler möchte ich sammeln können" identifiziert der Tester fehlende Spezifikationen: Welche Objekte? Wie viele? Was passiert beim Sammeln? Die präzisierte Anforderung verhindert spätere Missverständnisse.

Die Identifikation und Behebung fehlerhafter Anforderungen reduziert das Risiko der Entwicklung falscher oder nicht testbarer Funktionen.

Zusammenarbeit beim Systemdesign

Die enge Zusammenarbeit von Testern mit Systemdesignern während des Entwurfs kann das Verständnis jeder Partei für das Design und dessen Test erheblich verbessern.

Beispiel: Beim Design des Kollisionssystems in "Pixel Leap" schlägt der Tester modulare Schnittstellen vor, die isolierte Tests der Sprung-Plattform-Interaktionen ermöglichen. Dies verhindert später aufwendige Architekturänderungen.

Dieses erhöhte Verständnis kann das Risiko grundlegender Konstruktionsfehler reduzieren und ermöglicht die frühzeitige Identifikation potenzieller Tests zur Prüfung der Schnittstellen.

Kollaboration während der Codeerstellung

Arbeiten Entwickler und Tester während der Codeerstellung zusammen, kann das Verständnis auf beiden Seiten für den Code und dessen Test verbessert werden.

Beispiel: Während der Implementierung des Punktesystems diskutieren Entwickler und Tester gemeinsam Edge-Cases wie negative Punkte oder Überlauf-Situationen. Dadurch entstehen sowohl robusterer Code als auch bessere Testfälle.

Dies reduziert das Risiko von Fehlerzuständen im Programmtext und auch von fehlerhaften Tests zur Prüfung des Programmtextes (falsch negatives Ergebnis).

Verifikation und Validierung vor Freigabe

Wenn Tester die Software vor deren Freigabe verifizieren und validieren, können weitere Fehlerzustände erkannt und behoben werden, die ansonsten unentdeckt geblieben wären.

Beispiel: Vor Release von "Pixel Leap" entdeckt der Systemtest, dass bestimmte Sprungkombinationen bei schwächeren Geräten zu Performance-Problemen führen - ein Problem, das in der isolierten Entwicklungsumgebung nicht aufgefallen war.

Dies erhöht die Wahrscheinlichkeit, dass die Software den Bedürfnissen der Interessenvertreter gerecht wird und die Anforderungen erfüllt.

Zusätzlicher Nutzen

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

Praktische Auswirkungen

Durch frühzeitige Testbeteiligung entstehen messbare Vorteile: * Reduzierte Nacharbeit durch klarere Anforderungen * Weniger Architektur-Refactoring durch testfreundliches Design
* Geringere Bugrate durch kollaborative Entwicklung * Höhere Kundenzufriedenheit durch bessere Validierung

Fazit

Der Shift-Left-Ansatz macht aus Testern Partner im gesamten Entwicklungsprozess statt nur Prüfer am Ende der Kette. Die frühe Integration von Testwissen in alle Phasen - von Anforderungen über Design bis zur Implementierung - führt zu qualitativ besserer Software und effizienteren Entwicklungsprozessen.

Für "Pixel Leap" bedeutet dies: Testexpertise fließt von der ersten Konzeptidee bis zum finalen Release kontinuierlich ein und stellt sicher, dass das Spiel nicht nur technisch funktioniert, sondern auch die Spielererwartungen erfüllt.

Die sieben Grundsätze des Testens

Fundamentale Prinzipien der Qualitätssicherung

Die folgenden sieben Grundsätze haben sich über Jahrzehnte der Testpraxis herauskristallisiert und gelten als universelle Leitlinien für professionelles Testen. Sie bilden das konzeptionelle Fundament für alle Testaktivitäten.

Grundsatz 1: Testen zeigt die Anwesenheit von Fehlerzuständen

Testen kann das Vorhandensein von Fehlerwirkungen und damit von Fehlerzuständen nachweisen. Je nach Testaufwand und Intensität verringert sich die Wahrscheinlichkeit, dass noch unentdeckte Fehlerzustände im Testobjekt vorhanden sind.

> Testen kann jedoch nicht beweisen, dass keine Fehlerzustände im Testobjekt vorhanden sind.

Praktisches Beispiel aus "Pixel Leap": *Wenn alle durchgeführten Tests der Sprungmechanik erfolgreich verlaufen, bedeutet dies nicht, dass das System fehlerfrei ist. Es könnte noch ungetestete Kombinationen geben (z.B. Sprung während eines Power-ups bei gleichzeitigem Plattformwechsel), die Fehler aufdecken würden.*

Selbst wenn keine Fehlerwirkungen im Test aufgezeigt wurden, ist dies kein Nachweis für Fehlerfreiheit oder Korrektheit.

Grundsatz 2: Vollständiges Testen ist nicht möglich

Ein vollständiger Test aller möglichen Eingabewerte und deren Kombinationen unter Berücksichtigung aller Vor- und Randbedingungen ist nicht durchführbar - außer bei sehr trivialen Testobjekten.

Beispiel der kombinatorischen Explosion: *In "Pixel Leap" mit 10 Level-Typen, 8 Charaktervarianten, 15 Power-ups und 20 Gegnertypen ergeben sich bereits 24.000 Basis-Kombinationen. Hinzu kommen verschiedene Plattform-Konfigurationen, Timing-Varianten und Spielerzustände - die Gesamtzahl wird schnell astronomisch.*

Tests sind immer nur Stichproben, und der Testaufwand ist deshalb unter Verwendung von Testverfahren nach Risiko und Prioritäten zu steuern.

Grundsatz 3: Frühes Testen spart Zeit und Geld

Testaktivitäten - sowohl statische als auch dynamische - sollen im System- oder Softwareentwicklungslebenszyklus so früh wie möglich beginnen und definierte Testziele verfolgen. Dieser Ansatz wird auch als "Shift Left" bezeichnet.

Kostenvorteil des frühen Testens: *Ein Fehler in der Spielmechanik-Spezifikation von "Pixel Leap", der in der Anforderungsphase für 1 Stunde Aufwand behoben werden kann, würde in der Implementierungsphase 10 Stunden und nach dem Release möglicherweise ein komplettes Update mit Hunderten von Arbeitsstunden erfordern.*

Durch frühzeitiges Prüfen werden Fehlerzustände frühzeitig erkannt und kostenintensive späte Änderungen reduziert oder vermieden.

Grundsatz 4: Häufung von Fehlerzuständen

Fehlerzustände sind in der Regel nicht gleichmäßig über das gesamte System verteilt. Vielmehr finden sich die meisten Fehlerzustände in nur wenigen Teilen (Komponenten) eines Systems.

Praktisches Beispiel: *In "Pixel Leap" zeigt sich, dass 80% aller Bugs im Kollisionssystem auftreten, während die Menü-Navigation nahezu fehlerfrei ist. Diese Beobachtung führt dazu, dass zusätzliche Testressourcen auf das Kollisionssystem konzentriert werden.*

Die geschätzte oder beobachtete Anhäufung von Fehlerzuständen kann zur Risikoanalyse genutzt werden, um den Testaufwand gezielt auf fehlerträchtige Bereiche zu konzentrieren.

Grundsatz 5: Testfälle "nutzen sich ab"

Werden Tests an unveränderten Systemversionen nur wiederholt, decken sie keine neuen Fehlerwirkungen mehr auf. Damit die Effektivität der Tests nicht absinkt, sind vorhandene Testfälle regelmäßig zu prüfen und durch neue oder modifizierte Testfälle zu ergänzen.

Beispiel aus der Spieleentwicklung: *Die ursprünglichen Testfälle für "Pixel Leap" prüften Standard-Sprungsequenzen. Nach mehreren Updates finden diese Tests keine neuen Bugs mehr. Erst neue Testfälle für komplexe Sprung-Kombinationen oder Edge-Cases decken weitere Probleme auf.*

Trotzdem kann die Wiederholung unveränderten Testfälle sinnvoll sein, insbesondere bei automatisierten Regressionstests.

Grundsatz 6: Testen ist kontextabhängig

Je nach Einsatzgebiet und Umfeld des zu prüfenden Systems ist das Testen anzupassen. Keine zwei Systeme sind auf exakt gleiche Art und Weise zu testen.

Kontextuelle Unterschiede:

Systemtyp	Testfokus	Besonderheiten
Mobile Games	Performance, Akkulaufzeit, Touch-Interface	Verschiedene Gerätegrößen, OS-Versionen
Online-Multiplayer	Netzwerk-Synchronisation, Latenz	Server-Last, Verbindungsabbrüche
Embedded Games	Real-time Constraints, Hardware-Integration	Begrenzte Ressourcen, deterministische Abläufe

Intensität des Testens, Definition der Endekriterien und Testverfahren sind bei jedem System entsprechend seines Einsatzumfelds festzulegen.

Grundsatz 7: Trugschluss - Keine Fehler bedeutet ein brauchbares System

Trotz gründlicher Tests aller spezifizierten Anforderungen und Beheben aller gefundenen Fehlerzustände kann ein System entwickelt worden sein, das schwer zu nutzen ist oder die Bedürfnisse der Nutzer nicht erfüllt.

Beispiel aus "Pixel Leap": *Alle technischen Tests sind erfolgreich - Sprungmechanik funktioniert präzise, Kollisionen werden korrekt erkannt, Performance ist optimal. Dennoch beschweren sich Spieler, dass das Spiel zu schwierig und frustrierend ist, weil die Sprungdistanzen unintuitiv sind.*

> Ein technisch fehlerfreies System kann trotzdem unbrauchbar oder von schlechter Qualität sein.

Präventive Maßnahmen: * Frühzeitige Einbeziehung der späteren Nutzer in den Entwicklungsprozess * Nutzung von Prototyping und User-Testing * Validierung der User Experience zusätzlich zur technischen Verifikation

Fazit

Diese sieben Grundsätze bilden das konzeptionelle Rückgrat professionellen Testens. Sie helfen dabei, realistische Erwartungen zu setzen, Ressourcen effizient einzusetzen und sowohl technische als auch benutzerzentrierte Qualität sicherzustellen.

Für "Pixel Leap" bedeuten sie: Ein systematischer, risikobasierter Ansatz, der technische Korrektheit und Spielerfahrung gleichermaßen berücksichtigt, frühzeitig beginnt und kontinuierlich an neue Erkenntnisse angepasst wird.

Qualitätsmanagement und Qualitätssicherung

Grundlagen des Qualitätsmanagements

Zum Qualitätsmanagement (QM) gehören alle organisatorischen Tätigkeiten und Maßnahmen zum Leiten und Lenken einer Organisation bezüglich Qualität. Üblicherweise gehören das Festlegen der Qualitätspolitik und der Qualitätsziele, die Qualitätsplanung, die Qualitätssicherung, die Qualitätssteuerung und die Qualitätsverbesserung zu den Aufgaben des Qualitätsmanagements.

Das Qualitätsmanagement ist somit eine Kernaufgabe des Managements. In Branchen wie der Luft- und Raumfahrt, der Automobilindustrie und der Medizintechnik ist ein Qualitätsmanagementsystem vorgeschrieben.

Sehr verbreitet ist die Normenreihe ISO 9000 Quality Management Standards mit dem Standard ISO/IEC 90003 Software Engineering, der die Anwendung der allgemeinen Richtlinien der ISO 9001 auf Computersoftware festlegt.

Qualitätsmanagement in der Softwareentwicklung

Das Qualitätsmanagement (QM) umfasst alle Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität. Hierzu gehört das Festlegen der relevanten Arbeitsprozesse (in der Softwareentwicklung also das Festlegen, nach welchem Vorgehensmodell bzw. Entwicklungsprozess gearbeitet wird).

Die Qualitätssicherung (QS) konzentriert sich dann auf die Einhaltung dieser vereinbarten (Arbeits-)Prozesse und auf die Beurteilung der durch diese Prozesse produzierten Artefakte oder Produkte.

Verantwortung für Qualitätssicherung

Die Qualitätssicherung liegt in der Verantwortung aller Projektbeteiligten. Bei ordnungsgemäßer Befolgung der vorgegebenen Prozesse wird davon ausgegangen, dass die geforderten Qualitätsmerkmale – und somit das Qualitätsniveau – angemessen umgesetzt und dadurch erreicht werden.

Die erstellten Arbeitsergebnisse werden meist eine bessere Qualität aufweisen, was wiederum zur Vermeidung von Fehlerzuständen in den Arbeitsergebnissen und Dokumenten führt.

Qualitätssicherung und Testen

Abgrenzung und Zusammenhang

Der Begriff Qualitätssicherung wird oft in Bezug auf das Testen genutzt oder auch mit dem Testen gleichgesetzt. Qualitätssicherung und Testen sind eng verbunden, aber nicht das Gleiche.

> Die Qualitätssicherung ist darauf ausgerichtet, Vertrauen in die Erfüllung der Qualitätsanforderungen zu erzeugen – dies kann auch mit Testen erfolgen.

Qualitätssteuerung als Bewertungsansatz

Definition und Umfang

Alle Aktivitäten, die der Bewertung der Qualität einer Komponente oder eines Systems dienen, werden unter dem Begriff der Qualitätssteuerung zusammengefasst.

Testen ist neben formalen Methoden (Modellprüfung und Korrektheitsbeweis), Simulation und Prototyping eine der wichtigsten Formen der Qualitätssteuerung.

Produktorientierter Ansatz

Die Qualitätssteuerung – insbesondere das Testen – ist ein produktorientierter, korrigierender Ansatz, der das Erreichen eines angemessenen Qualitätsniveaus nachweist.

Umfassende Qualitätssteuerung

Zu einer effektiven Qualitätssteuerung gehört darüber hinaus die Analyse der Ursachen für Fehler:

- Erkennung: Testen deckt Fehlerzustände auf
- Behebung: Debugging beseitigt die Ursachen
- Verbesserung: Befunde werden in Retrospektiven diskutiert

Verwendung der Testergebnisse

Doppelte Nutzung der Testergebnisse

Ergebnisse des Testens werden sowohl von der Qualitätssicherung als auch von der Qualitätssteuerung verwendet:

Bereich	Verwendung der Testergebnisse	Zweck
Qualitätssteuerung	Behebung von Fehlerzuständen	Direkte Produktverbesserung
Qualitätssicherung	Rückmeldungen über Prozessqualität	Prozessverbesserung und Prävention

Kontinuierliche Verbesserung

In der Qualitätssicherung liefern Testergebnisse Rückmeldungen darüber, wie gut die Entwicklungs- und Testprozesse funktionieren und wo noch Verbesserungspotenzial besteht.

Fazit

Qualitätssicherung und Qualitätssteuerung bilden zusammen ein umfassendes System zur Gewährleistung hoher Softwarequalität. Während die Qualitätssicherung präventiv

auf Prozessebene wirkt, greift die Qualitätssteuerung korrigierend auf Produktebene ein.

Testen spielt in beiden Bereichen eine zentrale Rolle und liefert sowohl konkrete Fehlerbefunde als auch wertvolle Erkenntnisse für die kontinuierliche Verbesserung von Entwicklungsprozessen.

Der Testprozess

Einordnung in die Softwareentwicklung

Softwareentwicklungslebenszyklus-Modelle – kurz Entwicklungsmodelle – dienen dazu, die anfallenden Arbeiten bei der Neu- oder Weiterentwicklung von Softwaresystemen zu strukturieren, zu planen und zu steuern. Als Anleitung, um in einem Softwareprojekt Tests strukturiert durchzuführen, reicht eine Darstellung der Aufgaben, wie sie in den Entwicklungsmodellen zu finden ist, meist nicht aus.

Zusätzlich zur Einordnung des Testens in die gesamte Entwicklung wird ein verfeinerter Ablaufplan für die Testarbeiten selbst benötigt. Das heißt, der »Inhalt« der Entwicklungsaufgabe »Testen« muss in kleinere Arbeitsabschnitte gegliedert werden.

Aufbau eines Testprozesses

Bewährte Testaktivitäten

Es gibt eine Reihe von gebräuchlichen und in der Praxis bewährten Testaktivitäten. Ein geeigneter Testprozess besteht aus solchen Testaktivitäten. Je nach vorgegebener oder vorgefundener Projektsituation ist ein geeigneter, spezifischer Prozess für das Testen der Software daraus zusammenzustellen.

Welche Testaktivitäten in diesem Testprozess enthalten sind, wie sie umgesetzt und wann sie durchgeführt werden, hängt von vielen Faktoren ab und soll individuell in der Teststrategie eines Unternehmens oder eines Projekts festgelegt werden.

> Werden einzelne Testaktivitäten weggelassen, ist die Wahrscheinlichkeit höher, dass das Testen die zu erreichenden Ziele verfehlt.

Hauptaktivitäten im Testprozess

Ein Testprozess wird in der Regel folgende Aktivitäten umfassen:

1. Testplanung
2. Testüberwachung und -steuerung
3. Testanalyse
4. Testentwurf
5. Testrealisierung
6. Testdurchführung
7. Testabschluss

Jede dieser Aktivitäten besteht ihrerseits wieder aus mehreren Einzelaufgaben, die entsprechende Arbeitsergebnisse liefern und je nach Projekt variieren können.

Praktische Umsetzung des Testprozesses

Überlappung und Parallelität

Obgleich die Aufgaben des Testprozesses in sequenzieller Reihenfolge angegeben sind und die einzelnen Aufgaben teilweise logisch aufeinander aufbauen, können und dürfen sie sich in der praktischen Anwendung überschneiden und teilweise auch gleichzeitig durchgeführt werden.

Selbst bei einer angestrebten schrittweisen logischen Abfolge der Testaktivitäten wird es bei einer vorgegebenen sequenziellen Entwicklung sowohl Überlappung, Kombination, Parallelität oder Wegfall von einzelnen Teilen der Aktivitäten geben.

Anpassung an den Projektkontext

Eine Anpassung der Aktivitäten ist in Abhängigkeit des System- und Projektkontexts somit unabhängig vom eingesetzten Entwicklungsmodell meist erforderlich.

Iterativer Testprozess

Agile Entwicklungsansätze

Oft wird Software in kleinen Iterationen erstellt, so gibt es beim agilen Vorgehen »Build & Test«-Iterationen, die kontinuierlich durchgeführt werden. Testaktivitäten finden innerhalb dieses Entwicklungsansatzes somit ebenfalls iterativ und kontinuierlich statt.

Kontinuierliches Testen

In agilen Umgebungen werden die Testprozess-Aktivitäten in kurzen Zyklen wiederholt:

- Jede Iteration enthält alle Testaktivitäten
- Kontinuierliches Testen ermöglicht frühe Fehlererkennung
- Schnelle Rückkopplung zwischen Entwicklung und Qualitätssicherung

Testmanagement und Verantwortlichkeiten

Rolle des Testmanagements

Ein Großteil der Aktivitäten ist vom Testmanagement durchzuführen oder zu verantworten. Dies umfasst:

- Strategische Planung: Entwicklung der Teststrategie
- Operative Steuerung: Testüberwachung und -kontrolle
- Ressourcenmanagement: Personal- und Budgetplanung
- Berichtswesen: Testberichterstattung an Stakeholder

Inhaltliche Planung durch Testkonzept

Das Testkonzept bildet die inhaltliche Grundlage für die Testplanung und definiert:

- Testziele und Erfolgskriterien
- Testansatz und Testverfahren
- Teststufen und Testumgebungen
- Eingangskriterien und Endekriterien

Fazit

Ein strukturierter Testprozess ist essentiell für erfolgreiches Testen. Die sieben Hauptaktivitäten bieten einen bewährten Rahmen, der je nach Projektkontext angepasst werden kann.

Die Flexibilität des Testprozesses ermöglicht sowohl sequenzielle als auch agile Entwicklungsansätze, während die klare Struktur sicherstellt, dass alle wichtigen Aspekte des Testens berücksichtigt werden.

Erfolgreiches Testmanagement koordiniert diese Aktivitäten und stellt sicher, dass der Testprozess optimal an die jeweiligen Projektanforderungen angepasst ist.

Testplanung

Grundlagen der Testplanung

Eine strukturierte Abwicklung einer so umfangreichen Aufgabe wie des Testens kann nicht planlos erfolgen. Mit der Planung der Testarbeiten wird am Anfang des Softwareentwicklungsprojekts begonnen.

Wie bei jeder Planung sind während des Projektverlaufs regelmäßig Überprüfungen der bisherigen Planung und eventuelle Aktualisierungen und Anpassungen an neue Situationen und Rahmenbedingungen vorzunehmen. Die Testplanung ist somit eine wiederkehrende Aktivität und wird über den gesamten Produktlebenszyklus hinweg durchgeführt und gegebenenfalls angepasst.

Das Testkonzept als zentrale Aufgabe

Grundlage und Inhalt

Zentrale Aufgabe der Testplanung ist die Anfertigung eines Testkonzepts auf Grundlage der verfolgten Teststrategie.

> Das Testkonzept ist die detaillierte Beschreibung des angepassten und durchzuführenden Testprozesses.

Zentrale Planungsfragen

Hier ist festzulegen: * Welche Testobjekte zu prüfen sind * Welche Qualitätsmerkmale bzw. Testziele zu erreichen sind * Mit welchen Testaktivitäten diese nachzuweisen sind

Die Testziele sind zu definieren und ein Testansatz (z.B. eine Kombination von Testverfahren) ist festzulegen, mit dem die Ziele unter Berücksichtigung der gegebenen Randbedingungen am besten erreicht werden können.

Umfang des Testkonzepts

Das Testkonzept beschreibt: * Die Vorgehensweise * Die benötigten Ressourcen * Die Zeitplanung (siehe Testzeitplan) der beabsichtigten Tests mit allen dazugehörenden Aktivitäten

Eingangs- und Endekriterien

Definition und Zweck

Bei der Testplanung sind Kriterien festzulegen, wann eine Testaktivität als beendet angesehen und wann eine Testaktivität aufgenommen werden kann. Es sind Eingangs- und Endekriterien festzulegen.

Eingangskriterien (Definition of Ready)

Eingangskriterien (in der agilen Entwicklung als »Definition of Ready« bezeichnet) definieren Vorbedingungen für die Durchführung einer bestimmten Testaktivität.

Bei deren Nichterfüllung ist es wahrscheinlich, dass die Aktivität nicht so wie geplant abgearbeitet werden kann.

Endekriterien (Definition of Done)

Endekriterien (in der agilen Entwicklung »Definition of Done« genannt) verhindern, dass die einzelnen Testaktivitäten vorzeitig abgebrochen oder beendet werden, etwa aus Zeitdruck oder Ressourcenmangel.

Sie verhindern auch, dass einzelne Testaktivitäten zu ausführlich durchgeführt werden.

Risikomanagement in der Testplanung

Risikobetrachtung als Planungsbestandteil

Eigentlich zu jeder Planung – und somit auch bei der Testplanung – gehört die Berücksichtigung von möglichen Risiken, die eintreten können und die somit einer gewissen Vorbereitung bedürfen.

Risikokategorien

Dabei wird zwischen Produkt- und Projektrisiko unterschieden. Alle Risiken werden in einem Verzeichnis aufgelistet.

Sowohl das Risikoverzeichnis als auch die Eingangs- und Endekriterien sind häufig Teile des Testkonzepts.

Testbasis und Verfolgbarkeit

Integration der Testbasis

Das Testkonzept beinhaltet auch Informationen über die Testbasis, die Grundlage für die Testüberlegungen ist.

Sicherstellung der Verfolgbarkeit

Zwischen Testbasis und den anderen Arbeitsergebnissen der Testaktivitäten sind im Testkonzept Informationen zur Verfolgbarkeit zu erstellen.

Damit soll beispielsweise sichergestellt werden, dass leicht zu ermitteln ist, welche Änderungen an der Testbasis sich auf welche Testaktivität auswirken, damit diese dann angepasst oder ergänzt werden kann.

Teststufen und Testkonzepte

Zuordnung zu Teststufen

Im Testkonzept ist ebenfalls festzulegen, auf welcher Teststufe welche Tests auszuführen sind.

Hierarchische Testkonzepte

Für jede Teststufe kann ein eigenes Testkonzept verfasst werden. Ein sogenanntes Mastertestkonzept umfasst dann mehrere Teststufen.

Der Testzeitplan

Definition und Inhalt

Der Testzeitplan beinhaltet eine Aufzählung von Aktivitäten, Aufgaben und/oder Ereignissen des Testprozesses.

Zeitliche Festlegungen

Er dient der zeitlichen Festlegung und enthält für jede geplante Aktivität: * Den geplanten Anfangstermin * Den geplanten Endtermin * Gegenseitige Abhängigkeiten zwischen den Aktivitäten

Zielsetzung

Ziel der Planung ist das Ermitteln und im Projektverlauf die Einhaltung der vereinbarten Termine. Der Testplan kann Teil eines Testkonzepts sein.

Fazit

Eine durchdachte Testplanung ist das Fundament für erfolgreiches Testen. Das Testkonzept als zentrales Planungsdokument definiert nicht nur das “Was” und “Wie” des Testens, sondern auch das “Wann” und “Womit”.

Die Definition klarer Eingangs- und Endekriterien sowie die systematische Risikobetrachtung gewährleisten eine kontrollierte und zielgerichtete Testdurchführung.

Die Verfolgbarkeit zwischen Testbasis und Testaktivitäten stellt sicher, dass Änderungen systematisch verwaltet werden können und die Qualität der Tests langfristig gewährleistet bleibt.

Testüberwachung und Teststeuerung

Definition und Aufgaben

Die Testüberwachung und Teststeuerung umfasst die fortwährende Beobachtung der aktuell durchgeführten Testaktivitäten im Vergleich zur Planung, die Berichterstattung der ermittelten Abweichungen und die Durchführung der notwendigen Aktivitäten, um die geplanten Ziele auch unter den veränderten Situationen erreichen zu können.

Die Aktualisierung der Planung muss auch auf Grundlage der veränderten Situation erfolgen.

Grundlage: Endekriterien

Bewertungsmaßstäbe

Basis für Testüberwachung und -steuerung sind Endekriterien für die jeweilige Testaktivität bzw. Testaufgabe.

Die Bewertung, ob ein Endekriterium für die Testdurchführung innerhalb einer bestimmten Teststufe erreicht ist, kann folgende Aspekte umfassen:

Bewertungskriterien im Detail

Überdeckungsgrade prüfen

Das Erreichen der im Testkonzept definierten Überdeckungsgrade der Überdeckungselemente wird anhand der Testergebnisse und -protokolle geprüft.

Sind die Kriterien erfüllt, kann die Testaktivität beendet werden.

Qualitätsnachweis bewerten

Der Nachweis der geforderten Komponenten- oder Systemqualität wird auf Grundlage der Testergebnisse und -protokolle bewertet.

Ist die geforderte Qualität erreicht, ist keine Fortsetzung der Testaktivität erforderlich.

Risikoabdeckung nachweisen

Ist im Testkonzept vorgesehen, dass Produktrisiken geprüft und eine festgelegte Überdeckung der Risiken nachgewiesen werden soll, ist auch dieses anhand der Testergebnisse und Testprotokolle nachzuweisen.

Maßnahmen bei Nichterreichung der Kriterien

Wenn die geforderten Endekriterien mit den durchgeführten Tests nicht erreicht wurden, sind zusätzliche Tests zu entwerfen und durchzuführen.

Ist dies nicht möglich – aus welchen Gründen auch immer –, ist der Sachverhalt darzulegen und das damit verbundene Risiko zu bewerten.

Berichtswesen und Kommunikation

Testfortschrittsberichte

Der Testfortschritt im Vergleich zur Planung ist den Stakeholdern in Testfortschrittsberichten zur Kenntnis zu geben.

Diese Berichte enthalten neben den Unterschieden zur ursprünglichen Planung auch die oben beschriebenen Informationen bei: * Vorzeitiger Beendigung des Tests * Nichterreichung der vorgesehenen Endekriterien

Testabschlussberichte

Beim Erreichen von Projektmeilensteinen können zusammenfassende Testabschlussberichte angefertigt werden: * Freigabe * Ende der Iteration * Abschluss der Teststufe

Inhalte der Testberichte

Zielgruppenorientierte Informationen

Alle Testberichte sollen relevante Details für die jeweilige Zielgruppe der Berichte enthalten und aktuell über den Testfortschritt informieren sowie die Ergebnisse der Testdurchführung zusammenfassen.

Managementrelevante Informationen

Die Berichte sollen auch Fragen ansprechen bzw. Informationen liefern für das Projektmanagement:

Aspekt	Inhalt
Zeitliche Planung	(Voraussichtlicher) zeitlicher Abschluss der Testaufgaben
Ressourcenverbrauch	Gegenüberstellung der geplanten und tatsächlichen Ressourcennutzung
Aufwandsverwendung	Der verwendete Aufwand im Vergleich zur Planung

Methoden der Überwachung

Datenerfassung und -auswertung

Die Testüberwachung des Fortschritts im Testprozess kann erfolgen durch:

- Berichterstattung der Mitarbeiter: Manuelle Statusmeldungen und Erfahrungsberichte
- Werkzeugbasierte Auswertungen: Automatisierte Erfassung von Zahlenmaterial und Metriken

Kontinuierliche Kontrolle

Die fortwährende Beobachtung ermöglicht: * Rechtzeitige Erkennung von Abweichungen * Proaktive Steuerungsmaßnahmen * Anpassung der Teststrategie bei Bedarf

Fazit

Testüberwachung und Teststeuerung sind essentiell für erfolgreiches Testmanagement. Sie gewährleisten, dass Tests nicht nur plangemäß durchgeführt werden, sondern auch bei unvorhergesehenen Änderungen die gewünschte Qualität erreichen.

Die systematische Berichterstattung schafft Transparenz und ermöglicht allen Stakeholdern fundierte Entscheidungen über den weiteren Projektverlauf.

Durch die kontinuierliche Bewertung anhand klar definierter Endekriterien wird sichergestellt, dass weder Zeit und Ressourcen verschwendet noch wichtige Qualitätsaspekte übersehen werden.

Testanalyse

Definition und Zielsetzung

Bei der Testanalyse geht es darum, zu ermitteln, was genau zu testen ist. Dazu wird die Testbasis dahingehend untersucht, ob die zu verwendenden Dokumente ausreichend detailliert sind und testbare Merkmale bzw. Eigenschaften (Features) enthalten, um daraus Testbedingungen abzuleiten.

Wie umfangreich die Testbedingung geprüft werden soll, wird durch messbare festzulegende Überdeckungskriterien bestimmt. Ebenso sind die damit verbundenen Risiken und Risikostufen zu definieren und zu priorisieren.

Untersuchung der Testbasis

Prüfung von Dokumenten

Zur detaillierten Untersuchung der Testbasis und für die in Betracht gezogene Teststufe können folgende Dokumente bzw. Informationen herangezogen werden:

Anforderungsspezifikationen

Anforderungsspezifikationen, aus denen das gewünschte funktionale und nicht funktionale Komponenten- oder Systemverhalten hervorgeht.

Zu den Anforderungsspezifikationen gehören: * Fachanforderungen * Funktionale Anforderungen * Systemanforderungen * User Stories * Epics * Anwendungsfälle oder ähnliche Arbeitsergebnisse

Zum Beispiel kann die Spezifikation einer Anforderung zu ungenau in der Festlegung des vorausgesagten Ergebnisses bzw. Systemverhaltens sein, sodass sich Testfälle nicht so ohne Weiteres ableiten lassen. Eine Nachbesserung ist erforderlich.

Entwurfs- und Realisierungsinformationen

Entwurfs- und Realisierungsinformationen, aus denen die Komponenten- oder Systemstruktur hervorgeht:

- System- oder Softwarearchitekturdokumente
- Entwurfsspezifikationen
- Aufrufdiagramme
- Modelldiagramme (z.B. UML- oder Entity-Relationship-Diagramme)
- Schnittstellenspezifikationen oder ähnliche Arbeitsergebnisse

Zu analysieren ist beispielsweise, wie leicht Schnittstellen anzusprechen sind (Offenheit der Schnittstellen) und wie gut das Testobjekt in kleinere Einheiten zerlegbar ist, um diese Teile separat testen zu können.

Testobjekt selbst prüfen

Die Komponente oder das System selbst sind zu untersuchen – darunter: * Programmtext * Datenbank-Metadaten und -abfragen * Weitere Schnittstellen

Beispielsweise ist der Programmtext zu untersuchen, ob dieser gut strukturiert und damit sowohl leicht verstehbar ist als auch eine geforderte Codeüberdeckung einfach erreicht und nachgewiesen werden kann.

Berichte zur Risikoanalyse

Berichte zur Risikoanalyse, die funktionale, nicht funktionale und strukturelle Aspekte der Komponente oder des Systems beinhalten, sollen ebenfalls untersucht werden.

Besteht ein hohes Risiko bei Versagen der Software, ist das Testen sehr gründlich und entsprechend umfangreich durchzuführen. Bei unkritischer Software kann das Testen weniger formal durchgeführt werden.

Fehleranalyse in der Testbasis

Bedeutung der Testbasis-Qualität

»Grundlage« für den gesamten Testprozess ist die Testbasis. Ist die Testbasis fehlerhaft, können keine »korrekten« Testbedingungen und damit keine »korrekten« Testfälle abgeleitet werden.

Mögliche Fehlerarten in Dokumenten

Die Testbasis sollte daher zusätzlich im Hinblick auf mögliche Arten von Fehlern analysiert werden:

- Mehrdeutigkeiten: Formulierungen, die unterschiedlich interpretiert werden können
- Lücken oder Auslassungen: Unvollständige Funktionsbeschreibungen
- Inkonsistenzen: Widersprüchliche Aussagen in den Dokumenten
- Ungenauigkeiten: Unpräzise Spezifikationen
- Widersprüche: Sich widersprechende Anforderungen
- Überflüssige Inhalte: Textpassagen ohne neue Informationen

Entdeckte Fehler und Unstimmigkeiten in den Dokumenten sind in Fehlerberichten zu dokumentieren und zu beheben.

Testbarkeit

Definition der Testbarkeit

In dem Zusammenhang wird auch von Testbarkeit gesprochen. Testbarkeit meint zum einen:

- Ob die Testbasis ausreichend präzise formuliert ist, sodass Testbedingungen abgeleitet werden können
- Ob die durchzuführenden Tests nachweisen können, dass diese Testbedingungen erfüllt sind

Zum anderen wird damit ausgedrückt: * Ob oder wie gut das Testobjekt für die Testdurchführung zugänglich ist * Ob das Testobjekt geeignete Schnittstellen besitzt, über die der gewünschte Test oder Testfall tatsächlich ausgeführt werden kann

Frühzeitige Fehlererkennung

Die Identifizierung und Behebung von Fehlern in der Testbasis ist sehr wichtig, besonders wenn die Dokumente vorher keinem Reviewprozess unterzogen wurden.

Bei Vorgehensweisen wie Behavior Driven Development (BDD) und Acceptance Test Driven Development (ATDD) werden vor der Codierung Testbedingungen und Testfälle aus User Stories und Akzeptanzkriterien erstellt. Vorteilhaft ist, dass auf diese Weise Fehler frühzeitig erkannt werden können.

Priorisierung und Verfolgbarkeit

Priorisierung der Testbedingungen

Nachdem die Testbedingungen identifiziert und definiert sind, ist eine Priorisierung der Testbedingungen vorzunehmen.

> Damit soll sichergestellt werden, dass die wichtigen und risikoreichen Testbedingungen zuerst und ausreichend getestet werden.

Es kommt leider in Projekten vor, dass aus zeitlichen Restriktionen nicht alle geplanten Tests auch durchgeführt werden.

Verfolgbarkeit sicherstellen

Bei der Testplanung soll sichergestellt werden, dass eine eindeutige bidirektionale Verfolgbarkeit zwischen der Testbasis und den anderen Arbeitsergebnissen der Testaktivitäten vorhanden ist.

Die Verfolgbarkeit ist hier zu detaillieren: * Es muss klar sein, welche Testbedingung welche Anforderungen prüft * Umgekehrt muss erkennbar sein, welche Anforderungen durch welche Testbedingungen abgedeckt sind

Nur so ist später eine fundierte Aussage zu treffen, welche Anforderungen wie intensiv – mit welchen Testfällen, abgeleitet aus den Testbedingungen – geprüft werden sollen bzw. getestet wurden.

Berücksichtigung von Testverfahren

Systematische Herangehensweise

Bereits während der Testanalyse kann die Beachtung von Testverfahren (Blackbox-, Whitebox-Verfahren und erfahrungsbasiert) sehr hilfreich sein.

Die Verfahren stellen eine gewisse Systematik bereit, die: * Die Wahrscheinlichkeit verringert, Testbedingungen zu übersehen * Hilft, präzisere Testbedingungen zu definieren

Beispiel: Äquivalenzklassentest

So wird beispielsweise beim Äquivalenzklassentest sichergestellt, dass der gesamte Eingabebereich für die Erstellung von Testfällen herangezogen wird. Ein mögliches Übersehen oder Vergessen von negativen Eingaben in der Anforderungsdefinition wird somit verhindert.

Erfahrungsbasierte Verfahren

Werden bei der Durchführung der Tests erfahrungsbasierte Verfahren genutzt, können die Testbedingungen aus der Testanalyse als Testziele in Test-Chartas einfließen.

Test-Chartas können verstanden werden als: * »Test-Aufträge« mit Testzielen und möglichen Ideen für weitere Tests * Skizze oder grobe Landkarte für einen bestimmten explorativen Test

Wenn die Testziele auf die Testbasis rückführbar sind, kann auch bei den erfahrungsbasierten Tests die erreichte Überdeckung, z.B. der Anforderungen, gemessen werden.

Test-Chartas können auch erst im Testentwurf formuliert oder im Rahmen der Aktivitäten dort weiter verfeinert werden.

Fazit

Die Testanalyse ist eine fundamentale Aktivität im Testprozess, die das “Was” des Testens definiert. Sie gewährleistet, dass alle relevanten Aspekte identifiziert und systematisch in Testbedingungen überführt werden.

Die sorgfältige Analyse der Testbasis und die Sicherstellung ihrer Testbarkeit sind entscheidend für die Qualität des gesamten Testvorhabens. Nur auf Basis einer soliden Analyse können effektive und zielgerichtete Tests entwickelt werden.

Testentwurf

Definition und Zielsetzung

Beim Testentwurf geht es darum, festzulegen, wie getestet wird. Im Rahmen des Testentwurfs werden aus den Testbedingungen Testfälle bzw. Zusammenstellungen von Testfällen abgeleitet.

Hierzu werden in aller Regel Testverfahren genutzt, die systematische Ansätze zur Erstellung von Testfällen bereitstellen.

Abstrakte und konkrete Testfälle

Unterscheidung der Abstraktionsebenen

Die Spezifikation der Testfälle kann dabei auf zwei »Ebenen« erfolgen: abstrakte und konkrete Testfälle.

Abstrakte Testfälle

Ein abstrakter Testfall enthält keine konkreten Ein- oder Ausgabewerte für die Eingaben und erwarteten Ergebnisse, es werden logische Operatoren zur Beschreibung verwendet.

Vorteile abstrakter Testfälle: * Über mehrere Testzyklen mit unterschiedlichen konkreten Daten verwendbar * Dokumentieren adäquat den Umfang des jeweiligen Testfalls * Wiederverwendbar und wartungsfreundlich

Konkrete Testfälle

Ein konkreter Testfall beinhaltet konkrete Werte für die Eingaben und die vorausgesagten Ergebnisse.

Da nur konkrete Testfälle auf dem Rechner zur Ausführung gebracht werden können, sind die abstrakten Testfälle zu konkretisieren, d.h., die tatsächlichen Werte für Ein- und Ausgaben müssen festgelegt werden.

Praktisches Beispiel: Rabattberechnung in “Pixel Leap”

Stellen wir uns vor, “Pixel Leap” hat ein In-Game-Shop-System mit automatischen Rabatten basierend auf dem Kaufbetrag:

Rabattregeln: * Kaufpreis $<$ 15,00 € $\rightarrow$ Rabatt = 0% * 15,00 € $\leq$ Kaufpreis $\leq$ 20,00 € $\rightarrow$ Rabatt = 5% * 20,00 € $<$ Kaufpreis $<$ 25,00 € $\rightarrow$ Rabatt = 7% * Kaufpreis $\geq$ 25,00 € $\rightarrow$ Rabatt = 8,5%

Abstrakte Testfälle:

Abstrakter Testfall	1	2	3	4
Eingabewert x (Kaufpreis in €)	$x <$ 15000	15000 $\leq x \leq$ 20000	20000 $< x <$ 25000	$x \geq$ 25000
Vorausgesagtes Ergebnis (Rabatt in %)	0	5	7	8,5

Konkrete Testfälle:

Konkreter Testfall	1	2	3	4
Eingabewert x (Kaufpreis in €)	14500	16500	24750	31800
Vorausgesagtes Ergebnis (Rabatt in €)	0	825	1732,50	2703

Vollständige Testfallspezifikation

Umfassende Beschreibung erforderlich

Testfälle umfassen mehr als nur die Testdaten. Für jeden Testfall muss Folgendes beschrieben werden:

Vorbedingungen definieren

Die Ausgangssituation (Vorbedingung) muss klar beschrieben werden: * Welche Randbedingungen für den Test gelten * Welche Systemzustände vor der Testausführung einzuhalten sind * Welche Daten bereits vorhanden sein müssen

Erwartete Ergebnisse festlegen

Vor der Testdurchführung ist festzulegen, welche Ergebnisse bzw. welches Verhalten erwartet wird:

- Ausgaben: Direkte Systemausgaben
- Datenänderungen: Änderungen an globalen (persistenten) Daten
- Zustandsänderungen: Modifikationen des Systemzustands
- Weitere Veränderungen: Alle sonstigen Reaktionen auf die Testdurchführung

Anforderungen an Testdaten

Demzufolge sind Anforderungen an die Testdaten beim Testentwurf festzulegen.

Testorakel

Bestimmung der erwarteten Ergebnisse

Um die vorauszusagenden und erwarteten Ergebnisse zu bestimmen, sind adäquate Informationsquellen zu verwenden. Dabei gibt es zwei grundlegende Möglichkeiten:

Ableitung aus der Testbasis

Der Sollwert wird aus dem Eingabewert auf Grundlage der Testbasis (Anforderungsdefinition, Spezifikation usw.) des Testobjekts abgeleitet.

Inverse Funktionen nutzen

Gibt es für eine Funktion die entsprechende »Umkehr-« oder inverse Funktion, so kann diese nach dem Test ausgeführt und die Ausgabe mit der ursprünglichen Eingabe des Testfalls verglichen werden.

Ein Beispiel sind die Verschlüsselung und Entschlüsselung von Daten in "Pixel Leap" für sichere Spielstand-Übertragungen.

Weitere Aufgaben beim Testentwurf

Priorisierung und Verfolgbarkeit

Neben der Spezifikation von Testfällen gehören die Priorisierung der Testfälle und die Bereitstellung der Testinfrastruktur zu den Aufgaben beim Testentwurf:

Bei der Testanalyse wurden bereits die Testbedingungen priorisiert. Diese Priorisierung wird für die entworfenen Testfälle übernommen und kann weiter verfeinert werden.

So können bei einem Satz von Testfällen, der eine Testbedingung prüft, für einzelne Testfälle unterschiedliche Prioritäten vergeben werden (Testfälle mit hoher Priorität werden zuerst ausgeführt).

Gleiches gilt für die Verfolgbarkeit der Testbedingungen, diese kann nun auf die Testfälle bzw. Sätze von Testfällen heruntergebrochen werden.

Testinfrastruktur und Testumgebung

Bereitstellung der Infrastruktur

Die Testinfrastruktur ist zu ermitteln und bereitzustellen. Zur Testinfrastruktur gehören alle organisatorischen Elemente, die für die Testdurchführung notwendig sind:

- Testumgebung und erforderliche Testwerkzeuge
- Entsprechend ausgestattete Arbeitsplätze
- Hardware- und Softwareressourcen

Testumgebung definieren

Eine Testumgebung wird benötigt, um das Testobjekt unter Benutzung der spezifizierten Testfälle auf dem Rechner ausführen zu können.

Sie umfasst: * Erforderliche Hardware * Gegebenenfalls benötigte Simulatoren * Softwarewerkzeuge sowie weitere unterstützende Hilfsmittel

Damit es bei der Durchführung der Tests nicht zu zeitlichen Verzögerungen kommt, soll die Testinfrastruktur bereits zu diesem Zeitpunkt so weit wie möglich aufgebaut, integriert und geprüft sein.

Überdeckungselemente

Definition der Abdeckungskriterien

Um Kriterien festzulegen, wann ausreichend getestet ist, werden im Rahmen des Testentwurfs Überdeckungselemente bestimmt.

> Ein Überdeckungselement ist eine Eigenschaft oder eine Kombination von Eigenschaften, die aus einer oder mehreren Testbedingungen unter Verwendung eines Testverfahrens abgeleitet wird.

Beispiele für Überdeckungselemente

Bei verschiedenen Testverfahren ergeben sich unterschiedliche Überdeckungselemente:

- Äquivalenzklassenbildung: Die Äquivalenzklassen sind die Überdeckungselemente
- Entscheidungstabellentest: Die Spalten mit ausführbaren Kombinationen von Bedingungen sind die Überdeckungselemente

Qualitätssicherung im Testentwurf

Identifikation von Fehlern

Wie bei der Testanalyse kann auch beim Testentwurf der nützliche Seiteneffekt – die Identifikation von Fehlern in der Testbasis – eintreten.

Verfeinerung der Testbedingungen

Ebenso können Testbedingungen, die in der Testanalyse definiert wurden, im Testentwurf detaillierter spezifiziert werden.

Fazit

Der Testentwurf transformiert das “Was” der Testanalyse in das “Wie” des Testens. Durch die systematische Ableitung von Testfällen aus Testbedingungen entstehen ausführbare und nachvollziehbare Testspezifikationen.

Die Unterscheidung zwischen abstrakten und konkreten Testfällen ermöglicht sowohl Wiederverwendbarkeit als auch praktische Ausführbarkeit. Die vollständige Spezifikation aller Testfall-Aspekte gewährleistet reproduzierbare und aussagekräftige Testergebnisse.

Die frühzeitige Bereitstellung der Testinfrastruktur und die Definition klarer Überdeckungselemente schaffen die Voraussetzungen für eine effiziente Testdurchführung.

Testrealisierung

Grundlagen der Testvorbereitung

Die Testrealisierung bildet das entscheidende Bindeglied zwischen der konzeptionellen Testplanung und der praktischen Umsetzung. In dieser Phase werden sämtliche erforderlichen Vorbereitungen getroffen, um eine reibungslose und effektive Testdurchführung zu gewährleisten.

> Die Testrealisierung umfasst alle Aktivitäten zur finalen Vorbereitung der Testmittel, damit die entwickelten Testfälle ausführungsbereit sind.

Häufig werden die Arbeitsschritte von Testentwurf und Testrealisierung miteinander verzahnt durchgeführt, um Synergieeffekte zu nutzen und den Entwicklungsprozess zu optimieren.

Infrastruktur und Testumgebung

Aufbau der Testinfrastruktur

Ein zentraler Schwerpunkt liegt auf der detaillierten Umsetzung der Testinfrastruktur. Alle benötigten Testmittel müssen vervollständigt oder bei Bedarf neu erstellt werden.

Die erforderlichen Testrahmen erfordern eine vollständige Programmierung und anschließende Installation in der Testumgebung.

Qualitätssicherung der Testumgebung

Da Fehlerwirkungen auch durch Fehlerzustände in der Testumgebung entstehen können, ist eine sorgfältige Verifikation des korrekten Aufbaus unerlässlich.

Zur Vermeidung von Verzögerungen oder Behinderungen während der Testdurchführung müssen sämtliche zusätzlichen Komponenten korrekt implementiert sein:

- Service-Virtualisierung
- Simulatoren
- Platzhalter
- Treiber
- Weitere Infrastrukturelemente

Integration der Testdaten

Die ordnungsgemäße Überführung sämtlicher Testdaten in die Testumgebung ist zu gewährleisten. Dies ermöglicht eine nahtlose Verwendung der Testfälle ohne nachträgliche Modifikationen oder Ergänzungen während der Ausführungsphase.

Konkretisierung und Strukturierung der Tests

Verfeinerung der Testfälle

Neben der Infrastrukturvorbereitung erfordern die Testfälle eine abschließende Konkretisierung. Parallel dazu ist die Festlegung des Testablaufs erforderlich, wodurch die Testfälle in eine logische und sinnvolle Ausführungsreihenfolge gebracht werden.

Bei der Sequenzierung ist die Priorität der einzelnen Testfälle zu berücksichtigen.

Bildung von Testsuiten

Eine effektive Durchführung erfordert die zweckmäßige Gruppierung der Testfälle zu Testsuiten. Diese Strukturierung ermöglicht sowohl eine effiziente Abarbeitung während eines Testzyklus als auch eine übersichtliche Organisation der Testfälle.

> Eine Testsuite besteht aus mehreren Testfällen in ihrer Ausführungsreihenfolge, wobei die Sequenz so gewählt wird, dass Nachbedingungen eines Testfalls als Vorbedingungen des nachfolgenden Testfalls fungieren können.

Diese Herangehensweise vereinfacht den Testablauf erheblich, da nicht für jeden einzelnen Testfall separat Vor- beziehungsweise Nachbedingungen explizit definiert werden müssen.

Zusätzlich sollte jede Testsuite sämtliche Aufräumaktionen nach der Durchführung enthalten, um eine saubere Ausgangssituation für nachfolgende Tests zu gewährleisten.

Automatisierung und Effizienzsteigerung

Überführung in Testskripte

Testabläufe lassen sich häufig direkt in automatisierte oder manuelle Testskripte transformieren. Diese Automatisierung führt zu einer erheblichen Reduzierung des zeitlichen Aufwands bei der Testdurchführung im Vergleich zu einer ausschließlich manuellen und oft ad-hoc durchgeführten Ausführung.

Testausführungsplanung

Die konkrete Ausgestaltung einer effizienten Ausführung der Testfälle, Testsuiten und Testabläufe wird im Testausführungsplan festgelegt.

Qualitätssicherung und Verfolgbarkeit

Aktualisierung der Verfolgbarkeit

Die Verfolgbarkeit der Testfälle zu den zugrundeliegenden Anforderungen beziehungsweise Testbedingungen erfordert gegebenenfalls eine Überprüfung und Aktualisierung.

Gleichzeitig müssen auch die Testabläufe und Testsuiten in die Verfolgbarkeitsmatrix einbezogen werden, um eine vollständige Transparenz über die Testabdeckung zu gewährleisten.

Fazit

Die Testrealisierung stellt eine kritische Phase im Testprozess dar, in der theoretische Planungen in praktisch ausführbare Teststrukturen überführt werden. Die sorgfältige Vorbereitung der Infrastruktur, die durchdachte Strukturierung der Testfälle und die Gewährleistung der Verfolgbarkeit bilden das Fundament für eine erfolgreiche und effiziente Testdurchführung.

Testdurchführung

Grundlagen der Testausführung

Die Testdurchführung stellt die operative Umsetzung der zuvor entwickelten Testkonzepte dar. Entsprechend dem Testausführungsplan werden die Testfälle in verschiedenen Formen zur Ausführung gebracht.

Die Ausführung kann in unterschiedlichen Modalitäten erfolgen: * Manuelle Durchführung * Automatisierte Abarbeitung * Kontinuierliche Testabläufe * Testsitzungen in Paararbeit

Vorbereitung und Vollständigkeitsprüfung

Systemverifikation vor Testbeginn

Zu Beginn der Testdurchführung empfiehlt sich eine gründliche Überprüfung der Vollständigkeit aller zu testenden Komponenten sowie der eingesetzten Testmittel.

Das Testobjekt wird in der bereitgestellten Testumgebung installiert und auf grundlegende Start- sowie Ablauffähigkeit geprüft. Erst bei problemloser Grundfunktionalität kann mit der eigentlichen Testausführung begonnen werden.

Smoke-Test als Startverfahren

> Empfehlung: Die Testdurchführung sollte stets mit der Überprüfung der Hauptfunktionalitäten des Testobjekts mittels Smoke-Test beginnen.

Zeigen sich bereits in dieser Phase Fehlerwirkungen oder Abweichungen von den erwarteten Ergebnissen, ist eine tiefergehende Testausführung wenig zielführend. Die fehlerhaften Grundfunktionen müssen zunächst korrigiert werden, bevor detailliertere Prüfungen sinnvoll sind.

Protokollierung und Dokumentation

Vollständige Testprotokollierung

> Grundsatz: Tests ohne Protokollierung sind wertlos.

Die Ausführung der Testfälle und Testsuiten – ob manuell oder durch Werkzeugeinsatz entsprechend dem Testausführungsplan – erfordert eine exakte und vollständige Dokumentation. Es ist zu vermerken, welche Testläufe mit welchen Ergebnissen durchgeführt wurden:

- Bestanden
- Fehlgeschlagen
- Blockiert

Zweck der Protokollierung

Die Testprotokolle erfüllen mehrere wesentliche Funktionen:

Nachvollziehbarkeit: Die Testdurchführung muss auch für nicht direkt beteiligte Personen – beispielsweise Kunden – transparent und verständlich sein.

Strategienachweis: Es ist zu belegen, dass die geplante Teststrategie tatsächlich umgesetzt wurde.

Detaillierte Dokumentation: Aus dem Testprotokoll muss hervorgehen, welche Komponenten wann, von wem, mit welcher Intensität und mit welchem Ergebnis geprüft wurden.

Vollständigkeitsnachweis: Ausgelassene geplante Testfälle oder Testabläufe sind entsprechend zu vermerken.

Konfigurationsmanagement und Reproduzierbarkeit

Verwaltung der Testressourcen

> Wichtigkeit: Nachvollziehbarkeit und Reproduzierbarkeit sind kritische Erfolgsfaktoren.

Bei jeder Testdurchführung ist neben dem Testobjekt oder einzelnen Testelementen eine Vielzahl von Informationen und Dokumenten beteiligt:

- Testrahmen
- Eingabedateien
- Testprotokolle
- Weitere unterstützende Materialien

Die zu einem Testfall oder Testlauf gehörenden Informationen und Daten müssen so verwaltet werden, dass eine spätere Wiederholung der Tests mit identischen Daten und Randbedingungen problemlos möglich ist.

Die entsprechenden IDs und Versionen der verwendeten Testmittel sind zu dokumentieren und dem Konfigurationsmanagement zuzuführen.

Fehlerbehandlung und -management

Identifikation von Fehlerwirkungen

> Prüfungserfordernis: Tritt bei der Testausführung eine Differenz zwischen tatsächlichem und vorausgesagtem Ergebnis auf, ist bei der Auswertung der Testprotokolle zu entscheiden, ob tatsächlich eine Fehlerwirkung vorliegt.

Bei der Erkennung einer Fehlerwirkung sind folgende Schritte durchzuführen:

- Entsprechende Dokumentation der Fehlerwirkung
- Erste grobe Prüfung möglicher Ursachen
- Gegebenenfalls Spezifikation und Ausführung ergänzender Testfälle
- Erstellung eines Fehlerberichts über die aufgedeckten Fehlerwirkungen

Fehlerkorrektur und Nachtest

Nach der Korrektur des Fehlerzustands ist zu prüfen, ob die Fehlerwirkung beseitigt wurde und bei der Korrektur keine weiteren Fehlerzustände entstanden sind. Gegebenenfalls sind neue Testfälle zu spezifizieren, die den modifizierten oder neuen Programmcode überprüfen.

Qualitätssicherung bei der Fehlermeldung

> Sorgfaltspflicht: Es ist gewissenhaft zu prüfen, ob der Fehlerzustand tatsächlich im Testobjekt liegt.

Nichts schadet der Glaubwürdigkeit eines Testers mehr als eine gemeldete vermeintliche Fehlerwirkung, deren Ursache jedoch in einem fehlerhaften Testfall liegt. Befürchtungen oder Selbstzensur sollten jedoch nicht dazu führen, dass potenzielle Fehlerwirkungen vorsichtshalber nicht gemeldet werden – dies könnte ebenso fatal sein.

Praktikabilität der Fehlerkorrektur

Idealerweise sollten Fehlerzustände einzeln korrigiert und erneut getestet werden, um unbeabsichtigte gegenseitige Beeinflussungen der Änderungen zu vermeiden. In der Praxis erweist sich dies jedoch als nicht durchführbar.

Bei unabhängigen Testern (nicht den Entwicklern selbst) ist eine separate Korrektur einzelner Fehlerzustände nicht praktikabel. Der Aufwand, jede Fehlerwirkung einzeln dem Entwickler zu melden und erst nach erfolgter Korrektur weiterzutesten, ist nicht gerechtfertigt.

Deshalb werden üblicherweise mehrere Fehlerzustände korrigiert und anschließend mit einem neuen Softwareversionsstand zur erneuten Testung vorgelegt.

Messungen und Überdeckungsanalyse

Erfassung von Metriken

Neben der reinen Protokollierung der Differenzen zwischen erwarteten und tatsächlichen Ergebnissen sind Messungen zur Ermittlung der Überdeckungsgrade der Überdeckungselemente durchzuführen.

Bei Bedarf ist auch eine Protokollierung des Zeitverbrauchs vorzunehmen. Hierzu sind entsprechende Werkzeuge einzusetzen.

Verfolgbarkeit und Ergebnisbewertung

Aktualisierung der Verfolgbarkeitsmatrix

> Kontinuierliche Aufgabe: Bei allen bisherigen Aktivitäten des Testprozesses spielt die bidirektionale Verfolgbarkeit eine wichtige Rolle.

Die Verfolgbarkeit ist zu prüfen und zu aktualisieren, damit zwischen Testbasis, Testbedingungen, Testfällen, Testabläufen und Testergebnissen die jeweiligen Beziehungen aktuell bleiben.

Vollständigkeitsbewertung

Nach Abschluss der Testdurchführung kann mithilfe der Verfolgbarkeit ermittelt werden, ob zu jedem Element der entsprechenden Testbasis ein zugehöriger Testablauf ausgeführt wurde.

So kann beispielsweise festgestellt werden: * Welche Anforderungen alle geplanten und durchgeführten Tests bestanden haben * Welche Anforderungen aufgrund fehlgeschlagener Tests nicht erfolgreich verifiziert werden konnten * Ob Fehlerwirkungen bei der Testausführung aufgetreten sind * Welche Anforderungen noch nicht geprüft wurden, da geplante Tests noch ausstehen

Erfolgsbewertung und Abschluss

> Zielerreichung: Solche Informationen ermöglichen eine definitive Aussage, ob die vereinbarten Überdeckungskriterien erreicht wurden und damit der Test als erfolgreich beendet angesehen werden kann.

Durch die Überdeckungskriterien und die Verfolgbarkeit können die Ergebnisse der Tests so dokumentiert werden, dass sie für die Stakeholder verständlich und direkt nachvollziehbar sind.

Fazit

Die Testdurchführung stellt den operativen Höhepunkt des Testprozesses dar, in dem alle vorangegangenen Planungen und Vorbereitungen in konkrete Testergebnisse münden. Die systematische Protokollierung, gewissenhafte Fehlerbehandlung und kontinuierliche Verfolgbarkeit gewährleisten nicht nur die Qualität der Testausführung, sondern schaffen auch die notwendige Transparenz für alle Stakeholder.

Die Balance zwischen gründlicher Dokumentation und praktikablen Arbeitsabläufen ist dabei entscheidend für den Erfolg der gesamten Testaktivitäten.

Testabschluss

Grundlagen des Testabschlusses

Der Testabschluss bildet die finale Aktivität im Testprozess und dient der systematischen Konsolidierung aller Testerfahrungen und -ressourcen. In dieser Phase werden Daten aus abgeschlossenen Testaktivitäten zusammengetragen, um Erkenntnisse zu gewinnen sowie Testmittel und weitere relevante Informationen zu strukturieren.

> Der Testabschluss konsolidiert alle durchgeführten Testaktivitäten und bereitet sowohl die Übergabe an nachgelagerte Phasen als auch den Transfer von Wissen für zukünftige Projekte vor.

Zeitpunkte für den Testabschluss

Je nach gewähltem Entwicklungsmodell ergeben sich unterschiedliche Meilensteine für den Testabschluss:

Traditionelle Entwicklungsmodelle

- Freigabe eines Softwaresystems - Nach erfolgreichem Abschluss aller Teststufen
- Beendigung eines Testprojekts - Bei planmäßigem Abschluss oder vorzeitigem Projektabbruch
- Abschluss spezifischer Teststufen - Nach Beendigung einzelner Testphasen

Agile Entwicklungsansätze

- Fertigstellung einer Projektiteration - Beispielsweise als integraler Bestandteil einer Retrospektive oder Bewertungssitzung
- Abschluss von Sprint-Testaktivitäten - Nach erfolgreichem Sprint-Review

Wartungszyklen

- Abschluss der Testaktivitäten zu einem Wartungsrelease - Bei Update- oder Patch-Zyklen

Beispiel aus "Pixel Leap": Der Testabschluss für Version 1.2.0-beta erfolgte nach der Community-Beta-Phase und umfasste die Auswertung von 847 Spieler-Feedback-Reports sowie die Vorbereitung der Release-Version 1.2.1.

Vollständigkeitsüberprüfung und Fehlerbehandlung

Sicherstellung der Testaktivitäten-Vollständigkeit

Beim Testabschluss ist zu gewährleisten, dass sämtliche Testaktivitäten ordnungsgemäß beendet und alle Fehlerberichte vollständig sowie abgeschlossen sind.

Behandlung offener Fehlerwirkungen

Nicht behobene Fehlerwirkungen (ungelöste Abweichungen zu bestehenden Anforderungen) bleiben dokumentiert offen und werden in nachfolgende Iterationen oder Releases übertragen.

In agilen Entwicklungsansätzen werden diese als neue Product-Backlog-Elemente für kommende Iterationen aufgenommen.

Umgang mit Änderungswünschen

Change Requests (neue oder modifizierte Anforderungen), die sich während der Testauswertung ergeben, erfordern eine ähnliche Behandlung wie offene Fehlerwirkungen.

Beispiel aus "Pixel Leap": Die Community-Beta-Tests deckten den Wunsch nach einem Farbblindheits-freundlichen Modus auf. Dieser Change Request wurde als hochpriorisiertes Backlog-Element für Version 1.3.0 eingeplant.

Testabschlussbericht und Stakeholder-Kommunikation

Erstellung des Abschlussberichts

Ein Testabschlussbericht ist zu erstellen, der alle Testaktivitäten und -ergebnisse zusammenfasst. Er enthält eine abschließende Bewertung der durchgeführten Tests gegen die festgelegten Endekriterien.

Stakeholder-Bereitstellung

Der Testabschlussbericht wird allen relevanten Stakeholdern zur Verfügung gestellt, um Transparenz über die Testqualität und erreichte Abdeckung zu schaffen.

Beispiel aus "Pixel Leap": Der Testabschlussbericht für die Beta-Phase umfasste eine Executive Summary für das Management, detaillierte Metriken für das Entwicklungsteam und eine benutzerfreundliche Zusammenfassung für das Marketing-Team.

Archivierung und Übergabe der Testmittel

Bedeutung der Testmittel-Archivierung

Softwaresysteme werden üblicherweise über längere Zeiträume eingesetzt. Während dieser Zeit treten Fehlerwirkungen auf, die im ursprünglichen Test nicht identifiziert wurden, oder es entstehen weitere Änderungswünsche seitens der Kunden.

Beide Situationen führen zu Programmüberarbeitungen, und der modifizierte Code erfordert erneute Testdurchführung. Ein erheblicher Teil dieses Wartungstest-Aufwands kann vermieden werden, wenn die Testmittel weiterhin verfügbar bleiben.

Umfang der zu archivierenden Materialien

Die zu übergebenden Testmittel umfassen: * Testfälle und Testsuiten * Testprotokolle und Ausführungsberichte * Testinfrastruktur und Testumgebungen * Eingesetzte Werkzeuge und Testskripte * Testdaten und Konfigurationsdateien

Übergabe an die Wartungsabteilung

Die Testmittel sind an die Wartungsabteilung zu übertragen. In der Wartung kann dann eine Anpassung der bestehenden Testmittel anstatt einer Neuerstellung vorgenommen werden.

Wiederverwendung in ähnlichen Projekten

Die Testmittel lassen sich nach entsprechender Anpassung auch gewinnbringend für Projekte mit ähnlicher Aufgabenstellung verwenden.

Rechtliche und Compliance-Anforderungen

In vielen Branchen ist ein Nachweis über durchgeführte Tests erforderlich, da gesetzliche Bestimmungen dies vorschreiben. Ohne Archivierung aller Testmittel kann dieser Nachweis meist nicht erbracht werden.

Beispiel aus "Pixel Leap": Für die Konsolenzertifizierung mussten alle Testprotokolle der Performance- und Stabilitätstests für zwei Jahre archiviert werden, um Compliance-Audits zu bestehen.

Praktische Archivierungsansätze

> Praktischer Tipp: Da die nachträgliche "Konservierung" der Testmittel sehr aufwendig ist, wird in der Praxis häufig ein "Image" erstellt oder es wird bereits vorab ein Docker-Container erzeugt, der als Testumgebung dient und somit von vornherein wiederverwendbar ist.

Moderne Archivierungsansätze umfassen: * Container-basierte Testumgebungen - Docker-Images mit vorkonfigurierter Infrastruktur * Versionierte Testrepositories - Git-basierte Verwaltung aller Testmittel * Cloud-basierte Testarchive - Skalierbare Speicherlösungen mit einfachem Zugriff * Automatisierte Backup-Pipelines - Kontinuierliche Sicherung kritischer Testressourcen

Erfahrungsauswertung und Prozessverbesserung

Analyse der Testerfahrungen

Die während der vorherigen Testaktivitäten gesammelten Erfahrungen sollen systematisch analysiert werden, um für künftige Projekte zur Verfügung zu stehen. Abweichungen zwischen Planung und Umsetzung einzelner Aktivitäten sind dabei ebenso von Interesse wie die Ermittlung der vermuteten Gründe.

Identifikation von Verbesserungspotenzialen

Die gewonnenen Erkenntnisse sollen genutzt werden, um Verbesserungspotenzial zu erkennen und erforderliche Änderungen einzelner Aktivitäten für zukünftige Iterationen, Produktreleases und Projekte zu veranlassen.

Reifegradsteigerung des Testprozesses

Durch die Umsetzung der identifizierten Verbesserungen wird der durchgeführte Testprozess kontinuierlich an Reife gewinnen.

Strukturierte Lessons-Learned-Erfassung

Typische Bereiche der Erfahrungsauswertung umfassen:

Bereich	Fragestellungen	Maßnahmen
Planungsgenauigkeit	Wurden Zeitschätzungen eingehalten?	Verbesserung der Testschätzung
Werkzeugeinsatz	Welche Tools bewährten sich?	Tool-Standardisierung und -Schulungen
Testverfahren	Welche Testverfahren waren effektiv?	Methodenoptimierung
Kommunikation	Funktionierte die Stakeholder-Kommunikation?	Verbesserung der Testberichterstattung

Beispiel aus "Pixel Leap": Die Erfahrungsauswertung der Beta-Phase zeigte, dass automatisierte Performance-Tests 40% mehr kritische Issues identifizierten als manuelle Tests. Daraufhin wurde die Testautomatisierungsstrategie für das nächste Projekt ausgebaut.

Kontinuierliche Verbesserung und Wissenstransfer

Dokumentation von Best Practices

Erfolgreiche Ansätze und bewährte Praktiken werden dokumentiert und in die organisationsweite Wissensbasis integriert.

Schulung und Kompetenzentwicklung

Identifizierte Kompetenzlücken werden durch gezielte Schulungsmaßnahmen für das Testteam adressiert.

Organisatorische Lernzyklen

Die Erkenntnisse fließen in die kontinuierliche Verbesserung der Teststrategie und Testrichtlinien der Organisation ein.

Fazit

Der Testabschluss ist weit mehr als eine administrative Formalität. Er bildet die Brücke zwischen der aktuellen Testdurchführung und zukünftigen Qualitätssicherungsaktivitäten. Die systematische Archivierung der Testmittel, die strukturierte Auswertung der Testerfahrungen und die daraus abgeleiteten Verbesserungsmaßnahmen tragen entscheidend zur Reifegradentwicklung der gesamten Testorganisation bei.

Nur durch einen sorgfältig durchgeführten Testabschluss kann das in den Testaktivitäten generierte Wissen nachhaltig für die Organisation gesichert und in zukünftigen Projekten gewinnbringend eingesetzt werden.

Verfolgbarkeit

Grundlagen der Verfolgbarkeit im Testprozess

Die Verfolgbarkeit (Traceability) bildet das verbindende Element zwischen Testbasis, Testarbeitsergebnissen und Testmitteln. Sie ermöglicht die Nachvollziehbarkeit von Beziehungen zwischen Testbedingungen, Risiken und Testfällen und stellt somit einen kritischen Erfolgsfaktor für die Qualitätssicherung dar.

> Die Verfolgbarkeit schafft transparente Verbindungen zwischen allen Elementen des Testprozesses und ermöglicht eine fundierte Testüberwachung sowie Teststeuerung.

Beispiel aus "Pixel Leap": Die Verfolgbarkeitsmatrix verknüpfte die Anforderung "Sprungmechanik muss präzise funktionieren" mit den Testbedingungen "Sprungphysik" und "Kollisionserkennung", den entsprechenden Testfällen TC_001 bis TC_005 sowie den Risiken "Gameplay-Breaking-Bugs" und "Spielerfrustration".

Bedeutung für Testüberwachung und -steuerung

Fundamentale Rolle im Testprozess

Die Verfolgbarkeit ist entscheidend für die Implementierung einer effektiven Testüberwachung und Teststeuerung, die sich über den gesamten Testprozess erstrecken.

Beziehungen zwischen jedem Element der Testbasis und den verschiedenen Ergebnissen der Testaktivitäten müssen etabliert und gepflegt werden.

Überdeckungsanalyse

Der Grad der erreichten Überdeckung lässt sich mithilfe der Verfolgbarkeit bestimmen. Beispielsweise kann ermittelt werden, ob alle Anforderungen mit mindestens einem Testfall geprüft wurden.

Diese Erkenntnisse dienen als Grundlage für weitere Teststeuerungsmaßnahmen und ermöglichen fundierte Entscheidungen über den Testfortschritt.

Erweiterte Vorteile der Verfolgbarkeit

Auswirkungsanalyse von Änderungen

Die Verfolgbarkeit ermöglicht eine präzise Auswirkungsanalyse von Änderungen. Es kann systematisch analysiert werden, welche Modifikationen der Anforderungen sich auf welche Testbedingungen, Testabläufe, Testfälle und Testelemente auswirken.

Beispiel aus "Pixel Leap": Als die Spielphysik-Engine von Unity auf Unreal umgestellt wurde, zeigte die Verfolgbarkeitsmatrix sofort, dass 23 von 156 Testfällen angepasst werden mussten, da sie direkt mit der Engine-spezifischen Sprungmechanik verknüpft waren.

Risikobewertung und Restrisiko-Analyse

Bei vorhandener Verfolgbarkeit der Testergebnisse zu den identifizierten Risiken kann der Umfang des Restrisikos des Testobjekts fundierter beurteilt werden.

Diese Verknüpfung ermöglicht es, gezielt Bereiche mit hohem Restrisiko zu identifizieren und entsprechende Maßnahmen einzuleiten.

Abstraktion für Stakeholder-Kommunikation

Durch die Verfolgbarkeit können Testergebnisse auf eine abstraktere Ebene "gehoben" werden, wodurch das Testen auch für Nicht-Tester nachvollziehbar wird.

> Wichtige Erkenntnis: Die bloße Betrachtung durchgeführter Testfälle gibt keine Auskunft darüber, wie intensiv und wie "breit" getestet wurde.

Erst mit den Informationen aus der Verfolgbarkeit lassen sich für jede interessierte Personengruppe verständliche Aussagen treffen. Auch kann die Erfüllung von IT-Governance-Kriterien auf einem abstrakten Niveau nachgewiesen werden.

Verbesserung der Testberichterstattung

Testfortschritts- und Abschlussberichte

Testfortschrittsberichte und Testabschlussberichte werden durch die Verfolgbarkeit erheblich verständlicher und aussagekräftiger.

Der Status der Elemente der Testbasis ist in die Berichte aufzunehmen. Zu jeder einzelnen Anforderung kann eine der folgenden Aussagen getroffen werden:

Status	Beschreibung
Tests bestanden	Alle zugehörigen Tests wurden erfolgreich durchgeführt
Tests fehlgeschlagen	Fehlerwirkungen sind aufgetreten
Tests ausstehend	Geplante Tests stehen noch aus

Stakeholder-gerechte Aufbereitung

Technische Aspekte des Testens können mithilfe der Verfolgbarkeit so aufbereitet werden, dass sie für verschiedene Stakeholder verständlich sind.

Die Beurteilung der Produktqualität, der Prozessfähigkeit und des Projektfortschritts anhand von Unternehmenszielen wird somit ermöglicht.

Beispiel aus "Pixel Leap": Für das Management wurde aus der technischen Verfolgbarkeitsmatrix ein Dashboard erstellt, das zeigte: "87% der kritischen Gameplay-Features sind vollständig getestet und freigegeben", anstatt der technischen Details über 247 einzelne Testfälle.

Praktische Umsetzung und Werkzeugunterstützung

Organisatorische Managementsysteme

Um die Vorteile der Verfolgbarkeit vollständig zu nutzen, entwickeln einige Unternehmen eigene Managementsysteme. Diese Systeme organisieren die Arbeitsergebnisse so, dass die benötigten Informationen zur Verfolgbarkeit automatisch bereitgestellt werden.

Kommerzielle Testmanagement-Werkzeuge

Verschiedene am Markt erhältliche Testmanagement-Werkzeuge implementieren Verfolgbarkeitsfunktionen und automatisieren viele der manuellen Schritte.

Moderne Implementierungsansätze

Zeitgemäße Umsetzungen der Verfolgbarkeit umfassen:

- Integrierte ALM-Plattformen - Application Lifecycle Management mit durchgängiger Verfolgbarkeit
- Automatisierte Matrix-Generierung - Dynamische Erstellung von Verfolgbarkeitsmatrizen aus Metadaten
- Real-time Dashboards - Live-Aktualisierung der Verfolgbarkeitsinformationen
- API-basierte Synchronisation - Automatischer Abgleich zwischen verschiedenen Werkzeugen

Beispiel aus "Pixel Leap": Das Entwicklungsteam verwendete Jira für Requirement-Management, verknüpft mit TestRail für Testfälle und Jenkins für Testautomatisierung. Eine API-Integration stellte sicher, dass jede Code-Änderung automatisch mit den betroffenen Tests und Anforderungen verknüpft wurde.

Erfolgsfaktoren für effektive Verfolgbarkeit

Granularitätsebenen

Die Verfolgbarkeit sollte auf angemessenen Granularitätsebenen implementiert werden:

Ebene	Von	Zu	Zweck
Strategisch	Geschäftsziele	Anforderungen	Business-Alignment
Taktisch	Anforderungen	Testfälle	Abdeckungsnachweis
Operativ	Testfälle	Testergebnisse	Ausführungsnachweis

Bidirektionale Verfolgbarkeit

Eine bidirektionale Verfolgbarkeit ermöglicht sowohl Forward- als auch Backward-Tracing:

- Forward-Tracing: Von Anforderungen zu Tests und Ergebnissen
- Backward-Tracing: Von Fehlerwirkungen zurück zu ursprünglichen Anforderungen

Herausforderungen und Lösungsansätze

Wartungsaufwand

Die Pflege der Verfolgbarkeit erfordert kontinuierliche Aufmerksamkeit und kann ressourcenintensiv sein.

Lösungsansätze: * Automatisierung der Matrix-Updates * Integration in bestehende Workflows * Verwendung von Metadaten und Tags

Komplexitätsmanagement

Bei großen Projekten kann die Verfolgbarkeitsmatrix schnell unübersichtlich werden.

Bewältigungsstrategien: * Hierarchische Darstellungen * Filterbare Views für verschiedene Stakeholder * Fokussierung auf kritische Pfade

Fazit

Die Verfolgbarkeit ist kein Selbstzweck, sondern ein strategisches Instrument zur Qualitätssicherung und Risikominimierung. Sie schafft Transparenz, ermöglicht fundierte Entscheidungen und verbessert die Kommunikation zwischen allen Projektbeteiligten.

Eine gut implementierte Verfolgbarkeit transformiert das Testen von einer isolierten Aktivität zu einem integrierten Bestandteil des gesamten Entwicklungsprozesses und trägt maßgeblich zum Projekterfolg bei.

Investitionen in robuste Verfolgbarkeitssysteme zahlen sich durch verbesserte Qualität, reduzierte Risiken und erhöhte Stakeholder-Zufriedenheit langfristig aus.

Kontextuelle Einflüsse auf den Testprozess

Grundverständnis kontextueller Faktoren

Das Testen operiert niemals in einem Vakuum, sondern ist stets in den spezifischen Kontext der jeweiligen Organisation und des Projekts eingebettet. Die Gestaltung und Durchführung von Testaktivitäten wird maßgeblich durch eine Vielzahl kontextueller Faktoren beeinflusst, die eine adaptive und situationsgerechte Herangehensweise erfordern.

> Kernprinzip: Der Testprozess muss an den jeweiligen Kontext angepasst werden, um sowohl die Anforderungen der Stakeholder zu erfüllen als auch die verfügbaren Ressourcen optimal zu nutzen.

Testaktivitäten bilden einen integralen Bestandteil des organisatorischen Entwicklungsprozesses und dienen dem Nachweis, dass die Erwartungen der Stakeholder erfolgreich umgesetzt wurden.

Stakeholder-bezogene Einflussfaktoren

Klarheit und Präzision der Anforderungen

Die Qualität der Testfälle korreliert direkt mit der Klarheit und Präzision der formulierten Stakeholder-Bedürfnisse, Erwartungen und Anforderungen. Je detaillierter und eindeutiger diese in Dokumenten vorliegen, desto zielgerichteter können Testfälle zum Nachweis der korrekten Umsetzung entwickelt werden.

Beispiel aus "Pixel Leap": Die vage Anforderung "Das Spiel soll flüssig laufen" führte zu unklaren Testzielen. Erst die präzise Formulierung "60 FPS konstant bei 1080p auf Mindesthardware" ermöglichte konkrete Performance-Tests mit messbaren Kriterien.

Kollaborationsbereitschaft

Die Bereitschaft der Stakeholder zur Zusammenarbeit mit dem Projektteam, insbesondere mit den Testern, beeinflusst maßgeblich den Erfolg der Testaktivitäten. Diese bidirektionale Kooperationsbereitschaft umfasst sowohl die Offenheit der Stakeholder als auch die Kommunikationsfähigkeiten des Testteams.

Teamkomposition und Kompetenzfaktoren

Personelle Verfügbarkeit und Fähigkeiten

Die Zusammenstellung des Teams wirkt sich unmittelbar auf die Gestaltung des Testprozesses aus. Neben der zeitlichen Verfügbarkeit der Teammitglieder sind deren fachliche Kompetenzen, Domänenwissen und aufgabenspezifische Erfahrungen entscheidende Erfolgsfaktoren.

Zentrale Aspekte der Teamkompetenz: * Technische Expertise in den relevanten Technologien * Domänenverständnis für das zu testende System * Erfahrung mit angewandten Testverfahren * Kommunikations- und Kollaborationsfähigkeiten

Schulungs- und Entwicklungsbedarf

Identifizierte Kompetenzlücken können durch gezielte Schulungsmaßnahmen adressiert werden, um die Effektivität der Testaktivitäten zu steigern und die Qualität der Testergebnisse zu verbessern.

Beispiel aus "Pixel Leap": Als das Team von 2D- auf 3D-Entwicklung umstellte, waren Schulungen in 3D-Performance-Testing und räumlicher Kollisionserkennung erforderlich, um die neuen technischen Herausforderungen zu bewältigen.

Projektbeschränkungen und Ressourcenmanagement

Das Dreieck der Projektbeschränkungen

Jedes Projekt unterliegt fundamentalen Beschränkungen, die den Rahmen für alle Testaktivitäten definieren:

Dimension	Auswirkung auf Tests	Anpassungsstrategien
Umfang	Bestimmt Anzahl und Tiefe der Testfälle	Risikobasierte Priorisierung
Zeit	Definiert Testzyklen und -dauer	Testautomatisierung, Parallelisierung
Budget	Limitiert Werkzeuge und Personal	Effiziente Werkzeugauswahl, Skill-Sharing

Diese Beschränkungen erfordern strategische Entscheidungen über Priorisierung, Automatisierung und Ressourcenallokation.

Organisatorische Rahmenbedingungen

Strukturelle Einflüsse

Die Organisationsstruktur, etablierte Richtlinien und bewährte Praktiken prägen maßgeblich die Auswahl und Durchführung von Testaktivitäten. Diese organisatorischen Faktoren schaffen sowohl Möglichkeiten als auch Beschränkungen für die Testgestaltung.

Einfluss des Entwicklungslebenszyklusmodells

Das gewählte Modell des Softwareentwicklungslebenszyklus hat fundamentalen Einfluss auf die Testorganisation:

Traditionelle Modelle Das V-Modell XT definiert Testaktivitäten (Qualitätssicherung) detailliert und legt deren Einsatzzeitpunkte präzise fest. Dies schafft Klarheit, kann aber weniger Flexibilität bieten.

Agile Modelle Agile Entwicklungsmodelle bieten in der Regel weniger detaillierte Vorschriften für das Testen und einzelne Aktivitäten. Dies ermöglicht größere Anpassungsfähigkeit, erfordert aber mehr Eigenverantwortung des Teams.

Beispiel aus "Pixel Leap": Der Wechsel von Wasserfall zu Scrum erforderte eine Umstellung von großen, monolithischen Testphasen zu kontinuierlichen, integrierten Testaktivitäten in jedem Sprint.

System- und Architektureinflüsse

Auswirkungen der Systemarchitektur

Die gewählte und umgesetzte Systemarchitektur bestimmt direkt die möglichen Aufteilungen in Teilsysteme. Diese architekturellen Entscheidungen haben unmittelbare Auswirkungen auf:

- Verfügbare Teststufen (Komponententest, Komponentenintegrationstest, Systemtest, Systemintegrationstest, Abnahmetest)
- Anwendbare Vorgehensweisen für die Ableitung oder Auswahl von Testfällen (Testverfahren)

Testarten und Qualitätsmerkmale

Die erforderlichen Testarten beeinflussen ebenfalls den Testprozess. Eine Testart umfasst eine Gruppe von Testaktivitäten zur Prüfung zusammenhängender Qualitätsmerkmale.

Beispiel: Der Performance-Test fokussiert sich auf das Verhalten eines Systems unter verschiedenen Lastbedingungen und erfordert spezielle Infrastruktur und Messtechniken.

Werkzeuge und technische Infrastruktur

Verfügbarkeit und Gebrauchstauglichkeit

Werkzeuge spielen eine entscheidende Rolle bei der Testdurchführung. Ihre erfolgreiche Nutzung hängt ab von:

- Verfügbarkeit - Sind die benötigten Tools zugänglich und lizenziert?
- Gebrauchstauglichkeit - Können sie effektiv vom Team eingesetzt werden?
- Prozesskonformität - Fügen sie sich nahtlos in den bestehenden Arbeitsablauf ein?

Beispiel aus "Pixel Leap": Die Einführung eines neuen Automatisierungstools scheiterte zunächst an der Komplexität seiner Konfiguration. Erst nach Vereinfachung der Setup-Prozedur konnte es erfolgreich adoptiert werden.

Risikoprofile und Kritikalitätsfaktoren

Risikoadaptive Testprozesse

Bei hohen Produkt- und Projektrisiken muss der Testprozess entsprechend ausgelegt werden, um möglichst viele Risiken durch gezielte Testfälle zu adressieren und zu minimieren.

Komplexität und Kritikalität

> Herausforderung: Komplexe Systeme und solche, deren möglichst fehlerfreies Funktionieren von großer Wichtigkeit ist und bei denen im Fehlerfall hohe Verluste oder existenzielle Gefährdungen eintreten, sind nicht einfach zu testen.

Der Testprozess muss entsprechend der Komplexität und Kritikalität des zu testenden Systems dimensioniert werden.

Anwendungskontext und Einsatzumgebung

Kontextuelle Differenzierung

Der Kontext, in dem die Software eingesetzt wird, wirkt sich erheblich auf die Testanforderungen aus:

Anwendungstyp	Beispiel	Testintensität	Begründung
Interne Tools	Urlaubsplanungssystem	Moderat	Begrenzte Nutzerbasis, kontrollierbares Umfeld
Kritische Systeme	Industriesteuerung	Intensiv	Hohe Sicherheitsanforderungen, externe Auslieferung
Consumer Software	Mobile App	Hoch	Diverse Endgeräte, unkontrollierbare Umgebungen

Beispiel aus "Pixel Leap": Als Indie-Spiel für PC erforderte es weniger Zertifizierungstests als ein AAA-Konsolenspiel, aber mehr Kompatibilitätstests aufgrund der Hardware-Vielfalt der PC-Gaming-Community.*

Auswirkungen auf testbezogene Aspekte

Strategische Implikationen

Die beschriebenen kontextuellen Faktoren beeinflussen fundamentale testbezogene Aspekte:

- Teststrategie - Grundlegende Herangehensweise und Priorisierung
- Testverfahren - Auswahl geeigneter Methoden und Techniken
- Testautomatisierung - Umfang und Implementierungsansatz
- Überdeckung - Angestrebte Abdeckungsgrade und -kriterien
- Testdokumentation - Detaillierungsgrad und Formalisierung
- Testberichterstattung - Frequenz, Format und Zielgruppen

Adaptive Strategieentwicklung

Eine erfolgreiche Testorganisation entwickelt adaptive Strategien, die flexibel auf veränderte Kontextbedingungen reagieren können, ohne die grundlegenden Qualitätsziele zu kompromittieren.

Fazit

Die Anerkennung und systematische Berücksichtigung kontextueller Einflussfaktoren ist fundamental für den Erfolg von Testaktivitäten. Ein "One-Size-Fits-All"-Ansatz wird der Komplexität und Vielfalt realer Projektsituationen nicht gerecht.

Erfolgreiche Testorganisationen zeichnen sich durch ihre Fähigkeit aus, ihre Prozesse, Methoden und Ressourcen an den spezifischen Kontext anzupassen, ohne dabei die grundlegenden Prinzipien der Qualitätssicherung zu vernachlässigen.

Die kontinuierliche Reflexion und Anpassung des Testprozesses an sich verändernde Kontextbedingungen bildet die Grundlage für nachhaltige Verbesserungen in der Softwarequalität.

Fehlerkultur im Testkontext

Grundlagen einer konstruktiven Fehlerkultur

Menschen neigen dazu, ihre eigenen Fehler nur ungern einzugestehen - eine zutiefst menschliche Eigenschaft, die erheblichen Einfluss auf die Testpraxis hat. Das primäre Ziel des Softwaretestens besteht in der Aufdeckung von Abweichungen gegenüber Spezifikationen oder Kundenerwartungen, wodurch Fehlerwirkungen identifiziert und kommuniziert werden müssen.

> Kernherausforderung: Die Entwicklung einer Organisationskultur, die Fehlerwirkungen als Verbesserungsmöglichkeiten und nicht als persönliche Angriffe wahrnimmt.

Der Umgang mit den dabei entstehenden zwischenmenschlichen und psychologischen Herausforderungen erfordert besondere Aufmerksamkeit und strategische Herangehensweisen.

Paradigmenwechsel: Von destruktiv zu konstruktiv

Überwindung traditioneller Denkweisen

Softwareentwicklung wird häufig als konstruktive Tätigkeit wahrgenommen, während Prüfung und Testen von Dokumenten und Software als destruktive Aktivitäten betrachtet werden. Diese Sichtweise führt zu grundlegenden Einstellungsunterschieden zwischen den beteiligten Personen.

Diese Unterscheidung ist jedoch nicht gerechtfertigt: Das Testen stellt eine äußerst kreative und intellektuell herausfordernde Aufgabe dar, die in hohem Maße zum Projekterfolg und zur Produktqualität beiträgt.

Neudefinition der Testrolle

Tester sind nicht die "Spielverderber" des Entwicklungsprozesses, sondern Qualitätspartner, die durch ihre Arbeit den Wert des Endprodukts steigern.

Beispiel aus "Pixel Leap": Das QA-Team wurde zunächst als "Blocker" wahrgenommen, die den Release verzögerten. Nach einem Paradigmenwechsel wurden sie als "Qualitäts-Champions" anerkannt, die kritische Gameplay-Bugs aufdeckten, bevor sie die Spielerfahrung ruinieren konnten.*

Kommunikation von Fehlerwirkungen

Die Herausforderung der Fehlerkommunikation

Die Mitteilung identifizierter Fehlerwirkungen an Autoren oder das Management erfordert besonderen Takt und Fingerspitzengefühl. Die Art der Kommunikation kann die Zusammenarbeit zwischen Beteiligten - Analysten, Product Ownern, Designern, Entwicklern und Testern - fördern oder negativ beeinflussen.

Das Aufdecken von Fehlerzuständen kann als Kritik am Produkt und seinem Autor interpretiert werden, unabhängig davon, ob es sich um statische oder dynamische Tests handelt.

Typische Kommunikationssituationen

Fehleridentifikation erfolgt in verschiedenen Kontexten:

- Review-Sitzungen bei der Besprechung von Anforderungsdokumenten
- Refinement-Meetings zur Detaillierung und Verfeinerung von User Stories
- Dynamische Testausführung in der operativen Phase

Psychologische Barrieren und kognitive Verzerrungen

Bestätigungsfehler im Entwicklungskontext

Der psychologische "Bestätigungsfehler" (Confirmation Bias) stellt eine zentrale Herausforderung bei der Kommunikation über Fehlerzustände dar. Dieser kognitive Mechanismus verhindert die Akzeptanz von Informationen, die den eigenen aktuellen Überzeugungen widersprechen.

> Entwickler-Dilemma: Entwickler gehen natürlicherweise davon aus, dass ihr Code korrekt ist. Bestätigungsfehler erschweren die Einsicht, dass der eigene Code Fehlerzustände enthalten könnte.

Weitere kognitive Voreingenommenheiten können es für Projektbeteiligte schwierig machen, die durch Tests gewonnenen Informationen zu verstehen oder zu akzeptieren.

Das Phänomen des "Überbringers schlechter Nachrichten"

Eine typisch menschliche Eigenschaft besteht darin, den Überbringer schlechter Nachrichten für deren Inhalt verantwortlich zu machen. Testergebnisse werden oft als schlechte Nachrichten wahrgenommen, obwohl das Gegenteil zutrifft.

> Perspektivwechsel: Erkannte Fehlerwirkungen sind grundsätzlich positiv zu bewerten, da sie behoben werden können und dadurch die Qualität des Testobjekts verbessert wird.

Beispiel aus "Pixel Leap": Ein kritischer Speicher-Leak wurde kurz vor Release entdeckt. Anstatt den Tester zu beschuldigen, feierte das Team die rechtzeitige Entdeckung, die einen katastrophalen Post-Launch-Patch verhinderte.*

Soziale Kompetenz als Erfolgsfaktor

Erforderliche Soft Skills

Tester und Testmanager müssen ausgeprägte soziale Kompetenzen mitbringen, um Konflikten vorzubeugen und das Konfliktpotenzial zu reduzieren. Diese Fähigkeiten ermöglichen eine effektive Kommunikation über Fehlerzustände, Fehlerwirkungen, Testergebnisse, Testfortschritte und Risiken.

Ein gegenseitig respektvoller Umgang führt zu einem positiven Arbeitsklima und zu konstruktiven Beziehungen zwischen allen Projektbeteiligten.

Strategien für positive Kommunikation

Erfolgreiche Fehlerkultur basiert auf konstruktiven Kommunikationsstrategien:

Prinzip	Umsetzung	Beispiel
Gemeinsame Ziele betonen	Fokus auf Qualitätsziele	"Wir alle wollen ein stabiles Produkt ausliefern"
Positive Effekte hervorheben	Fehlerwirkungen als Verbesserungschancen	"Diese Entdeckung verhindert Produktionsprobleme"
Lernorientierung fördern	Fehleranalyse als Kompetenzentwicklung	"Diese Erkenntnis hilft bei zukünftigen Projekten"
Managementnutzen kommunizieren	Zeit- und Kosteneinsparungen betonen	"Frühe Fehlererkennung reduziert Nacharbeitskosten"

Praktische Kommunikationsstrategien

Deeskalation und Konfliktprävention

Konfrontative Kommunikation ("laute Töne" oder Streit) ist niemals förderlich für eine produktive Zusammenarbeit. Bewährte Deeskalationsstrategien umfassen:

- Gemeinsame Ziele in Erinnerung rufen - Die angestrebte hohe Qualität des Systems betonen
- Positive Effekte wiederholt kommunizieren - Nutzen der Fehlererkennung und -korrektur hervorheben
- Kompetenzentwicklung unterstützen - Informationen über Fehlerzustände als Lernchancen präsentieren

Zielgruppenspezifische Kommunikation

Die Kommunikation von Testergebnissen muss an die jeweilige Zielgruppe angepasst werden:

Entwickler-Ebene

- Technische Details und Reproduzierbarkeit
- Lernaspekte und Vermeidungsstrategien
- Konstruktive Lösungsvorschläge

Management-Ebene

- Zeit- und Kosteneinsparungen durch frühzeitige Fehlererkennung
- Risikominimierung bei schlechter Produktqualität
- ROI der Qualitätssicherungsmaßnahmen

Beispiel aus "Pixel Leap": Performance-Issues wurden dem Entwicklungsteam mit konkreten Profiling-Daten kommuniziert, während dem Management die Auswirkungen auf die Spielerretention und potenzielle Review-Scores vermittelt wurden.*

Dokumentation und objektive Berichterstattung

Neutrale und faktenorientierte Dokumentation

Die Art der Dokumentation spielt eine entscheidende Rolle bei der Kommunikation. Testergebnisse und andere Erkenntnisse müssen neutral und faktenorientiert dokumentiert werden.

Grundprinzipien objektiver Fehlerberichterstattung: * Personenbezogene Kritik vermeiden - Fokus auf das Problem, nicht die Person * Objektive Fehlerberichte erstellen - Tatsachenbasierte Beschreibungen * Sachliche Review-Befunde - Konstruktive Verbesserungsvorschläge

Empathische Kommunikation

Ein "Rollentausch" hilft dabei, die Perspektive des Gegenübers zu verstehen. Gründe für negative Reaktionen können so eher nachvollzogen und berücksichtigt werden.

Vermeidung von Missverständnissen: * Aktives Nachfragen - "Wie haben Sie das verstanden?" * Bestätigung suchen - "Habe ich Sie richtig verstanden?" * Klarstellung anbieten - "Was ich gemeint habe, ist..."

Zielorientierung und Voreingenommenheit

Klare Definition von Testzielen

Die explizite Definition von Testzielen hat auch psychologische Auswirkungen. Menschen richten ihre Pläne und ihr Verhalten gerne an definierten Zielen aus.

Neben den Testzielen können weitere Ziele vom eigenen Team, Management und anderen Stakeholdern vorgegeben werden.

Objektivität und minimale Voreingenommenheit

Tester müssen mit minimaler persönlicher Voreingenommenheit diese Ziele verfolgen und konsequent daran festhalten. Dies erfordert:

- Bewusstsein für eigene Bias - Reflexion persönlicher Präferenzen
- Faktenbasierte Bewertung - Objektive Kriterien anwenden
- Stakeholder-Balance - Verschiedene Interessen berücksichtigen

Aufbau einer reifen Fehlerkultur

Organisatorische Transformation

Die Entwicklung einer gesunden Fehlerkultur ist ein organisatorischer Transformationsprozess, der folgende Elemente umfasst:

Kulturwandel-Initiativen

- Leadership-Commitment - Führung muss Fehlerkultur vorleben
- Schulungs- und Sensibilisierungsmaßnahmen - Team-Entwicklung und Workshops
- Anreizsystem-Anpassung - Belohnung für frühzeitige Fehlererkennung

Prozessintegration

- Retrospektiven und Lessons Learned - Systematische Reflektion
- Blameless Postmortems - Ursachenanalyse ohne Schuldzuweisungen
- Kontinuierliche Verbesserung - Iterative Prozessoptimierung

Beispiel aus "Pixel Leap": Das Unternehmen führte "Failure Parties" ein - Meetings, in denen besonders lehrreiche Bugs gefeiert und analysiert wurden, was zu einer 40%igen Steigerung der Fehlerberichterstattung führte.*

Messbare Erfolgsindikatoren

Eine reife Fehlerkultur lässt sich an konkreten Metriken messen:

Indikator	Messgröße	Zielrichtung
Meldebereitschaft	Anzahl gemeldeter Fehlerwirkungen	Steigerung
Frühzeitige Erkennung	Fehlerfindungsanteil in frühen Phasen	Steigerung
Teamkommunikation	Feedback-Qualität in Reviews	Verbesserung
Konfliktreduktion	Eskalationen bei Fehlermeldungen	Reduzierung

Fazit

Eine konstruktive Fehlerkultur ist kein Luxus, sondern eine Notwendigkeit für erfolgreiche Softwareentwicklung. Sie transformiert das Testen von einer potenziell konfrontativen Aktivität zu einem kollaborativen Qualitätsprozess.

Die Investition in soziale Kompetenzen, kommunikative Fähigkeiten und organisatorische Veränderungen zahlt sich durch verbesserte Teamdynamik, höhere Produktqualität und reduzierte Projektrisiken aus.

Erfolgreiche Organisationen erkennen, dass Fehlerwirkungen nicht das Problem sind - sondern die Art, wie sie kommuniziert, verstanden und behoben werden. Eine reife Fehlerkultur macht den Unterschied zwischen Teams, die unter dem Druck von Qualitätsproblemen leiden, und solchen, die sie als Wachstumschancen nutzen.

Komplementäre Denkweisen in der Softwareentwicklung

Divergierende Perspektiven als Qualitätsfaktor

Softwaredesigner, Programmierer und Tester operieren mit fundamental unterschiedlichen Denkweisen, da sie in ihren jeweiligen Rollen verschiedene Ziele verfolgen. Designer konzipieren Produkte, Programmierer realisieren Implementierungen, während Tester das gelieferte Resultat verifizieren und validieren sowie dabei Fehlerwirkungen und Fehlerzustände identifizieren.

> Qualitätsprinzip: Eine höhere Produktqualität wird durch die strategische Kombination und Integration dieser komplementären Denkweisen erreicht.

Die bewusste Nutzung dieser Diversität in den Herangehensweisen stellt einen kritischen Erfolgsfaktor für die Entwicklung hochwertiger Software dar.

Kompetenzprofil erfolgreicher Tester

Charakteristische Denkweisen

Die Denkweise einer Person spiegelt ihre Annahmen und bevorzugten Methoden für Entscheidungsfindung und Problemlösung wider. Für Teammitglieder, die in der Testerrolle erfolgreich agieren wollen, sind spezifische Denkweisen und Kompetenzen erforderlich.

Kernkompetenzen und Eigenschaften

Das ideale Kompetenzprofil eines Testers umfasst eine ausgewogene Kombination verschiedener Fähigkeiten:

Grundlegende Charakteristika

- Ausgeprägte Neugier - Intrinsische Motivation zur Erkundung und Hinterfragung
- Professioneller Pessimismus - Konstruktiv-kritische Grundhaltung
- Kritischer Blick - Systematische Infragestellung des Testobjekts

Methodische Fähigkeiten

- Gründlichkeit und Sorgfalt - Detailgenauigkeit bei der Untersuchung
- Methodisches Vorgehen - Strukturierte Herangehensweise an komplexe Probleme
- Analytisches Denken - Systematische Zerlegung und Bewertung von Sachverhalten

Kreative und soziale Kompetenzen

- Kreativität - Entwicklung innovativer Testansätze und -szenarien
- Positive Kommunikation - Konstruktive Interaktion mit Projektbeteiligten
- Kollegiale Beziehungen - Förderung der Teamdynamik und Zusammenarbeit

Beispiel aus "Pixel Leap": Ein Tester entdeckte einen kritischen Bug, indem er systematisch unkonventionelle Eingabekombinationen ausprobierte, die normale Spieler niemals verwenden würden - eine Kombination aus methodischem Vorgehen und kreativer Problemlösung.*

Domänen- und Technikkompetenz

Fachbereichswissen Kenntnisse in der Anwendungsdomäne ermöglichen effektive Kommunikation mit Endanwendern und Fachbereichsvertretern. Dieses Wissen unterstützt das Verständnis und die Nachvollziehbarkeit von Nutzerwünschen und -erwartungen.

Technische Expertise Zur Steigerung der Testeffizienz sind fundierte technische Kenntnisse unerlässlich. Diese umfassen: * Testverfahren-Auswahl - Situationsgerechte Methodenwahl * Werkzeugkompetenz - Effizienter Einsatz geeigneter Testwerkzeuge * Systemverständnis - Architektur- und Implementierungskenntnisse

Teamorientierung als Schlüsselkompetenz

> Zentrale Erkenntnis: Eine der wichtigsten Kompetenzen eines Testers ist die Fähigkeit, effektiv die Teamarbeit zu unterstützen und damit einen erfolgreichen Beitrag zu den Teamzielen zu leisten.

Die Denkweisen und Kompetenzen erfahrener Tester entwickeln sich über Jahre hinweg und reifen durch praktische Erfahrung und kontinuierliche Reflexion.

Entwicklerpsychologie und kognitive Beschränkungen

Lösungsorientierte Denkweise

Entwickler sind primär daran interessiert, Lösungen zu entwerfen und zu realisieren, wobei sie nicht bevorzugt über potenzielle Schwächen ihrer Lösungen reflektieren. Diese lösungsorientierte Herangehensweise kann jedoch durch Elemente der Tester-Denkweise ergänzt werden.

Bestätigungsfehler bei Entwicklern

Der Bestätigungsfehler (Confirmation Bias) erschwert es Entwicklern zusätzlich, Fehlerzustände in ihren eigenen Arbeitsergebnissen zu identifizieren. Diese kognitive Verzerrung verstärkt die natürliche Tendenz zur Selbstbestätigung.

Das Dilemma der Selbsttestung

Grundsätzliche Herausforderung

Die fundamentale Frage "Kann der Entwickler seine Denkweise ändern und sein eigenes Programm testen?" besitzt keine universell gültige Antwort. Wenn der Tester gleichzeitig der Entwickler des Programms ist, muss er seine eigene Arbeit kritisch prüfen.

Psychologische Barrieren der Selbstprüfung

> Menschliche Natur: Nur wenige Menschen sind in der Lage, den für objektive Selbstprüfung notwendigen Abstand zum selbst erschaffenen Produkt herzustellen. Die Neigung, eigene Fehler zu übersehen oder zu rationalisieren, ist tief verwurzelt.

Es besteht ein natürliches Interesse daran, möglichst keine Fehlerzustände im eigenen Programmcode zu finden, da dies das Selbstbild und die wahrgenommene Kompetenz bedroht.

Schwächen von Entwicklertests

Die große Schwäche sogenannter Entwicklertests liegt in der optimistischen Grundhaltung gegenüber dem eigenen Werk. Entwickler, die ihre selbst programmierten Komponenten testen müssen, unterliegen mehreren Risikofaktoren:

Risikofaktor	Manifestation	Auswirkung
Übermäßiger Optimismus	Vernachlässigung kritischer Szenarien	Unvollständige Testabdeckung
Interessenskonflikt	Oberflächliche Testdurchführung	Reduzierte Effektivität
Konzeptuelle Blindheit	Übersehen von Designfehlern	Fundamentale Probleme bleiben unentdeckt

Beispiel aus "Pixel Leap": Ein Entwickler testete seine Sprungmechanik nur mit "normalen" Spielereingaben und übersah dabei, dass extreme Eingabesequenzen zu Speicherüberläufen führten - ein Problem, das erst durch einen unabhängigen Tester entdeckt wurde.*

Blindheit gegenüber systemischen Fehlern

Bei grundsätzlichen Designfehlern - beispielsweise aufgrund von Missverständnissen der Aufgabenstellung - können Entwickler durch ihre eigenen Tests keine Aufklärung erzielen. Die erforderlichen Testfälle kommen ihnen schlichtweg nicht in den Sinn, da sie dieselben konzeptuellen Grundannahmen teilen.

Lösungsansätze für die Selbsttest-Problematik

Kollaborative Testansätze

Eine bewährte Methode zur Verringerung der "Blindheit gegenüber eigenen Fehlhandlungen" ist die paarweise Zusammenarbeit. Testen in Paaren ("Buddy Testing") ermöglicht es, dass Entwicklerkollegen die Programme jeweils gegenseitig testen.

Managementabwägung: Effizienz vs. Objektivität

Das Management muss strategische Entscheidungen bezüglich des optimalen Gleichgewichts zwischen verschiedenen Faktoren treffen:

Vorteile der Selbsttestung

- Zeitersparnis - Keine Einarbeitungszeit erforderlich
- Tiefes Systemverständnis - Detaillierte Kenntnisse der Implementierung
- Direkte Verfügbarkeit - Unmittelbare Testdurchführung möglich

Nachteile der Selbsttestung

- Kognitive Blindheit - Übersehen eigener Fehlhandlungen
- Bestätigungsfehler - Unbewusste Bestätigung eigener Annahmen
- Reduzierte Objektivität - Emotionale Bindung an die eigene Arbeit

Risikobasierte Entscheidungsfindung

Die Entscheidung zwischen Selbst- und Fremdtestung muss in Abhängigkeit von der Kritikalität des Testobjekts und dem damit verbundenen Risiko im Fehlerfall getroffen werden.

Beispiel aus "Pixel Leap": Für kritische Systeme wie die Speicher-Engine wurde unabhängiges Testen mandatorisch, während einfache UI-Komponenten durch Entwickler-Pairing getestet werden konnten.*

Wissenstransfer und interdisziplinäre Kompetenz

Bidirektionale Lernbeziehungen

Um Testfälle zu erstellen, muss sich der Tester das benötigte Wissen über das Testobjekt aneignen, was Zeit erfordert. Gleichzeitig bringt er jedoch vertieftes Test-Know-how mit, das Entwickler oft nicht besitzen oder sich erst aneignen müssen.

Förderung der Zusammenarbeit durch gegenseitiges Verständnis

Die Zusammenarbeit zwischen Testern und Entwicklern wird durch gegenseitige Kenntnis der jeweiligen Aufgabenbereiche erheblich gefördert:

Entwickler-Testkompetenzen

- Grundlagen des Testens - Verständnis für Testprinzipien und -methoden
- Qualitätsmerkmale - Bewusstsein für verschiedene Qualitätsdimensionen
- Testverfahren - Grundkenntnisse systematischer Testansätze

Tester-Entwicklungskompetenzen

- Softwarearchitektur - Verständnis für Systemaufbau und -design
- Programmiergrundlagen - Nachvollziehbarkeit von Implementierungsentscheidungen
- Entwicklungsprozesse - Einblick in Entwicklungszyklen und -herausforderungen

Agile Integration: Der Whole-Team-Ansatz

In der agilen Vorgehensweise wird dieser interdisziplinäre Ansatz systematisch umgesetzt. Der "Whole-Team-Ansatz" integriert verschiedene Rollen und Kompetenzen in einem kohärenten Arbeitsmodell.

Kompetenzentwicklung und Organisationslernen

Strategische Kompetenzplanung

Organisationen sollten bewusst in die Entwicklung komplementärer Denkweisen und Kompetenzen investieren:

Entwicklungsbereich	Zielgruppe	Methoden	Nutzen
Test-Mindset	Entwickler	Workshops, Shadowing	Verbesserte Selbsttestung
System-Thinking	Tester	Architektur-Schulungen	Effizientere Teststrategien
Cross-Training	Alle Rollen	Rotation, Mentoring	Bessere Zusammenarbeit

Kulturwandel und Mindset-Entwicklung

Die Förderung komplementärer Denkweisen erfordert einen systematischen Kulturwandel, der über reine Wissensvermittlung hinausgeht. Es geht um die Entwicklung einer Grundhaltung, die Diversität in Herangehensweisen als Stärke und nicht als Hindernis betrachtet.

Beispiel aus "Pixel Leap": Regelmäßige "Perspective Swaps" - Sitzungen, in denen Entwickler und Tester ihre Rollen tauschten - führten zu einem 35%igen Rückgang kritischer Bugs und verbesserten Teamkommunikation.*

Fazit

Die bewusste Kultivierung und Integration verschiedener Denkweisen in Softwareentwicklungsteams ist kein Luxus, sondern eine strategische Notwendigkeit. Die natürlichen kognitiven Beschränkungen und Verzerrungen einzelner Rollen können nur durch systematische Komplementarität überwunden werden.

Erfolgreiche Organisationen erkennen, dass die vermeintlichen "Gegensätze" zwischen konstruktivem Entwickeln und kritischem Testen in Wahrheit synergetische Kräfte darstellen. Die Herausforderung liegt nicht darin, diese Unterschiede zu eliminieren, sondern sie produktiv zu orchestrieren.

Investitionen in interdisziplinäre Kompetenzentwicklung, gegenseitiges Verständnis und kollaborative Arbeitsformen zahlen sich durch höhere Produktqualität, reduzierte Entwicklungszyklen und verbesserte Teamdynamik nachhaltig aus.

Glossar - Softwaretesting

Inhaltsverzeichnis

A

Abnahmetest

Eine Teststufe mit dem Schwerpunkt zu bestimmen, ob ein System abgenommen werden kann.

Abnahmetestgetriebene Entwicklung

Ein auf Zusammenarbeit basierender Test-First-Ansatz, der Abnahmetests in der Fachsprache der Stakeholder definiert.

Agile Softwareentwicklung

Eine auf iterativer und inkrementeller Entwicklung basierende Gruppe von Softwareentwicklungsmethoden, wobei sich Anforderungen und Lösungen durch die Zusammenarbeit von selbstorganisierenden funktionsübergreifenden Teams entwickeln.

Akzeptanzkriterien

Diejenigen Kriterien, die eine Komponente oder ein System erfüllen muss, um durch den Benutzer, Kunden oder eine bevollmächtigte Instanz abgenommen zu werden.

Alpha-Test

Eine Art Abnahmetest, der in der Testumgebung des Herstellers durch Akteure außerhalb der Herstellerorganisation durchgeführt wird.

Anforderung

Eine Vorschrift die zu erfüllende Kriterien enthält.

Anomalie

Ein Zustand, der von der Erwartung abweicht.

Anweisungsüberdeckung

Die Überdeckung von ausführbaren Anweisungen.

API-Test

Ein Testansatz, der durch das Übermitteln von Anfragen an ein Testobjekt über dessen Programmierschnittstelle ausgeführt wird.

Äquivalenzklasse

Eine Teilmenge des Wertebereichs einer Variablen innerhalb einer Komponente oder eines Systems, für die aufgrund der Spezifikation erwartet wird, dass alle Werte gleichartig behandelt werden.

Äquivalenzklassenbildung

Ein Black-Box-Testverfahren, bei dem die Testbedingungen Äquivalenzklassen sind, und für jede Klasse ein repräsentatives Element ausgeführt wird.

Audit

Die unabhängige Überprüfung eines Arbeitsergebnisses oder Prozesses, durch eine dritte Partei, um die Übereinstimmung mit vorgegebenen Kriterien zu bewerten.

Auf Zusammenarbeit basierender Testansatz

Ein Testansatz, der durch Zusammenarbeit zwischen Stakeholdern auf Vermeidung von Fehlerzuständen fokussiert.

Auswirkungsanalyse

Die Ermittlung aller Arbeitsergebnisse, welche durch eine Änderung beeinflusst werden, inklusive einer Abschätzung der erforderlichen Ressourcen, um die Änderung bewerkstelligen zu können.

B

Befund

Ein Ergebnis einer Bewertung, das eine wichtige Fehlerwirkung, ein Problem, oder eine Möglichkeit beschreibt.

Benutzerabnahmetest

Eine Art Abnahmetest, der durchgeführt wird um festzustellen, ob vorgesehene Benutzer das System abnehmen.

Benutzererlebnis

Wahrnehmungen und Reaktionen einer Person, die aus der tatsächlichen und/oder der erwarteten Benutzung eines Softwareproduktes resultieren.

Bestanden

Der Status eines Testergebnisses, wenn erwartetes Ergebnis und Istergebnis übereinstimmen.

Beta-Test

Eine Art Abnahmetest, der an einem zur Testumgebung des Entwicklers externen Standort durch Akteure außerhalb der Herstellerorganisation durchgeführt wird.

Betrieblicher Abnahmetest

Eine Art Abnahmetest, der durchgeführt wird um festzustellen, ob der Betrieb und/oder die Systemadministration ein System abnehmen können.

Black-Box-Test

Testen auf der Grundlage einer Analyse der Spezifikation der Komponente oder des Systems.

Black-Box-Testverfahren

Ein Testverfahren, das auf der Spezifikation einer Komponente oder eines Systems basiert.

Breitband-Delphi

Ein expertenbasiertes Verfahren zur Testschätzung, mit dem Ziel, durch Einbeziehung von Teammitgliedern zu einer möglichst genauen Schätzung zu kommen.

C

Checklistenbasierter Test

Ein erfahrungsbasiertes Testverfahren, bei dem die Testfälle entworfen werden, um Elemente einer Checkliste auszuführen.

Checklistenbasiertes Review

Ein Reviewverfahren, das entlang einer Liste an Fragen oder geforderten Eigenschaften geführt wird.

D

Dashboard

Eine Darstellung der dynamischen Messung der operationalen Leistung von Unternehmen oder Aktivitäten. Dazu werden visuelle Darstellungen der Metriken mittels Zeiger- oder Zählerinstrumenten genutzt, die an das Amaturenbrett eines Autos erinnern, so dass der Effekt von Ereignissen oder Aktivitäten leicht verstanden und zu operationalen Zielen in Beziehung gesetzt werden kann.

Debugging

Der Prozess der Aufdeckung, Analyse und Entfernung der Ursachen von Fehlerwirkungen in einer Komponente oder einem System.

Dynamischer Test

Testen, das die Ausführung des Testelements beinhaltet.

E

Effektivität

Der Umfang in welchem richtige und vollständige Ziele erreicht werden.

Effizienz

Der Grad, zu dem Mittel verwendet werden im Verhältnis zu den erzielten Ergebnissen.

Eingangskriterien

Die Menge an Bedingungen für den offiziellen Start einer bestimmten Aufgabe.

Eintrittswahrscheinlichkeit des Risikos

Die Wahrscheinlichkeit dafür, dass ein Risiko eintritt.

Endekriterien

Die Menge an Bedingungen für den offiziellen Abschluss einer bestimmten Aufgabe.

Entscheidungstabellentest

Ein Black-Box-Testverfahren, bei dem Testfälle im Hinblick auf die Ausführung von Kombinationen der Bedingungen und aus ihnen resultierender Aktionen einer Entscheidungstabelle entworfen werden.

Erfahrungsbasiertes Testverfahren

Ein Testverfahren, das auf der Erfahrung, dem Wissen und der Intuition der Tester basiert.

Erschöpfender Test

Ein Testansatz, bei dem die Testsuite alle Kombinationen von Eingabewerten und Vorbedingungen umfasst.

Erwartetes Ergebnis

Das beobachtbare vorausgesagte Verhalten eines Testelements unter bestimmen Bedingungen, basierend auf seiner Testbasis.

Explorativer Test

Ein Testansatz, bei dem die Tester auf der Grundlage ihres Wissens, der Erkundung des Testobjekts und der Ergebnisse früherer Tests dynamisch Tests entwerfen und durchführen.

F

Feature-getriebene Entwicklung

Ein iterativ inkrementeller Softwareentwicklungsprozess, der mit Blick auf die Funktionalitäten mit Kundenwert (Features) betrieben wird. Feature-getriebene Entwicklung wird meist bei agiler Softwareentwicklung genutzt.

Fehlerangriff

Ein Testverfahren zur Bewertung eines bestimmten Qualitätsmerkmals eines Testobjekts, indem versucht wird, bestimmte Fehlerwirkungen auszulösen.

Fehlerbericht

Die Dokumentation des Auftretens, der Art und des Status eines Fehlerzustands.

Fehlerdichte

Die Anzahl der Fehlerzustände pro Größeneinheit eines Arbeitsergebnisses.

Fehlerfindungsanteil

Die Anzahl der Fehlerzustände, die in einer Teststufe gefunden wurden, dividiert durch die Gesamtzahl der Fehlerzustände, die in dieser Teststufe und danach mit jeglichen Mitteln gefunden wurden.

Fehlermanagement

Der Prozess der Erkennung, Aufzeichnung, Klassifizierung, Untersuchung, Lösung und Schließung von Fehlerzuständen.

Fehlernachtest

Eine Art änderungsbezogenes Testen, das nach der Behebung eines Fehlerzustands durchgeführt wird, um zu bestätigen, dass eine Fehlerwirkung nicht mehr auftritt.

Fehlerschweregrad

Der Grad der Auswirkungen, den ein Fehlerzustand auf Entwicklung oder Betrieb einer Komponente oder eines Systems hat.

Fehlerwirkung

Ein Ereignis in welchem eine Komponente oder ein System eine geforderte Funktion nicht im spezifizierten Rahmen ausführt.

Siehe auch: Fehlerzustand, Fehlhandlung

Fehlerzustand

Eine Unzulänglichkeit oder ein Mangel in einem Arbeitsergebnis, sodass es seine Anforderungen oder Spezifikationen nicht erfüllt.

Siehe auch: Fehlerwirkung, Fehlhandlung

Fehlgeschlagen

Der Status eines Testergebnisses, wenn erwartetes Ergebnis und Istergebnis nicht übereinstimmen.

Fehlhandlung

Eine menschliche Handlung, die zu einem Fehlerzustand führt.

Siehe auch: Fehlerzustand, Fehlerwirkung

Formales Review

Ein Review, das einem definierten Prozess folgt und ein formell dokumentiertes Ergebnis liefert.

Funktionale Angemessenheit

Der Grad, zu dem die Funktionen die Erfüllung spezifizierter Aufgaben und Ziele ermöglichen.

Funktionale Korrektheit

Der Grad, zu dem eine Komponente oder ein System die richtigen Ergebnisse mit der erforderlichen Genauigkeit liefert.

Funktionale Vollständigkeit

Der Grad, zu dem die Menge der Funktionen alle spezifizierten Aufgaben und Benutzerziele abdeckt.

Funktionaler Test

Testen, welches durchgeführt wird, um die Erfüllung der funktionalen Anforderungen durch eine Komponente oder ein System zu bewerten.

G

Gebrauchstauglichkeit

Der Grad, zu dem eine Komponente oder ein System durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um festgelegte Ziele effektiv, effizient und zufriedenstellend zu erreichen.

Gebrauchstauglichkeitslabor

Eine Testeinrichtung, in der eine unaufdringliche Beobachtung der Reaktionen und Erwiderungen der Teilnehmer auf Software stattfindet.

Gebrauchstauglichkeitstest

Testen mit dem Ziel herauszufinden inwieweit das System durch spezifizierte Benutzer in einem bestimmten Kontext mit Effektivität, Effizienz und Zufriedenheit genutzt werden kann.

Grenzwert

Der kleinste oder der größte Wert einer geordneten Äquivalenzklasse.

Grenzwertanalyse

Ein Black-Box-Testverfahren, bei dem die Testfälle unter Nutzung von Grenzwerten entworfen werden.

Grundursache

Die Ursache eines Fehlerzustands. Wenn man sie behebt, dann wird das Vorkommen der Fehlerart reduziert oder eliminiert.

Grundursachenanalyse

Ein Analyseverfahren, das die Grundursache eines Fehlerzustands identifizieren soll.

Gutachter

Ein Teilnehmer eines Reviews, der Fehlerzustände im Arbeitsprodukt identifiziert.

H

Heuristik

Eine allgemein anerkannte Faustregel, die dabei hilft, ein Ziel zu erreichen.

I

Informelles Review

Ein Review, das keinem definierten Prozess folgt und kein formell dokumentiertes Ergebnis liefert.

Inkrementelles Entwicklungsmodell

Eine Art von Softwareentwicklungslebenszyklusmodell, bei dem die Komponente oder das System über eine Serie von Inkrementen entwickelt wird.

Inspektion

Eine formale Reviewart, die festgelegte Teamrollen und Messungen verwendet, um Fehlerzustände in einem Arbeitsergebnis zu identifizieren und den Reviewprozess sowie den Softwareentwicklungsprozess zu verbessern.

Integrationstest

Eine Teststufe mit dem Schwerpunkt auf dem Zusammenwirken zwischen Komponenten oder Systemen.

Integrität

Der Grad, zu dem nur autorisierter Zugriff und Änderungen an einer Komponente, einem System oder Daten erlaubt sind.

Intuitive Testfallermittlung

Ein Testverfahren, bei dem Tests auf Basis des Wissens der Tester über frühere Fehlerwirkungen oder auf Basis von allgemeinem Wissen über Fehlerauswirkungen abgeleitet werden.

Istergebnis

Im Test beobachtetes/erzeugtes Verhalten einer Komponente oder eines Systems unter festgelegten Bedingungen.

Iteratives Entwicklungsmodell

Eine Art von Softwareentwicklungslebenszyklusmodell, bei dem die Komponente oder das System durch eine Serie von wiederholten Zyklen entwickelt wird.

K

Kompatibilität

Der Grad, zu dem eine Komponente oder ein System Informationen mit anderen Komponenten oder Systemen austauschen kann, und/oder ihre geforderten Funktionen bei einer gemeinsamen Benutzung einer Hardware- oder Softwareumgebung ausführen kann.

Komponente

Kleinste Einheit eines Systems, die für sich alleine getestet werden kann.

Komponentenintegrationstest

Der Integrationstest von Komponenten.

Komponententest

Eine Teststufe mit dem Schwerpunkt auf einer einzelnen Hardware- oder Softwarekomponente.

Konformität

Die Einhaltung von Normen, Konventionen, gesetzlichen Bestimmungen oder ähnlichen Vorschriften durch ein Arbeitsergebnis.

Kontinuierliche Auslieferung

Ein automatisiertes Softwareentwicklungsverfahren, bei dem Codeänderungen automatisch eingebaut, getestet und für die Freigabe zur Produktion vorbereitet werden.

Kontinuierliche Bereitstellung

Ein automatisiertes Software-Release-Verfahren, bei dem alle Code-Änderungen in der Produktion bereitgestellt werden, nachdem sie alle spezifizierten Tests bestanden haben.

Kontinuierliche Integration

Ein automatisiertes Softwareentwicklungsverfahren, das alle Änderungen zusammenführt, integriert und testet, sobald diese der Versionsverwaltung übergeben werden.

Kontinuierlicher Test

Ein Testansatz, der das Testen früh, häufig und während des gesamten Softwareentwicklungslebenszyklus einbindet und automatisiert, um eine Rückmeldung zu den Geschäftsrisiken eines Software-Releasekandidaten so schnell wie möglich zu erhalten.

Kontrollfluss

Die Abfolge, in der Anweisungen durch einen Geschäftsprozess, eine Komponente oder ein System ausgeführt werden.

M

Mean Time To Failure

Die durchschnittliche Zeitspanne vom Betriebsbeginn bis zu einer Fehlerwirkung einer Komponente oder eines Systems.

Messung

Der Prozess, eine Zahl oder Kategorie einer Einheit zuzuweisen, um ein Attribut dieser Einheit zu beschreiben.

Metrik

Die Mess-Skala und das genutzte Verfahren einer Messung.

Moderator

1) Die Person, die für das Durchführen eines Reviews verantwortlich ist. 2) Die Person, die eine Gebrauchstauglichkeitstestsitzung durchführt.

N

N-Switch-Überdeckung

Die Überdeckung einer Sequenz von (N+1) Zustandsübergängen.

Nachbedingung

Der erwartete Zustand eines Testelements und seiner Umgebung nach der Testdurchführung.

Negativtest

Eine Testart, bei der eine Komponente oder ein System auf eine Weise verwendet wird, die so nicht vorgesehen ist.

Neuronenüberdeckung

Die Überdeckung von aktivierten Neuronen im neuronalen Netz für eine Menge von Tests.

Nicht-funktionaler Test

Testen, welches durchgeführt wird, um die Erfüllung der nicht-funktionalen Anforderungen durch eine Komponente oder ein System zu bewerten.

P

Performanz

Der Grad, zu dem eine Komponente oder ein System Zeit, Ressourcen und Kapazität verbraucht während sie/es seine vorgesehenen Funktionen ausführt.

Planungspoker

Ein konsensbasiertes Schätzverfahren, das hauptsächlich zum Schätzen des Aufwands oder der relativen Größe von User-Storys in der agilen Softwareentwicklung verwendet wird. Es ist eine Variante des Breitband-Delphi-Verfahrens, bei dem das Team einen Stapel an Karten mit vorgegebenen Werten für die Schätzung verwendet.

Platzhalter

Ein Testdouble, das vordefinierte Antworten gibt.

Priorität

Die Stufe der Wichtigkeit, die einem Objekt (z.B. Fehlerzustand) zugeordnet worden ist.

Produktrisiko

Ein Risiko, das die Qualität eines Produktes beeinträchtigt.

Programmierstandard

Ein Qualitätsstandard, der erforderliche Merkmale von Code definiert.

Projektrisiko

Ein Risiko, das den Projekterfolg beeinträchtigt.

Protokollant

Eine Person, die während einer Reviewsitzung Informationen aufzeichnet.

Q

Qualität

Der Grad, zu dem ein Arbeitsergebnis die expliziten und impliziten Bedürfnisse seiner Stakeholder erfüllt.

Qualitätskosten

Die gesamten Kosten, die durch Qualitätssicherungsaktivitäten und durch Fehlerwirkungen entstehen. Sie werden oft in Kosten der Fehlervorbeugung, der -Ermittlung, der internen Fehlerwirkungen und den externen Fehlerwirkungen aufgeteilt.

Qualitätsmerkmal

Eine Kategorie von Merkmalen, die sich auf die Qualität des Arbeitsergebnisses auswirken.

Qualitätsrisiko

Ein Produktrisiko oder ein Projektrisiko, welches das Qualitätsmanagement beeinträchtigt.

Qualitätssicherung

Aktivitäten, die darauf fokussieren, Vertrauen in die Erfüllung der Qualitätsanforderungen zu erzeugen.

Qualitätssteuerung

Aktivitäten, die der Bewertung der Qualität einer Komponente oder eines Systems dienen.

R

Regressionstest

Eine Art änderungsbezogenes Testen um festzustellen, ob in unveränderten Bereichen der Software Fehlerzustände eingebaut oder frelegt wurden.

Regulatorischer Abnahmetest

Eine Art von Abnahmetest, der durchgeführt wird, um die Konformität eines Testobjekts festzustellen.

Reife

(1) Die Fähigkeit einer Organisation, ihre Prozesse (Abläufe) effizient und effektiv zu gestalten. (2) Der Grad, zu dem eine Komponente oder ein System die Anforderungen an die Zuverlässigkeit im Normalbetrieb erfüllt.

Retrospektive

Eine regelmäßige Veranstaltung, bei der Teammitglieder Ergebnisse diskutieren, ihre Arbeitsweisen überprüfen und Verbesserungsmöglichkeiten aufzeigen.

Review

Eine Art statischer Test, bei dem ein Arbeitsergebnis oder -prozess von einer oder mehreren Personen bewertet wird, um Fehlerzustände zu erkennen oder Verbesserungen zu erzielen.

Risiko

Ein Faktor, der zu negativen Konsequenzen in der Zukunft führen könnte, gewöhnlich ausgedrückt durch das Schadensausmaß und die Eintrittswahrscheinlichkeit.

Risikoanalyse

Der Prozess, der die Risikoidentifikation und Risikobewertung umfasst.

Risikobasierter Test

Ein Testvorgehen, bei welchem sich das Management, die Auswahl, die Priorisierung und die Anwendung von Testaktivitäten und Ressourcen an entsprechenden Risikotypen und Risikostufen orientieren.

Risikobewertung

Der Prozess der Begutachtung von identifizierten Risiken und der Festlegung der Risikostufe.

Risikoidentifizierung

Die Ermittlung, Erkennung und Beschreibung von Risiken.

Risikomanagement

Der Prozess zur Behandlung von Risiken.

Risikominderung

Der Prozess, mit dem Entscheidungen getroffen und Schutzmaßnahmen umgesetzt werden, um das Risiko auf eine vorgegebene Stufe zu reduzieren oder um es auf einer Stufe zu halten.

Risikosteuerung

Der Prozess, der die Risikominderung und Risikoüberwachung umfasst.

Risikostufe

Das Maß eines Risikos, definiert durch dessen Schadensausmaß und Eintrittswahrscheinlichkeit.

Risikoüberwachung

Die Tätigkeit, die den Status bekannter Risiken überprüft und an Stakeholder berichtet.

S

Schadensausmaß des Risikos

Der Schaden, der entsteht, wenn ein Risiko eintritt.

Sequenzielles Entwicklungsmodell

Eine Art von Softwareentwicklungslebenszyklusmodell, bei dem ein komplettes System in einer Abfolge von mehreren diskreten und aufeinanderfolgenden Phasen ohne Überlappung entwickelt wird.

Service-Virtualisierung

Ein Verfahren, das die virtuelle Bereitstellung von Diensten ermöglicht, die entfernt bereitgestellt, zugegriffen und verwaltet werden.

Shift-Left

Ein Testansatz zur Ausübung von Test- und Qualitätssicherungsaktivitäten so früh wie möglich im Softwareentwicklungslebenszyklus.

Sicherheit (safety)

Die Fähigkeit eines Systems, unter definierten Bedingungen zu keinem Zustand zu führen, der Menschenleben, Gesundheit, Eigentum oder die Umgebung gefährdet.

Sicherheit (security)

Der Grad, zu dem eine Komponente oder ein System seine Daten und Ressourcen vor unberechtigtem Zugriff oder unberechtigter Nutzung schützt und den ungehinderten Zugriff und die Nutzung für seine berechtigten Benutzer sicherstellt.

Simulator

Eine Komponente oder ein System, die bzw. das sich wie ein gegebenes System verhält oder funktioniert.

Sitzungsbasierter Test

Ein Testansatz, bei dem die Testaktivitäten als Testsitzungen geplant werden.

Skalierbarkeit

Die Fähigkeit bis zu dem eine Komponente oder ein System veränderten Lastbedingungen angepasst werden kann.

Smoke-Test

Eine Testsuite, die die Hauptfunktionalität einer Komponente oder eines Systems überdeckt, um vor Beginn der geplanten Testausführung festzustellen, ob die Komponente oder das System ordnungsgemäß funktioniert.

Softwareentwicklungslebenszyklus

Die Aktivitäten, die in jeder Stufe der Softwareentwicklung durchgeführt werden, sowie ihre logischen und zeitlichen Verknüpfungen miteinander.

Statische Analyse

Der Prozess der Bewertung eines Testobjekts (Komponente oder System) basierend auf seiner Form, seiner Struktur, seines Inhalts oder seiner Dokumentation, ohne es auszuführen.

Statischer Test

Testen, das die Ausführung eines Testelements nicht beinhaltet.

System unter Test

Ein System als Testobjekt.

Systemintegrationstest

Der Integrationstest von Systemen.

Systemtest

Eine Teststufe mit dem Schwerpunkt zu verifizieren, dass ein System als Ganzes die spezifizierten Anforderungen erfüllt.

Szenariobasiertes Review

Ein Reviewverfahren bei dem ein Arbeitsergebnis hinsichtlich der Fähigkeit spezifische Szenarien abzudecken beurteilt werden kann.

T

Technisches Review

Ein formales Review durch technische Experten, die die Qualität eines Arbeitsergebnisses untersuchen und Abweichungen von Spezifikationen und Standards feststellen.

Test

Eine Menge von einem oder mehreren Testfällen.

Test in Paaren

Ein Testansatz, bei dem zwei Teammitglieder gleichzeitig beim Testen eines Arbeitsprodukts zusammenarbeiten.

Test-Charta

Die Dokumentation eines Ziels und der Agenda einer Testsitzung.

Test-First-Ansatz

Ein Ansatz zur Softwareentwicklung, bei dem die Testfälle entworfen und implementiert werden, bevor die zugehörige Komponente oder das zugehörige System entwickelt wird.

Testablauf

Eine Folge von Testfällen in der Reihenfolge ihrer Durchführung, mit allen erforderlichen Aktionen zur Herstellung der Vorbedingungen und zum Aufräumen nach der Durchführung.

Testabschluss

Die Aktivität, die Testmittel für eine spätere Anwendung verfügbar macht, Testumgebungen in einem zufriedenstellenden Zustand hinterlässt, und die Testergebnisse an die relevanten Stakeholder übermittelt.

Testabschlussbericht

Eine Art von Testbericht, der beim Erreichen von Abschlussmeilensteinen erstellt wird und eine Beurteilung der entsprechenden Testelemente anhand der Endekriterien liefert.

Testanalyse

Die Aktivität, die Testbedingungen durch eine Analyse der Testbasis identifiziert.

Testansatz

Die Art und Weise der Umsetzung von Testaufgaben.

Testart

Eine Gruppe von Testaktivitäten basierend auf bestimmten Testzielen mit dem Zweck, eine Komponente oder ein System auf spezifische Merkmale zu prüfen.

Testausführungsplan

Ein Zeitplan für die Ausführung von Testsuiten innerhalb eines Testzyklus.

Testautomatisierung

Der Einsatz von Software zur Durchführung oder Unterstützung von Testaktivitäten.

Testautomatisierungsframework

Eine Menge von Testrahmen und Testbibliotheken zur Testautomatisierung.

Testbarkeit

Der Grad, zu dem Testbedingungen für eine Komponente oder ein System festgelegt und Tests durchgeführt werden können, um festzustellen, ob diese Testbedingungen erfüllt sind.

Testbasis

Alle Informationen, die als Grundlage für die Testanalyse und den Testentwurf verwendet werden können.

Testbedingung

Ein testbarer Aspekt einer Komponente oder eines Systems, der als Grundlage für das Testen identifiziert wurde.

Testbericht

Dokumentation, die das Testen und die Ergebnisse zusammenfasst.

Testberichterstattung

Sammlung und Analyse der Daten über Testaktivitäten und ihre anschließende Konsolidierung in einem Bericht, um die Stakeholder zu informieren.

Testdaten

Für die Testdurchführung benötigte Daten.

Testdurchführung

Die Aktivität der Ausführung eines Tests für eine Komponente oder ein System, die Istergebnisse erzeugt.

Testelement

Ein Teil eines Testobjekts, der im Testprozess verwendet wird.

Testen

Der Prozess innerhalb des Softwareentwicklungslebenszyklus, der die Qualität einer Komponente oder eines Systems und der zugehörigen Arbeitsergebnisse bewertet.

Testentwurf

Die Aktivität, die Testfälle aus Testbedingungen ableitet und spezifiziert.

Tester

Eine Person, die das Testen durchführt.

Testergebnis

Das Ergebnis und die Konsequenz der Durchführung eines Tests.

Testfall

Eine Menge von Vorbedingungen, Eingaben, Aktionen (falls anwendbar), erwarteten Ergebnissen und Nachbedingungen, welche auf Basis von Testbedingungen entwickelt wurden.

Testfortschritt

Der Fortschritt des Testens gegenüber einer Baseline.

Testfortschrittsbericht

Ein regelmäßiger Testbericht, der den Fortschritt der Testaktivitäten gegenüber einer Baseline, Risiken und Alternativen, die eine Entscheidung erfordern, enthält.

Testgetriebene Entwicklung

Ein Softwareentwicklungsverfahren, bei dem Testfälle entwickelt und automatisiert werden, und anschließend Software inkrementell entwickelt wird, um diese Testfälle zu bestehen.

Testkonzept

Die Dokumentation der Testziele sowie der Maßnahmen und Zeitplanung, um diese zu erreichen, zum Zweck der Koordination des Testens.

Testlauf

Die Ausführung einer Testsuite auf einer bestimmten Version des Testobjekts.

Testmanagement

Der Prozess der Konzeptionierung, Zeitplanung, Schätzung, Überwachung, Berichterstattung, Steuerung und des Abschlusses von Testaktivitäten.

Testmanager

Die Person, die für das Projektmanagement von Testaktivitäten und Testressourcen und für die Bewertung eines Testobjekts verantwortlich ist.

Testmittel

Die Arbeitsergebnisse, die während des Testprozesses erstellt werden und dazu gebraucht werden, um die Tests zu planen, zu entwerfen, auszuführen, auszuwerten und darüber zu berichten.

Testobjekt

Das zu testende Arbeitsergebnis.

Testplan

Eine Liste von Aktivitäten, Aufgaben und Meilensteinen des Testprozesses, ihren geplanten Anfangs- und Endterminen sowie ihrer gegenseitigen Abhängigkeiten.

Testplanung

Eine Aktivität im Testprozess zur Erstellung und Fortschreibung des Testkonzepts.

Testprotokoll

Eine chronologische Aufzeichnung von Einzelheiten der Testausführung.

Testprozess

Die Menge zusammenhängender Aktivitäten bestehend aus Testplanung, Testüberwachung, Teststeuerung, Testanalyse, Testentwurf, Testrealisierung, Testdurchführung und Testabschluss.

Testpyramide

Ein graphisches Modell, welches das Verhältnis der Testumfänge der einzelnen Teststufen darstellt, mit mehr Umfang an der Basis als an der Spitze.

Testquadranten

Ein Klassifikationsmodell für Testarten bzw. Teststufen in vier Quadranten, das sich auf zwei Dimensionen von Testzielen bezieht: Unterstützung des Produktteams vs. Hinterfragen des Produkts und technologische Ausrichtung vs. geschäftliche Ausrichtung.

Testrahmen

Ein Satz von Treibern und Testdoubles, die zum Ausführen einer Testsuite erforderlich sind.

Testrealisierung

Die Aktivität, die auf Basis der Testanalyse und des -Entwurfs die Testmittel vorbereitet, welche für die Testdurchführung benötigt werden.

Testrichtlinie

Dokumentation, die auf hohem Abstraktionsniveau die Prinzipien, den Ansatz und die wichtigsten Ziele einer Organisation in Bezug auf das Testen zusammenfasst.

Testschätzung

Eine näherungsweise Abschätzung verschiedener Aspekte des Testens.

Testsitzung

Ein ununterbrochener Zeitraum, der mit Testdurchführung verbracht wird.

Testskript

Eine Abfolge von Anweisungen für die Durchführung eines Tests.

Teststeuerung

Die Aktivität, die Korrekturmaßnahmen entwickelt und anwendet, um ein Testprojekt auf den richtigen Weg zu bringen, wenn es vom Plan abweicht.

Teststrategie

Eine Beschreibung, wie Testen durchzuführen ist, um Testziele unter den gegebenen Umständen zu erreichen.

Teststufe

Eine spezifische Instanziierung eines Testprozesses.

Testsuite

Eine Menge von Testskripten oder Testabläufen, die in einem bestimmten Testlauf ausgeführt werden sollen.

Testüberwachung

Die Aktivität, die den Status von Testaktivitäten überprüft, alle Abweichungen vom Plan oder der Erwartung identifiziert und den Status an die Stakeholder meldet.

Testumgebung

Eine Umgebung, die Hardware, Instrumente, Simulatoren, Software-Tools und andere unterstützende Elemente enthält, welche zur Durchführung von Tests benötigt werden.

Testverfahren

Eine Vorgehensweise zum Definieren von Testbedingungen, Entwerfen von Testfällen und Spezifizieren von Testdaten.

Testziel

Der Zweck des Testens.

Testzyklus

Eine Testprozess-Instanz für eine bestimmte Version eines Testobjekts.

Treiber

Eine Komponente oder ein Werkzeug, das eine andere Komponente vorübergehend ersetzt und ein Testelement in Isolation steuert oder aufruft.

U

Überdeckung

Der Grad, zu dem bestimmte Überdeckungselemente von einer Testsuite ausgeführt wurden, ausgedrückt in Prozent.

Überdeckungselement

Eine Eigenschaft oder eine Kombination von Eigenschaften, die aus einer oder mehreren Testbedingungen unter Verwendung eines Testverfahrens abgeleitet wurde(n).

Überdeckungskriterien

Die Kriterien zur Definition der Überdeckungselemente, die zum Erreichen eines Testziels erforderlich sind.

Übertragbarkeit

Der Grad, zu dem eine Komponente oder ein System von einer Hardware, Software oder einer anderen Betriebs- oder Nutzungsumgebung auf eine andere übertragen werden kann.

Unabhängigkeit des Testens

Trennung der Verantwortlichkeiten, welche objektives Testen fördert.

Unittest-Framework

Ein Werkzeug, das eine Umgebung für einen Komponententest bereitstellt. In dieser Umgebung wird die Komponente isoliert oder mit geeigneten Treibern und Platzhaltern getestet. Darüber hinaus wird dem Entwickler zusätzliche Unterstützung (z.B. Debugging) zur Verfügung gestellt.

Ursache-Wirkungs-Diagramm

Eine graphische Darstellung zur Organisation und Darstellung der Zusammenhänge verschiedener möglicher Ursachen eines Problems. Mögliche Gründe einer echten oder potentiellen Fehlerursache oder -wirkung sind in Kategorien und Subkategorien einer horizontalen Baumstruktur organisiert, deren Wurzelknoten die (potentielle) Fehlerursache/-wirkung darstellt.

User-Story

Eine Benutzer- oder Geschäftsanforderung bestehend aus einem Satz in der Alltags- oder Geschäftssprache, welche die von einem Benutzer benötigte Funktionalität, ihre Begründung und die nicht-funktionalen Kriterien erfasst, und auch Akzeptanzkriterien enthält.

V

V-Modell

Ein sequentielles Modell des Softwareentwicklungslebenszyklus, das eine eins-zu-eins Beziehung zwischen den Phasen der Software-Entwicklung von der Anforderungsspe-

zifikation bis zur Lieferung, und den korrespondierenden Teststufen vom Abnahmetest bis zum Komponententest beschreibt.

Validierung

Bestätigung durch Überprüfung, dass ein Arbeitsergebnis den Bedürfnissen eines Stakeholders entspricht.

Verfolgbarkeit

Die Fähigkeit, explizite Beziehungen zwischen Arbeitsergebnissen oder zwischen Elementen von Arbeitsergebnissen darzustellen.

Verfügbarkeit

Der Grad, zu dem eine Komponente oder ein System betriebsbereit ist und bei Bedarf für die Nutzung zur Verfügung steht.

Verhaltensgetriebene Entwicklung

Eine kollaborative Entwicklungsvorgehensweise, bei der das Team den Schwerpunkt auf die Lieferung des erwarteten Verhaltens einer Komponente oder eines Systems für den Kunden legt, welches die Basis des Testens bildet.

Verifizierung

Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind.

Vorbedingung

Der erforderliche Zustand des Testelements und seiner Umgebung vor der Testdurchführung.

W

Walkthrough

Eine Reviewart, bei der ein Autor die Reviewteilnehmer durch ein Arbeitsergebnis leitet und die Teilnehmer Fragen stellen und Auffälligkeiten kommentieren.

Wartbarkeit

Der Grad, zu dem eine Komponente oder ein System von den dafür vorgesehenen Personen gewartet werden kann.

Wartung

Der Prozess der Modifikation einer Komponente oder eines Systems nach Auslieferung, um Fehlerzustände zu korrigieren, Qualitätsmerkmale zu verbessern oder für eine andere Umgebung zu adaptieren.

Wartungstest

Testen der Änderungen an einem laufenden System oder der Auswirkungen einer geänderten Umgebung auf ein laufendes System.

White-Box-Test

Ein Test, der auf der Analyse der internen Struktur einer Komponente oder eines Systems basiert.

White-Box-Testverfahren

Ein Testverfahren, das auf der inneren Struktur einer Komponente oder eines Systems basiert.

Z

Zustandsübergangstest

Ein Black-Box-Testverfahren, bei dem Testfälle entworfen werden, um Elemente eines Zustandsübergangsmodells auszuführen.

Zuverlässigkeit

Der Grad, zu dem eine Komponente oder ein System ihre spezifizierten Funktionen unter den festgelegten Bedingungen während einer bestimmten Zeitspanne ausführt.

Zweig

Ein Kontrollübergang zwischen zwei aufeinanderfolgenden Knoten im Kontrollflussgraphen eines Testelements.

Zweigtest

Ein White-Box-Testverfahren, bei dem die Testbedingungen Zweige sind.

Zweigüberdeckung

Die Überdeckung von Zweigen in einem Kontrollflussgraphen.

Verwendung in Ihren Dokumenten

Um auf Begriffe in diesem Glossar zu verlinken, verwenden Sie folgende Syntax in Ihren Jupyter Notebooks:

```
Die <a href="./Glossar.ipynb#fehlerwirkung" title="→ Glossar öffnen" class="glossary
entsteht durch einen <a href="./Glossar.ipynb#fehlerzustand" title="→ Glossar öffne
```

Tipp: Speichern Sie dieses Glossar als `Glossar.ipynb` in Ihrem Projektordner und verlinken Sie von Ihren Hauptdokumenten darauf.