Testing (I'd rather test dying)

1 Klassifizierung von Testverfahren {#1}

1.1 Wer testet? {#1.1}

1.1.1 Mensch (manuell) vs. Maschine (automatisch) {#1.1.1}

Manual testing and automated testing are two different ways of testing a software application's performance. Manual testing is done by a human tester who executes tests one-by-one. Automated testing is done by a tool or a framework that uses code and script to execute tests. Manual testing is prone to human errors and may not be accurate. Automated testing is faster, more reliable, and code-based. The choice of testing method depends on the test scenario.

Manual testing is a time-consuming process and requires human resources. It is possible without programming knowledge and allows random testing. However, it is not accurate because of the possibilities of human errors. On the other hand, automated testing is very fast and reliable because it is code and script-based. It is not possible without programming knowledge and doesn't allow random testing.

In general, any type of project can involve both manual and automated testing. However, the correct choice of testing strategy depends on the project itself. It is important to find a balance between manual and automated testing. Manual testing requires the tester to act as the target software user, while automated testing allows you to execute repetitive tasks and regression tests without the intervention of a manual tester.

1.1.2 Entwickler vs Benutzer{#1.1.2}

Software testing can be performed by both developers and users, but the objectives and methods can differ significantly:

Developer Testing: Developers are primarily responsible for writing and maintaining the source code of computer programs to develop software. They use unit tests and integration tests to ensure that individual methods, functions, classes, components, or modules work as expected. These tests are usually automated and can run very quickly by a continuous integration server. Developers need to have programming skills and proficiency at writing code.

User Testing: This is often referred to as acceptance testing, where the software is tested in a 'real world' scenario by the intended audience. Users may not have detailed knowledge of the system's inner workings, but they can provide valuable insights into how the software performs under typical usage conditions. User testing can help identify usability issues, confusing workflows, and unexpected behaviors that might not be apparent in developer testing.

In summary, developer testing is more focused on the technical correctness of the software, while user testing is more concerned with the software's usability and overall user experience. Both types of testing are crucial for delivering a high-quality software product.

1.2 Was wird getestet? {#1.2}

1.2.1 Komponente (Unit-Test/Funktionstest/Klassentest) vs. Integration vs. System (End-to-End) {#1.2.1}

Die Unterschiede zwischen Komponententests (auch bekannt als Unit-Tests oder Funktionstests), Integrationstests und Systemtests (End-to-End-Tests) sind wie folgt:

Komponententests (Unit-Tests/Funktionstests/Klassentests): Diese Tests überprüfen jede Softwarekomponente einzeln. Entwickler führen diese Tests durch, um sicherzustellen, dass einzelne Methoden, Funktionen, Klassen oder Module wie erwartet funktionieren. Diese Tests sind in der Regel automatisiert und können sehr schnell von einem Continuous Integration Server ausgeführt werden.

Integrationstests: Diese Tests integrieren individuell getestete Einheiten logisch und bewerten später die Interaktion oder Kommunikation zwischen diesen Einheiten. Sie sind dazu da, um sicherzustellen, dass verschiedene Teile der Software korrekt zusammenarbeiten. Sie können manuell oder automatisiert durchgeführt werden.

Systemtests (End-to-End-Tests): Diese Tests überprüfen ein komplettes Softwareprodukt. Sie sind dazu da, um sicherzustellen, dass das gesamte System, einschließlich aller seiner Komponenten und Integrationen, wie erwartet funktioniert. Sie können manuell oder automatisiert durchgeführt werden.

Zusammenfassend lässt sich sagen, dass Komponententests dazu dienen, die Funktionalität von einzelnen Teilen der Software zu überprüfen, während Integrationstests die Zusammenarbeit zwischen verschiedenen Teilen der Software testen und Systemtests die Funktionalität des gesamten Systems überprüfen.

1.2.2 Testpyramide {#1.2.2}

Die Testpyramide ist ein Konzept, das von Mike Cohn entwickelt wurde, um das Testen von Anwendungen zu systematisieren. Sie besteht aus drei Ebenen, die nach Geschwindigkeit und Kosten der Tests organisiert sind.

Unit-Tests (Komponententests): Diese bilden die unterste und breiteste Ebene der Pyramide. Sie sind schnell und können mit wenig Aufwand durchgeführt werden. Unit-Tests überprüfen das Verhalten einzelner Komponenten wie Klassen, Funktionen oder Methoden.

Integrationstests: Diese befinden sich auf der zweiten Stufe der Pyramide. Sie sind dazu da, um kritische Schnittstellen zu prüfen. Integrationstests prüfen die Schnittstelle und das Zusammenspiel zwischen zwei Komponenten.

Systemtests (End-to-End-Tests): Diese bilden die oberste Stufe der Pyramide. Sie sind langsam und wartungsintensiv und sollten daher möglichst bewusst eingesetzt und geplant werden. Systemtests überprüfen das Verhalten des gesamten Systems.

Das Konzept der Testpyramide plädiert dafür, dass Unit-Tests häufig und schnell erfolgen sollen, während komplexere und langsamere Integrations- und Systemtests seltener und weniger häufig durchgeführt werden sollten.

1.3 Wie wird getestet? {#1.3}

1.3.1 Bottom-Up vs. Top-Down {#1.3.1}

Top-Down Testing:

- This is an integration testing technique that tests the higher level modules first and then the lower level modules.
- It uses stubs to imitate the lower level modules that are not yet integrated.
- It examines the risk by collecting the internal operational failure impacts.
- It goes from major to minor components.

Bottom-Up Testing:

- This is an integration testing technique that tests the lower level modules first and then the higher level modules.
- It uses drivers to call the lower level modules that are integrated.
- It evaluates the individual processes risk with models' support.
- It goes from small to significant modules.

In summary, the top-down approach starts testing from the high-level modules and moves towards the low-level modules, using stubs for the lower level modules that are not yet integrated. On the other hand, the bottom-up approach starts testing from the low-level modules and moves towards the high-level modules, using drivers to call the lower level modules that are integrated. Quelle

1.3.2 statisch (Kompilierzeit) vs. dynamisch (Laufzeit) {#1.3.2}

Statisches (Kompilierzeit) und dynamisches (Laufzeit) Softwaretesting sind zwei verschiedene Ansätze zur Überprüfung von Softwarequalität.

Statisches Testing (Kompilierzeit):

- Statisches Testing wird durchgeführt, ohne den Code auszuführen.
- Es überprüft den Code, Anforderungsdokumente und Design-Dokumente, um Fehler zu finden.
- Statisches Testing verhindert Fehler.
- Es wird vor der Code-Bereitstellung durchgeführt.
- Statisches Testing ist weniger kostspielig.
- Es beinhaltet eine Checkliste für den Testprozess.

Dynamisches Testing (Laufzeit):

- Dynamisches Testing wird durchgeführt, indem der Code ausgeführt wird.
- Es überprüft das funktionale Verhalten des Softwaresystems, die Speichernutzung, die CPU-Nutzung und die Gesamtleistung des Systems.
- Dynamisches Testing findet und behebt Fehler.
- Es wird nach der Code-Bereitstellung durchgeführt.
- Dynamisches Testing ist kostspieliger.
- Es beinhaltet Testfälle für den Testprozess.

Zusammenfassend lässt sich sagen, dass statisches Testing dazu dient, Fehler zu verhindern und zu finden, indem der Code und die Dokumente überprüft werden, ohne dass der Code ausgeführt wird. Dynamisches Testing hingegen findet und behebt Fehler, indem der Code ausgeführt und das Verhalten des Softwaresystems überprüft wird. Quelle 1 Quelle 2

1.3.3 ohne Kenntnis des Codes (Blackbox) vs. mit Kenntnis des Codes (Whitebox) {#1.3.3}

Sicher, ich kann den Unterschied zwischen Blackbox- und Whitebox-Softwaretests erklären.

Blackbox-Testing:

- Blackbox-Testing ist eine Testmethode, bei der die interne Struktur/Design/Implementierung des zu testenden Elements dem Tester nicht bekannt ist.
- Nur das externe Design und die Struktur werden getestet.
- Blackbox-Testing konzentriert sich auf die Eingabe und Ausgabe der Software.
- Es verwendet Methoden wie Äquivalenzpartitionierung, Grenzwertanalyse und Fehlervermutung, um Testfälle zu erstellen.
- Blackbox-Testing ist im Allgemeinen für das funktionale Testen der Software verwendet.
- Es ist einfach zu verwenden, erfordert keine Programmierkenntnisse und ist effektiv bei der Erkennung von funktionalen Problemen.

Whitebox-Testing:

- Whitebox-Testing ist eine Testmethode, bei der die interne Struktur/Design/Implementierung des zu testenden Elements dem Tester bekannt ist.
- Implementierung und Auswirkungen des Codes werden getestet.
- Whitebox-Testing erfordert Kenntnisse in Programmiersprachen, Softwarearchitektur und Designmustern.
- Es verwendet Methoden wie Kontrollflusstests, Datenflusstests und Anweisungsabdeckung.
- Whitebox-Testing wird für das Testen der Software auf Einheitenebene, Integrationsebene und Systemebene verwendet.
- Es ist effektiv bei der Erkennung interner Defekte und stellt sicher, dass der Code effizient und wartbar ist.

Zusammenfassend lässt sich sagen, dass Blackbox-Testing dazu dient, die Funktionalität der Software zu testen, ohne dass Kenntnisse über die interne Funktionsweise der Software erforderlich sind. Whitebox-Testing hingegen erfordert Kenntnisse über die interne Funktionsweise der Software und testet den Code, die Algorithmen und Methoden. Quelle 1 Quelle 2

1.3.4 explorativ {#1.3.4}

Exploratives Testen ist ein Ansatz zum Testen von Software, der als gleichzeitiges Lernen, Testdesign und Testausführung beschrieben wird. Cem Kaner, der den Begriff 1984 geprägt hat, definiert exploratives Testen als "einen Stil des Softwaretests, der die persönliche Freiheit und Verantwortung des einzelnen Testers betont, die Qualität seiner Arbeit kontinuierlich zu optimieren, indem er testbezogenes Lernen, Testdesign, Testdurchführung und Interpretation der Testergebnisse als sich gegenseitig unterstützende Aktivitäten verwendet, die während des gesamten Projekts parallel ablaufen".

Während die Software getestet wird, lernt der Tester Dinge, die zusammen mit Erfahrung und Kreativität neue Tests hervorbringen. Exploratives Testen wird oft fälschlicherweise als Black-Box-Testtechnik angesehen. Korrekterweise ist es ein Testansatz, der auf jede Testtechnik in jeder Phase des Entwicklungsprozesses angewendet werden kann.

Explorative Tests versuchen herauszufinden, wie die Software tatsächlich funktioniert, und Fragen zu stellen, wie sie mit schwierigen und einfachen Fällen umgehen kann. Die Qualität des Tests hängt von der Fähigkeit

des Testers ab, Testfälle zu erfinden und Programmfehler zu finden. Je mehr der Tester über das Produkt und die verschiedenen Testmethoden weiß, desto besser wird der Test.

Exploratives Testen konzentriert sich auf Discovery und stützt sich auf die Anleitung des einzelnen Testers, um Defekte aufzudecken, die im Rahmen anderer Tests nicht ohne Weiteres abgedeckt werden können. Es ist ein wichtiger Teil einer umfassenden Testabdeckungsstrategie.

Quelle 1 Quelle 2

1.3.5 Schreibtischtest/Review {#1.3.5}

Schreibtischtest, auch bekannt als Desk Checking, ist ein Prozess, bei dem der Quellcode eines Programms manuell überprüft wird. Es beinhaltet das Durchlesen der Funktionen innerhalb des Codes und das manuelle Testen dieser Funktionen, oft mit mehreren Eingabewerten.

Es ist eine informelle Methode der Verifizierung, die von jedem Programmierer, der Software entwickelt, genutzt wird. Das Debuggen von Software während ihrer Entwicklung ist eine Form des Schreibtischtests. Der Entwickler setzt Haltepunkte oder überprüft die Ausgabe aus dem Modell, um zu verifizieren, dass es den in dem konzeptionellen Modell entwickelten Algorithmen entspricht.

Ein Schreibtischtest ist nicht narrensicher. Es liegt in der Verantwortung des Designer/Programmierers, sicherzustellen, dass er alle möglichen Pfade der Logik durchlaufen hat und jeden Datensatz verwendet hat, der benötigt wird. Schreibtischtests unterliegen menschlichen Fehlern, da der Auswerter die Anforderungen verstehen muss, bevor er die Logik bewerten kann.

Während Schreibtischtests nützlich sind, um logische Fehler und andere Probleme innerhalb des Quellcodes eines Programms aufzudecken, sind sie zeitaufwendig und unterliegen menschlichen Fehlern. Daher ist eine integrierte Entwicklungsumgebung (IDE) oder ein Debugging-Tool besser geeignet, um kleinere Probleme wie Syntaxfehler zu erkennen.

Quelle 1 Quelle 2 Quelle 3

1.4 Wann wird getestet? {#1.4}

1.4.1 Vor vs nach der Entwicklung {#1.4.1}

Software-Testing kann sowohl vor als auch nach der Entwicklung stattfinden, und beide Ansätze haben ihre eigenen Vor- und Nachteile.

Vor der Entwicklung: Dies wird oft als “statisches Testen” bezeichnet. Es beinhaltet die Überprüfung von spezifischen Dokumenten und Anforderungen, bevor der Code geschrieben wird. Es kann Aktivitäten wie Anforderungsanalyse, Software-Design und Einheitentests umfassen. Dieser Ansatz ermöglicht es, Fehler frühzeitig zu erkennen und zu beheben, bevor der Code entwickelt wird.

Nach der Entwicklung: Dies wird oft als “dynamisches Testen” bezeichnet. Es beinhaltet das Testen der funktionalen und nicht-funktionalen Aspekte der Software, während der Code in einer Laufzeitumgebung ausgeführt wird. Es kann Phasen wie Alpha-Tests, Beta-Tests und Akzeptanztests umfassen. Dieser Ansatz ermöglicht es, die Software unter realen Bedingungen zu testen und sicherzustellen, dass sie den Anforderungen der Benutzer entspricht.

Es ist wichtig zu beachten, dass das Testen nach der Entwicklung das inhärente Risiko birgt, dass Sie Ihren Implementierungscode refaktorisieren müssen (manchmal erheblich), um eine korrekte Testabdeckung zu gewährleisten, was je nachdem, wie spät diese Tests geschrieben werden, teuer werden kann.

In der Praxis wird oft eine Kombination aus beiden Ansätzen verwendet, um eine umfassende und effektive Teststrategie zu gewährleisten. Es ist wichtig, eine Balance zu finden und den Testprozess an die spezifischen Anforderungen und Ziele des Projekts anzupassen.

Quelle 1 Quelle 2 Quelle 3 Quelle 4 Quelle 5

1.4.2 Abnahmetests {#1.4.2}

Ein Abnahmetest, auch bekannt als Akzeptanztest oder User Acceptance Test (UAT), ist in der Softwaretechnik die Überprüfung, ob eine Software aus Sicht des Benutzers wie beabsichtigt funktioniert und dieser die Software akzeptiert.

Software-Anbieter testen dies oft mit Beta-Tests. Der Akzeptanztest ist nicht zu verwechseln mit einem Systemtest, der gewährleistet, dass die Software nicht abstürzt und die dokumentierten Anforderungen erfüllt sind.

Akzeptanztests sollten von einem Fachexperten, vorzugsweise dem Eigentümer oder Kunden der Software durchgeführt werden und nach der Zusammenfassung der Ergebnisse in der Entwicklung fortschreiten oder die Software überarbeiten.

In der Software-Entwicklung ist der Akzeptanztest eine der letzten Phasen eines Projektes, bevor der Kunde oder Nutzer die Software benutzt. Deshalb werden die Tests unter realitätsgetreuen Bedingungen durchgeführt.

Der Akzeptanztest fungiert als abschließende Überprüfung der erforderlichen Geschäftsfunktionen und das ordnungsgemäße Funktionieren des Systems unter realen Einsatzbedingungen im Namen des zahlenden Kunden oder eines bestimmten Großkunden.

Wenn die Software, wie angefordert und ohne Probleme bei normalem Gebrauch funktioniert, kann man das gleiche Maß an Stabilität in der Produktion erwarten.

Akzeptanztests werden in der Regel von Kunden oder Endbenutzern durchgeführt. Sie fokussieren sich nicht auf die Identifizierung einfacher Probleme wie Rechtschreibfehler, kosmetische Probleme und Softwareabstürze. Tester und Entwickler erkennen und beheben diese Probleme in der Regel in früheren Unit-Tests, Integrationstests und Systemtestphasen.

Die Ergebnisse dieser Tests geben Vertrauen bei den Klienten, wie die Software nach der Produktion funktionieren wird. Ein bestandener Akzeptanztest kann auch eine gesetzliche oder vertragliche Voraussetzung für die Annahme der Software sein.

1.5 Warum wird getestet? {#1.5}

1.5.1 Regressionstest (Depressionstests tbh) {#1.5.1}

Ein Regressionstest, auch bekannt als Regressionstesting, ist ein Prozess in der Softwaretechnik, bei dem wiederholt Testfälle ausgeführt werden, um sicherzustellen, dass Modifikationen in bereits getesteten Teilen

der Software keine neuen Fehler („Regressionen“) verursachen. Solche Modifikationen entstehen regelmäßig z. B. aufgrund der Pflege, Änderung und Korrektur von Software.

Regressionstests sind eine Reihe von Tests, die unmittelbar vor der Freigabe eines neuen Features oder Updates durchgeführt werden, um sicherzustellen, dass die kritischen Funktionen Ihrer Software weiterhin funktionieren. Regressionstests werden durchgeführt, um sicherzustellen, dass ein neuer Build keine Fehler in andere, bestehende Funktionen Ihrer Software eingeführt hat.

Regressionstests konzentrieren sich auf zwei Elemente der Quellcodeänderungen:

1. Verhält sich die neue Änderung in der erwarteten, gewünschten Weise?
2. Sind andere Funktionen betroffen, auch solche, die scheinbar nichts mit der Änderung zu tun haben?

Idealerweise werden Regressionstests nach jeder Quellcodeänderung durchgeführt. Bei einer Anwendung auf Unternehmensebene sind wahrscheinlich Tausende von Tests erforderlich, die automatisierte Regressionstest-Tools erfordern.

Es ist wichtig zu beachten, dass Regressionstests ein wesentliches Problem darstellen, insbesondere bei nichtdeterministischen Echtzeitsystemen, da in diesen Systemen eine Wiederholung des Tests streng genommen nicht gewährleistet ist. Eine Lösung dieses Problems liegt in der Implementierung eines automatischen Testsystems.

Quelle 1 Quelle 2 Quelle 3

1.5.2 Lasttest/Belasstungstest {#1.5.2}

Ein Lasttest, auch bekannt als Belastungstest, ist ein Prozess in der Softwaretechnik, bei dem die Leistung von Systemen unter unterschiedlichen Arbeitslasten geprüft wird. Dies kann beispielsweise vor der Veröffentlichung (Produktivsetzung) oder während der Entwicklung oder des Betriebs einer Software notwendig sein.

Lasttests sind ein Teilbereich der Leistungstests für Software, Websites, Anwendungen und verwandte Systeme. Es handelt sich um einen nicht-funktionalen Test, der das Verhalten mehrerer Benutzer simuliert, die gleichzeitig auf das System zugreifen. Lasttests, die auch als „Volumentests“ bezeichnet werden, reproduzieren die Leistung, Stabilität und Funktionalität des Websystems unter Live-Bedingungen und sind daher eine der letzten und wichtigsten Testarten vor der Implementierung.

Bei Lasttests werden mehrere kritische Aspekte des Websystems ermittelt, darunter die folgenden:

- Die Gesamtbetriebskapazität der Anwendung, einschließlich der Anzahl der gleichzeitigen Benutzer, die unterstützt werden können.
- Die Fähigkeit der Anwendung, auf Spitzenlasten zu reagieren2.
- Die Stabilität der Infrastruktur der Anwendung.
- Antwortzeiten, Durchsatzraten und Ressourcenbedarf der Anwendung bei unterschiedlicher Benutzerbelastung.

Lasttests sind ein wichtiger Prozess, der vor dem Start jeder Client/Server-Internet- und Intranet-Anwendung durchgeführt wird2. Sie gilt sowohl für Front-End-Software, wie z. B. eine Website, als auch für Back-End-Systeme, wie z. B. die Server, die die Website hosten.

Quelle 1 Quelle 2

1.5.3 Smoketest {#1.5.3}

Smoke Testing, auch bekannt als Build Verification Testing oder Confidence Testing, ist eine vorläufige Testmethode oder ein Sanity-Test, um einfache Fehler aufzudecken, die schwerwiegend genug sind, um beispielsweise eine potenzielle Softwarefreigabe abzulehnen.

Es handelt sich um eine Software-Testmethode, die bestimmt, ob der verwendete Build stabil ist oder nicht. Smoke-Tests sind eine Mindestmenge von Tests, die auf jedem Build ausgeführt werden. Smoke-Testing überprüft die Funktionalität wichtiger Teile eines Programms, bestimmt jedoch nicht, ob das Programm fehlerfrei ist.

Smoke-Tests werden in der Regel früh im Prozess durchgeführt, bevor detailliertere Tests stattfinden. Der Zweck dieser Tests besteht in der Regel darin, zu bestätigen, dass die Anwendungs-Builds stabil genug sind, um weiter getestet zu werden.

In den meisten Test-Szenarien werden Smoke-Tests durchgeführt, um sicherzustellen, dass die kritischen Funktionen Ihrer Software reibungslos funktionieren. Durch die Durchführung von Smoke-Tests können Sie schnell kritische Fehler identifizieren und diese frühzeitig im Entwicklungsprozess beheben, bevor Sie in detailliertere Details eintauchen.

Ein Software-Build ist ein Verfahren, bei dem der Quellcode in eine eigenständige Form umgewandelt wird, die auf jedem System ausgeführt werden kann. Es gibt jedoch immer Risiken, dass der Build in Ihrer Umgebung nicht funktioniert. Beispielsweise könnten Konfigurations-, Code-, Regressions- oder Umgebungsprobleme bestehen. Daher wird der erste Build einem Smoke-Test unterzogen, bevor er zu anderen Teststufen geschickt wird.

Quelle 1 Quelle 2

2 Methoden zur Ermittlung von Testfällen {#2}

2.1 Anweisungsüberdeckung vs. Zweig-/Pfadüberdeckung {#2.1}

Die **Anweisungsüberdeckung** (auch bekannt als C0 oder Statement Coverage) ist ein Maß für die Qualität der Abdeckung, das prüft, ob alle Anweisungen im Code mindestens einmal ausgeführt werden. Sie ist ein grundlegendes Maß, das jedoch nicht alle möglichen Pfade durch den Code abdeckt.

Die **Zweigüberdeckung** (auch bekannt als C1 oder Branch Coverage) ist ein stärkeres Maß für die Qualität der Abdeckung. Sie erfordert, dass alle Zweige des Programms (d.h., sowohl der “wahre” als auch der “falsche” Zweig jeder Entscheidung) durchlaufen werden. Eine 100%ige Zweigüberdeckung schließt sowohl eine 100%ige Entscheidungsüberdeckung als auch eine 100%ige Anweisungsüberdeckung ein.

Die **Pfadüberdeckung** (auch bekannt als Path Coverage) ist das stärkste Maß für die Qualität der Abdeckung. Sie erfordert, dass jeder mögliche Pfad durch das Programm von Anfang bis Ende begangen wird. Sie ist eine echte Obermenge der Zweigüberdeckung und wird daher als das härteste Maß der Überdeckungsmessung betrachtet.

Es ist wichtig zu beachten, dass jedes dieser Maße seine eigenen Stärken und Schwächen hat und in verschiedenen Kontexten angemessen sein kann. Es ist auch wichtig zu beachten, dass höhere Abdeckungsgrade in der Regel mehr Testaufwand erfordern.

Quelle 1 Quelle 2 Quelle 3 Quelle 4

2.2 Äquivalenzklassen {#2.2}

Der Äquivalenzklassentest ist eine Methode zur Qualitätsprüfung von Software. Das Ziel der Bildung von Äquivalenzklassen ist es, eine hohe Fehlerentdeckungsrate mit einer möglichst geringen Anzahl von Testfällen zu erreichen. Die Äquivalenzklassen sind also bezüglich Ein- und Ausgabedaten ähnliche Klassen bzw. Objekte, bei denen erwartet wird, dass sie sich gleichartig verhalten.

Die Bildung von Testfällen zu Äquivalenzklassen folgt dieser Abfolge:

1. Analyse und Spezifikation der Eingabedaten, der Ausgabedaten und der Bedingungen gemäß den Spezifikationen
2. Bildung der Äquivalenzklassen durch Klassifizierung der Wertebereiche für Ein- und Ausgabedaten
3. Bestimmung der Testfälle durch Werteauswahl für jede Äquivalenzklasse

Die erstellten Testfälle gelten somit für alle Objekte der erstellten Äquivalenzklasse, sodass nicht für jede Ausprägung ein eigener Testfall erstellt werden muss. Es wird zwischen gültigen Äquivalenzklassen und ungültigen Äquivalenzklassen unterschieden. Bei gültigen Äquivalenzklassen werden gültige Eingabedaten, bei ungültigen Äquivalenzklassen ungültige Eingabedaten verwendet.

Quelle 1

2.3 Grenzwertanalyse/Extremwertetest {#2.3}

Die Grenzwertanalyse (auch bekannt als Extremwertetest) ist eine Methode zur Qualitätsprüfung von Software. Sie wird verwendet, um die korrekte Behandlung von Werten und erwarteten Ergebnissen zu prüfen, die an den Grenzen von geordneten Äquivalenzklassen vorhanden sind.

Die Grenzwertanalyse konzentriert sich auf die Testfälle an den Rändern der Äquivalenzklassen, da hier häufig Fehler auftreten. Beispielsweise könnten in einer Software, die das Alter einer Person verarbeitet, die Grenzwerte bei den Altersklassen 0 und 16 liegen.

Die Schritte zur Durchführung einer Grenzwertanalyse sind:

1. Gruppierung von gleichwertig behandelten Testbedingungen in gültige Äquivalenzklassen.
2. Bildung von ungültigen Äquivalenzklassen durch Negation der gültigen Klassen.
3. Für jede gültige Äquivalenzklasse wird nur ein Testfall stellvertretend ausgeführt (statt alle einzeln).
4. Für jede ungültige Äquivalenzklasse wird ebenfalls nur ein Testfall durchgeführt.

Die Grenzwertanalyse ist eine effektive Methode zur Identifizierung von Implementierungs- und Interpretationsfehlern an den Grenzen der einzelnen Klassen.

Quelle 1 Quelle 2 Quelle 3

3 Modultests erstellen und durchführen {#3}

3.1 Test-Doubles: Stubs vs. Mocks {#3.1}

Stubs und Mocks sind beides Test-Doubles, die in der Softwareentwicklung verwendet werden, insbesondere bei Unit-Tests. Beide sehen aus wie und verhalten sich wie ihre Produktionsäquivalente, sind aber tatsächlich

vereinfacht.

Ein **Stub** ist ein Objekt, das vordefinierte Daten enthält und diese verwendet, um während der Tests auf Aufrufe zu antworten. Es wird verwendet, wenn wir Objekte, die mit echten Daten antworten würden oder unerwünschte Nebeneffekte hätten, nicht einbeziehen können oder wollen. Stubs sind ideal für Zustandstests, bei denen der Fokus auf dem Ergebnis und dem Verhalten des tatsächlich getesteten Objekts liegt.

Ein **Mock** ist ein Objekt, das das Verhalten des zu testenden Codes überprüft, indem es prüft, ob die richtigen Methoden aufgerufen wurden und ob sie mit den richtigen Argumenten aufgerufen wurden. Mocks zeichnen sich im Verhaltenstest aus, bei dem die Interaktionen zwischen den Komponenten genau untersucht werden.

Obwohl Stubs und Mocks unter die allgemeine Kategorie der Test-Doubles fallen, haben sie unterschiedliche Anwendungsfälle und können in verschiedenen Kontexten angemessen sein.

Quelle 1 Quelle 2 Quelle 3 Quelle 4

3.2 Eigenschaften guter Unit-Tests {#3.2}

Gute Unit-Tests sollten folgende Eigenschaften aufweisen:

1. **Korrekt**: Die Tests müssen in sich fehlerfrei sein und zu den Anforderungen passen.
2. **Isoliert**: Die Tests müssen unabhängig von anderen Tests durchführbar sein und dürfen andere Tests nicht beeinflussen.
3. **Schnell**: Die Tests müssen schnell laufen, um häufig durchgeführt werden zu können.
4. **Aussagekräftig**: Die Tests sollten ihre Absicht durch sprechende Benennung kundtun.
5. **Wartbar**: Die Tests sollten den Regeln für sauberen Code folgen und sich leicht an den veränderten Produktivcode anpassen lassen.
6. **Einfach durchführbar**: Die Tests müssen so einfach wie möglich (am besten auf Knopfdruck) durch einen beliebigen Entwickler durchführbar sein.

Diese Eigenschaften tragen dazu bei, dass Unit-Tests effektiv und effizient sind und einen hohen Wert für die Qualitätssicherung der Software liefern.

Quellen 1

4 Testkonzepte erstellen, Tests durchführen, Testergebnisse bewerten und dokumentieren {#4}

4.1 Definition der Inhalte eines Tests {#4.1}

- **Testkonzept**: Ein Testkonzept beschreibt für ein spezifisches Projektvorhaben die vorgegebenen und zu erfüllenden Mindestanforderungen für die geplante Testvorgehensweise, die Zuständigkeiten, der Verantwortlichkeiten und Abhängigkeiten. Es konkretisiert die im Dokument, das für die übergreifende Teststrategie des Gesamtunternehmens erstellt wurde, beschriebenen Prozesse und Richtlinien. Es identifiziert die Testobjekte, die zu testenden Features und die Testaufgaben.

- **Testdaten**: Testdaten sind Daten aus dem Definitionsbereich eines Moduls, Programms oder Softwaresystems, die zum Testen herangezogen werden. Die Auswahl geeigneter Testdaten ist die schwierigste Aufgabe beim Testen. Testdaten können im Testobjekt (z.B. in einer Datenbank)

gespeichert werden oder über Externe oder externe Schnittstellen zur Verfügung gestellt werden, wie etwa durch Systeme, Systemkomponenten, Hardware oder einen Anwender.

- **Testszenario**: Ein Testszenario ist eine (logische und chronologische) Kombination mehrerer Testfälle mit dem Zweck, einen komplexen Sachverhalt zu überprüfen. Idealerweise hat ein Testszenario folgende Merkmale: (a) Es ist eine Geschichte, die (b) motiviert, (c) glaubwürdig erscheint, (d) komplex ist und (e) leicht überprüft werden kann5.

Quelle 1 Quelle 2 Quelle 3 Quelle 4 Quelle 5

4.2 Beschreiben des Testumfangs {#4.2}

- Grenzbelastung: Die Grenzbelastung ist eine relevante Größe für die Einkommenserzielung und die Entscheidung, ob und in welchem zeitlichen Umfang die eigene Arbeitskraft auf dem Arbeitsmarkt angeboten wird. Sie gibt an, welcher Anteil eines zusätzlich verdienten Euros wieder abgegeben werden müsste – sei es durch Transferentzug im Sozialbereich, durch Einkommensteuern oder als Beitrag zur Sozialversicherung. Je höher die Grenzbelastung ausfällt, desto weniger bleibt vom zusätzlichen Bruttoeinkommen netto übrig.

- Stabilität: Stabilität deckt den zeitlichen Aspekt ab und überprüft damit also die Konstanz bzw. Ähnlichkeit von Messergebnissen zu verschiedenen Zeitpunkten. Ein gängiges Verfahren zur Überprüfung von Stabilität ist die Test-Retest-Methode. In der Pharmazie, bei Stabilitätstests fungiert die Wirkstoffkonzentration als Messgröße, die laut Definition während der Haltbarkeitsdauer 90% des Ausgangswertes nicht unterschreiten darf. Sie ist ein Maß für chemische und für die meisten physikalischen Instabilitäten. Das Ziel von Stabilitätsprüfungen ist es herauszufinden, wie sich ein pharmazeutisches Produkt oder ein Wirkstoff unter bestimmten Bedingungen (Temperatur, Luftfeuchtigkeit, Licht) während einer bestimmten Zeitperiode verändert.

Quelle 1 Quelle 2 Quelle 3 Quelle 4 Quelle 5

4.3 Testdatengeneratoren

Ein Testdatengenerator ist ein spezialisiertes Softwaretool, das falsche oder Scheindaten zur Verwendung beim Testen von Softwareanwendungen generiert. Die erzeugten Daten können entweder zufällig oder speziell ausgewählt sein, um ein gewünschtes Ergebnis zu erzeugen.

Testdatengeneratoren sind Werkzeuge, mit denen Dummy-Datensätze für Testzwecke erstellt werden können. Diese Tools helfen Unternehmen bei der Entwicklung und Prüfung von Anwendungen und Produkten mit realistischen Datensätzen, die in einer Vielzahl von Szenarien verwendet werden können.

Es gibt verschiedene Arten von Testdatengeneratoren, darunter Zufallsdatengeneratoren, musterbasierte Generatoren und datengesteuerte Generatoren. Die Wahl des richtigen Testdatengenerators hängt von verschiedenen Faktoren ab, wie der Art der zu erzeugenden Daten, der Komplexität der Datensätze und den Kosten des Tools.

Quelle 1 Quelle 2

5 Testprozess {#5}

5.1 Auswahl des Testverfahrens {#5.1}

Die Auswahl des richtigen Testverfahrens im Softwaretesting hängt von verschiedenen Faktoren ab, darunter die Art der Software, die getestet wird, die Anforderungen und Ziele des Tests und die verfügbaren Ressourcen. Hier sind einige Arten von Softwaretests, die Sie in Betracht ziehen könnten:

1. **Unit-Tests**: Diese Tests sind sehr einfach und erfolgen nah an der Quelle der Anwendung. Sie sind ideal, um einzelne Komponenten oder Funktionen der Software zu testen.
2. **Integrationstests**: Diese Tests prüfen, wie verschiedene Teile der Software zusammenarbeiten. Sie sind nützlich, um Probleme mit der Interaktion zwischen Komponenten zu identifizieren.
3. **Funktionstests**: Diese Tests überprüfen, ob die Software wie erwartet funktioniert und ob sie die festgelegten Anforderungen erfüllt.
4. **Akzeptanztests**: Diese Tests werden durchgeführt, um zu bestätigen, dass die Software die Anforderungen der Benutzer erfüllt und bereit für die Bereitstellung ist.

Es ist wichtig, zwischen manuellen und automatischen Tests zu unterscheiden. Manuelle Tests werden von einem Menschen durchgeführt, während automatische Tests von einem Computer durchgeführt werden, der ein zuvor geschriebenes Testskript ausführt.

Die Qualität der automatischen Tests hängt von der Qualität der Testskripts ab. Automatische Tests sind ein wichtiger Bestandteil von Continuous Integration und Continuous Delivery.

Es ist wichtig, dass Sie eine Teststrategie entwickeln, die Ihre spezifischen Anforderungen und Ziele berücksichtigt. Dies sollte Aspekte wie Testumfang, Testabdeckung, Risikoabschätzung, Testziele und Kriterien für Testbeginn, Testende und Testabbruch, Vorgehensweise (Testarten), Hilfsmittel und Werkzeuge zum Testen, Dokumentation, Testumgebung, Testdaten, Testorganisation, alle Ressourcen, Ausbildungsbedarf und Testmetriken umfassen.

Quelle 1 Quelle 2

5.2 Kriterien für Testergebnisse definieren {#5.2}

Die Definition von Kriterien für Testergebnisse ist ein wichtiger Schritt im Softwaretestprozess. Hier sind einige Aspekte, die Sie berücksichtigen sollten:

1. Testziele: Definieren Sie, was Sie mit dem Test erreichen möchten. Dies könnte beispielsweise sein, bestimmte Funktionen zu überprüfen, die Leistung zu messen oder die Kompatibilität mit bestimmten Systemen zu testen.
2. Erfolgskriterien: Legen Sie fest, was ein erfolgreicher Test ausmacht. Dies könnte beispielsweise sein, dass keine Fehler gefunden werden, dass die Software eine bestimmte Leistung erreicht oder dass sie mit den vorgesehenen Systemen kompatibel ist.
3. Fehlerkriterien: Definieren Sie, was als Fehler gilt. Dies könnte beispielsweise sein, dass die Software abstürzt, dass sie nicht die erwartete Leistung erbringt oder dass sie nicht mit den vorgesehenen Systemen kompatibel ist.
4. Messbare Kriterien: Stellen Sie sicher, dass Ihre Kriterien messbar sind. Zum Beispiel könnten Sie die Anzahl der gefundenen Fehler, die Zeit, die die Software zum Ausführen bestimmter Aufgaben benötigt, oder die Menge der Ressourcen, die sie verbraucht, messen.

5. Testdaten: Überlegen Sie, welche Daten Sie für den Test benötigen und wie Sie diese Daten beschaffen können.

6. Testverfahren: Wählen Sie das geeignete Testverfahren aus, z.B. Unit-Tests, Integrationstests, Funktionstests oder Akzeptanztests.

7. Testgütekriterien: Berücksichtigen Sie die Testgütekriterien wie Objektivität, Reliabilität und Validität.

Es ist wichtig, dass Sie Ihre Testkriterien sorgfältig definieren und dokumentieren, damit Sie Ihre Tests effektiv durchführen und Ihre Ergebnisse zuverlässig interpretieren können.

Quelle 1 Quelle 2 Quelle 3

5.3 Testdaten generieren und auswählen {#5.3}

Die Generierung und Auswahl von Testdaten ist ein wichtiger Schritt im Softwaretestprozess. Hier sind einige Methoden, die Sie verwenden können:

1. Manuelle Erstellung: Sie können Testdaten manuell erstellen, indem Sie Daten eingeben, die spezifische Testfälle abdecken. Dies kann zeitaufwendig sein, insbesondere wenn Sie eine große Menge an Daten .

2. Verwendung von Testdatengeneratoren: Es gibt verschiedene Tools und Bibliotheken, die Ihnen helfen können, Testdaten zu generieren. Beispielsweise können Sie den MIGANO Online Tools - Testdaten-Generator oder generatedata.com verwenden, um Testdaten in verschiedenen Formaten (z.B. CSV, Excel, XML oder SQL) zu generieren.

3. Verwendung von Bibliotheken in Programmiersprachen: Einige Programmiersprachen haben Bibliotheken, die Ihnen helfen können, Testdaten zu generieren. Zum Beispiel hat Python eine Bibliothek namens Faker, die Ihnen hilft, “Fake-Daten” oder Beispiel-Daten zu erzeugen.

Bei der Auswahl von Testdaten ist es wichtig, dass Sie Daten auswählen, die eine Vielzahl von Szenarien abdecken, einschließlich normaler, Grenz- und ungültiger Fälle. Sie sollten auch sicherstellen, dass Ihre Testdaten die Realität so gut wie möglich widerspiegeln.

Quelle 1 Quelle 2 Quelle 3

5.4 Testprotokoll und Auswertung {#5.4}

Ein Testprotokoll und eine Auswertung sind wichtige Elemente in vielen Bereichen, einschließlich Softwareentwicklung, wissenschaftlicher Forschung und medizinischer Tests. Hier sind einige Punkte, die in einem Testprotokoll und einer Auswertung berücksichtigt werden sollten:

Testprotokoll:

- Es legt die Durchführung und Auswertung der Studie fest.
- Es nennt alle Arbeitsschritte einer Studie.
- Es dient als Fahrplan und Handbuch einer Studie.
- Es unterstützt die Qualitätssicherung.
- Es sollte immer mit einer Versionsnummer und einem Datum versehen sein.

Auswertung:

- Die Auswertung beinhaltet die Analyse der Daten, die aus einem Test stammen.
- Sie können Tests auch nach Test-ID und Namen nachverfolgen, jeden Schritt eines Tests identifizieren, Prioritätsstufen und Notizen hinzufügen und tatsächliche und erwartete Ergebnisse vergleichen.
- Es ist wichtig, die Testergebnisse sorgfältig zu analysieren und zu überprüfen, um die Funktionen auf Basis der Testergebnisse zu aktualisieren.

Bitte beachten Sie, dass die spezifischen Anforderungen an ein Testprotokoll und eine Auswertung je nach Kontext variieren können.

Quelle 1 Quelle 2 Quelle 3

6 Auswerten von Testprotokollen (Ist-/Soll-Vergleich) {#6}

Ein Soll-Ist-Vergleich im Softwaretesting ist ein Prozess, bei dem die tatsächlichen Testergebnisse (Ist) mit den erwarteten Ergebnissen (Soll) verglichen werden. Dieser Vergleich ist entscheidend, um zu bestätigen, dass die Software das tut, was sie soll. Hier sind einige Schritte, die in der Regel bei der Auswertung von Testergebnissen befolgt werden:

1. Durchführung der Tests: Zunächst werden die Tests durchgeführt. Dies kann manuell oder automatisch erfolgen.
2. Sammeln der Testergebnisse: Nachdem die Tests durchgeführt wurden, werden die Testergebnisse gesammelt.
3. Soll-Ist-Vergleich: Die gesammelten Testergebnisse (Ist) werden mit den erwarteten Ergebnissen (Soll) verglichen.
4. Identifizierung von Abweichungen: Wenn die Ist-Ergebnisse nicht den Soll-Ergebnissen entsprechen, werden diese Abweichungen identifiziert.
5. Analyse der Abweichungen: Die Ursachen für die identifizierten Abweichungen werden analysiert.
6. Korrekturmaßnahmen: Basierend auf der Analyse der Abweichungen werden Korrekturmaßnahmen definiert.
7. Dokumentation: Alle Schritte, von den Testergebnissen über die identifizierten Abweichungen bis hin zu den Korrekturmaßnahmen, werden dokumentiert.

Es ist wichtig zu beachten, dass der Soll-Ist-Vergleich nicht nur dazu dient, Fehler zu finden, sondern auch, diese einer Lösung zuzuführen. Daher ist das Ziel des Softwaretests nicht, möglichst viele Fehler zu finden, sondern diese auch einer Lösung zuzuführen.

Quelle 1

7 Kontrollverfahren {#7}

7.1 Hardwartests {#7.1}

1. **Hardwaretest**: This refers to the process of testing hardware components to ensure they are functioning correctly. It can involve various tests such as speed tests, GPU tests, microphone tests, and more.

2. **Wareneingangskontrolle**: This is the process of checking incoming goods to ensure they are undamaged and meet the company's quality requirements. It aims to economically secure companies, maintain operational processes, and prevent subsequent costs.

3. **Mangelhafte Lieferung**: This term refers to a delivery that has defects or issues. If the goods delivered are not in perfect condition, have a physical, chemical, or biological hazard, or do not correspond to what was agreed in the purchase contract, it is considered a defective delivery.

4. **Warenausgangskontrolle**: This is the process of checking outgoing goods to ensure they are correctly packaged and ready for shipment. It is an important step in the process of dispatching goods, contributing to customer satisfaction by ensuring that goods arrive quickly and safely at their destination.

5. **Abnahmeprotokoll**: This is a document that confirms the correct execution of work by a contractor. It is used to compare the current state of a building with the contents of the construction description and the construction contract. It is signed by both the client and the contractor, giving it significant legal importance.

Quelle 1, Quelle 2, Quelle 3 Quelle 4, Quelle 5, Quelle 6, Quelle 7, Quelle 8, Quelle 9, Quelle 10, Quelle 11, Quelle 12, Quelle 13, Quelle 14

7.2 Softwaretests {#7.2}

1. **Software-Test**: Software testing is the act of examining the artifacts and the behavior of the software under test by validation and verification. It can provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, analyzing the product requirements for completeness and correctness in various contexts like industry perspective, business perspective, feasibility and viability of implementation, usability, performance, security, infrastructure considerations, etc.

2. **Testverfahren**: Test procedures indicate how testing is done. There are fundamentally two types of test techniques: static test techniques and dynamic test techniques. The dynamic test involves the execution of the test element. The static test, on the other hand, tests work results without executing code.

3. **Abnahmeprotokoll**: An acceptance protocol is a document that confirms the correct execution of work by a contractor. It is used to compare the current state of a building with the contents of the construction description and the construction contract. It is signed by both the client and the contractor, giving it significant legal importance.

Quelle 1, Quelle 2, Quelle 3, Quelle 4, Quelle 5, Quelle 6, Quelle 7