Was ist ein Testkonzept? Das Testkonzept beschreibt die Testziele, Testobjekte, Testarten, Testinfrastruktur sowie die Testorganisation. Es umfasst ebenfalls die Testplanung und die Testfallbeschreibungen. Für jeden Testfall wird eine detaillierte Testfallbeschreibung erstellt. Diese stellt die Spezifikation des Tests dar

Inhalte eines Testkonzepts :• Testziele • Teststrategie und Teststufen • Testobjekte • Testarten • Testabdeckung • Testrahmen • Testumgebung • Testinfrastruktur • Testorganisation • Testfallbeschreibungen • Testplan**Teststrategien & Teststufen** • Teststrategie / Vorgehensweise • Analytisch, modellbasiert, methodisch, etc. • **Teststufen (V-Modell)** • Unit tests, integration tests, acceptance tests, etc. • Defizite der Testumgebung • Etc.**Testobjekte** Abzudeckende Bestandteile der Software •Frontend •Backend •Schnittstellen •Etc.**Testarten Statisch** •Tests ohne Programmausführung • Reviews • Code Analyse •Etc. **Dynamisch** •Tests mit Programmausführung • Unit tests •Integration •Etc.*Testabdeckung* Wie werden welche Teile getestet Bespiele: • Login-Funktion: •IntegrationTest • Unit tests •Schnittstelle zu BankX**Testrahmen** • Voraussetzungen • Vorkenntnisse, Tester, etc. • Mängelklassifizierung **Testrahmen: Mängelklassifizierung** • Definition eines Mangels • Klassifizierung • Bedeutung und Wichtigkeit**Mängelklassifizierung - Was ist ein Mangel?** • „Bugs“ • Ein Nichterfüllen der Spezifikation • „Nicht was sich der Kunde vorgestellt hat“?**Mängelklassifizierung - Klassifizierung** • Klasse 1: belangloser Mangel •Einwandfrei und anforderungsgerecht • Klasse 2: leichter Mangel •Verwendung möglich,Klasse 3: schwerer Mangel •Verwendung ist noch möglich • Klasse 4: kritischer Mangel •Unbrauchbar; **Testumgebung** • Immer: Getrennt vom produktiven System • Trotzdem „so ähnlich wie möglich“ • Test- & Produktivumgebung • Evtl. auch noch Stage • Produktive Daten? • Datenschutz?T**estumgebung - Gemeinsamkeiten** • Hardware & Arch • CPU, Ram, x64, ARM, ... • OS • Linux (distro), Windows, ... • Deployment • Docker, K8s, rsync, Azure, GCP ... • Schnittstellen • Firewalls, Datenschutz & Datensicherheit **Testumgebung - Unterschiede** • Daten • Kundendaten sind sensibel! • Versand von E-Mails, SMS, etc. • Mailhog • Hardware • Systemlast **Testinfrastruktur Beschreibung** der: •Testsysteme •Testdaten •Testhilfsmittel (Software für Testmanagement, CI/CD, etc. **Testorganisation** • Wer organisiert die Tests • Wer führt die Tests durch • Wie werden die Resultate verwertet • usw… **Was ist ein Testprotokoll?** Das Testprotokoll hält die Testergebnisse fest. Die Testergebnisse sind gemäss den im Testkonzept definierten Mängelklassen bewertet.**Der Aufbau eines Testfalls** Tests sind nach einem Schema aufgebaut: •Vorbedingungen •Eingabewerte • Resultat • NachbedingungenT**estfall - Vorbedingungen** „Die Vorbedingung einer Funktion oder eines Programms gibt an, unter welchen Voraussetzungen das Verhalten der Funktion definiert ist.“ • Beispiele: • User mit der Rolle „Admin“ • Eine Liste mit 100 Benutzer • Ich bin als Admin eingeloggt

Testfall - Eingabewerte Welche Werte eingegeben / mitgegeben werden, welche Schritte durchlaufen werden, etc. •**Testfall - Resultat** Entscheidungsgrundlage für Test „fail“ oder „pass“Beispiele: • assert(user.admin).isTrue() • Benutzer ist jetzt als Admin gekennzeichnet Testfall - Nachbedingungen Dinge die nach dem positiven Test zutreffen müssen • Bespiele: • Nachdem ein Benutzer gelöscht wurde darf dieser nicht mehr in der Liste der Benutzer erscheinen **Einführung in Unit Testing** Unit Testing ist der Prozess des Überprüfens kleiner Einheiten des Codes, um sicherzustellen, dass sie wie erwartet funktionieren. **Definition von Unit Testing** Ein Testverfahren, bei dem individuelle Codeeinheiten isoliert getestet werden, um ihre Funktionalität sicherzustellen. **Zweck von Unit Tests** Sicherstellung der Codequalität, Identifikation von Bugs in frühen Entwicklungsphasen und Schutz vor Regressionsfehlern.**Was ist Jest?** Jest ist ein beliebtes JavaScript-Testframework, das entwickelt wurde, um Entwicklern eine effiziente und intuitive Testumgebung zu bieten. **Test Suites & Test Cases** Mit describe werden Test-Suites erstellt, die eine Gruppe von Testfällen (definiert mit test oder it) zusammenfassen, um eine logische Struktur zu bieten.**Assertions** Dies sind die tatsächlichen Überprüfungen in Tests. Mit Jest's expect Funktion kann man eine Vielzahl von Vergleichen und Überprüfungen durchführen, um das erwartete Verhalten zu validieren. **Lifecycle-Methoden** Methoden wie beforeAll, beforeEach, afterEach und afterAll ermöglichen das Festlegen von Setup- und Teardown-Aktionen, die vor oder nach Tests oder Test-Suites ausgeführt werden.**Mocking** Jest bietet Funktionen zum Erstellen von simulierten Funktionen (Mocks), mit denen das Verhalten von Abhängigkeiten kontrolliert und überprüft werden kann, um isolierte Tests zu gewährleisten. **Asynchrone Tests mit Jest** Jest unterstützt standardmässig das Testen von asynchronem Code, um beispielsweise der Zugriff auf eine Datenbank oder eine externe API zu simulieren und nachzustellen.**Best Practices** für Jest Um die Qualität und Zuverlässigkeit von Jest-Tests sicherzustellen, sollte man bewährte Methoden und Richtlinien befolgen. *Kleine, fokussierte Tests+Verwendung von beschreibenden Testnamen Mocken von externen+Abhängigkeiten Vermeidung globaler Zustände+Regelmäßige Testabdeckungsüberprüfung.* **Einführung in Code Reviews** Code Reviews spielen eine essentielle Rolle in der Softwareentwicklungspraxis. • **Was sind Code Reviews**: Ein systematischer Überblick über den Quellcode, um Fehler zu finden und die Codequalität zu verbessern. • **Warum sind sie wichtig:** Sie fördern Code-Qualität, verhindern Bugs und ermöglichen kontinuierliches Lernen innerhalb des Teams. • **Vorteile von Code Reviews**: Erhöhung der Code-Qualität, Wissenstransfer zwischen Teammitgliedern und frühzeitiges Aufdecken von Problemen. • **Gemeinsame Ziele**: Sicherstellen, dass der Code den Standards entspricht und potenzielle Fehler minimiert werden. Code Reviews bilden die Grundlage für qualitativ hochwertige Software und Teamzusammenarbeit.**Branches und ihr Zweck:** Unabhängige Arbeitskopien eines Projekts, die das parallele Arbeiten an unterschiedlichen Features oder Fixes erlauben. • **Commits und ihre Bedeutung**: Einzelaufzeichnungen von Änderungen im Code, die dokumentieren, was und von wem geändert wurde. **Ein Pull Request (PR)** ein Vorschlag zur Integration von Änderungen aus einem Branch in einen anderen, oft von einem Fork in das ursprüngliche Repository. • **Erstellung eines PRs**: Initiierung eines Vorschlags, Änderungen zu einem Projekt beizutragen, indem man die Änderungen zur Überprüfung und Diskussion stellt. • **Review-Prozess von PRs**: Andere Entwickler überprüfen den vorgeschlagenen Code, geben Feedback und diskutieren potenzielle Anpassungen. • **Merge eines PRs**: Nach der Überprüfung und Genehmigung werden die vorgeschlagenen Änderungen in das Ziel-Repository integriert. • Konfliktlösung: Wenn Änderungen in einem PR nicht direkt mit dem Zielbranch kompatibel sind, müssen diese Konflikte manuell gelöst werden.**Code Review Prozess auf Github** Der Code Review Prozess auf Github stellt sicher, dass CodeÄnderungen effektiv und qualitativ hochwertig sind. **Ablauf eines Code Reviews** Ein Entwickler schlägt Änderungen vor, diese werden von Teammitgliedern überprüft, Feedback wird gegeben, Änderungen werden vorgenommen und schließlich wird der Code integriert.**Best Practices für effektive Code Reviews** Effektive Code Reviews sind entscheidend für die Aufrechterhaltung hoher Code-Qualitätsstandards und eine funktionierende Teamdynamik. **Klare und konstruktive Kommentare**: Feedback sollte präzise und hilfreich sein, ohne persönlich oder kritisch zu wirken. Es geht darum, den Code und nicht den Autor zu bewerten. • **Fokus auf wesentliche Aspekte**: Statt sich auf kleinere Stilfragen zu konzentrieren, sollte der Schwerpunkt auf der Architektur, Logik und Performance des Codes liegen. • **Kontinuierliche Weiterbildung fördern:** Nutze Code Reviews als Möglichkeit zum Wissensaustausch und zur Förderung von Best Practices innerhalb des Teams. • **Nicht alles auf einmal erwarten:** Akzeptiere, dass nicht jeder PR perfekt sein wird. Ziel ist es, stetige Verbesserungen zu fördern und größere Fehler zu vermeiden. • **Zeitnahes Review**: Schnelles Feedback kann den Entwicklungsprozess beschleunigen und stellt sicher, dass Änderungen noch frisch im Gedächtnis sind.

Werkzeuge und Integrationen: Github Actions für automatisierte Tests: Ein CI/CD-Tool direkt von Github, das es ermöglicht, Workflows für das Testen, Bauen und Veröffentlichen von Software automatisch auszuführen. • **Linters:** Werkzeuge, die den Code automatisch auf Einhaltung bestimmter Stil- und Qualitätsstandards überprüfen. • **Code-Coverage-Tools:** Diese messen, welcher Anteil des Codes tatsächlich durch Tests abgedeckt wird, und helfen dabei, potenzielle Schwachstellen zu identifizieren. • **Integration von Drittanbietern:** Tools wie Slack, Jira oder Trello können in Github integriert werden, um die Kommunikation und das Projektmanagement zu erleichtern. • **Security-Scanner:** Automatische Tools, die den Code auf Sicherheitslücken und bekannte Schwachstellen überprüfen. **Umgang mit Feedbac**k Ein konstruktiver Umgang mit Feedback im Code Review Prozess fördert eine positive und produktive Teamkultur. • Objektivität bewahren: • Offenheit für andere Perspektiven: • Klare Kommunikation:Feedback annehmen und lernen• Dankbarkeit zeigen **Zusammenarbeit und Teamdynamik** Eine gesunde Teamdynamik und effektive Zusammenarbeit sind Schlüsselkomponenten für den Erfolg von Softwareprojekten. • Kommunikation ist der Schlüssel Feedback-Kultur pflegen• Konflikte proaktiv angehen: **Einführung in Clean Code** Clean Code ist die Kunst der Programmierung, die über das reine Schreiben von funktionierendem Code hinausgeht. • Definition• Wichtigkeit• Lesbarkeit• Wartbarkeit• Teamarbeit **Benennung von Variablen und Funktionen** Gute Namensgebung erleichtert das Verständnis des Codes und verringert die Notwendigkeit von Kommentaren. • **Klarheit**: Wähle Namen, die den Zweck oder die Funktion deutlich machen, z.B. calculateTotalPrice statt calcTP. • **Konsistenz**: Verwende durchgehend die gleichen Namenskonventionen, z.B. camelCase • **Vermeidung von Abkürzungen**: Es ist besser, userAccount zu schreiben als usrAcc, um Verwirrung zu vermeiden. • **Länge**: Namen sollten informativ sein, aber nicht zu lang. Ein Gleichgewicht finden zwischen Präzision und Kürze. • **Kontextbezogenheit**: Namen sollten im Kontext sinnvoll sein, z.B. width in einer Rectangle-Klasse anstelle von rectWidth.

Funktionen sind das Herzstück eines jeden Codes und sollten klar, kompakt und fokussiert sein. • **Einzelne Verantwortlichkeit**: Jede Funktion sollte nur eine Sache tun. Dies erleichtert das Verstehen, Testen und Warten des Codes. • **Kurze Länge**: Idealerweise sollte eine Funktion nicht länger als 20 Zeilen sein. Kurze Funktionen sind leichter zu verstehen und zu überprüfen. • **Keine Seiteneffekte:** Funktionen sollten keine verborgenen Effekte haben. Sie sollten nur Werte ändern, die sie als Parameter erhalten, und Rückgabewerte bereitstellen. • **Klare Parameter**: Verwende wenige, klare und beschreibende Parameter. Übermäßig viele Parameter können Anzeichen für eine überlastete Funktion sein. • **Beschreibende Namen:** Der Name einer Funktion sollte ihre Absicht klar kommunizieren, z.B. retrieveUserData() statt getData().

Kommentare: Während Kommentare hilfreich sein können, ist der beste Code der, der für sich selbst spricht. • **Wann zu kommentieren**: Wenn der Code komplexe Geschäftslogik oder spezielle Workarounds beinhaltet, die nicht sofort ersichtlich sind.**Formatierung und Strukturierung** Ein konsistent formatierter und gut strukturierter Code verbessert die Lesbarkeit und erleichtert die Zusammenarbeit. • **Einrückungen:** Verwende konsistente Einrückungsstile, z.B. 2 oder 4 Leerzeichen, um Blöcke zu definieren. • **Coding Style:** Halte dich an einen festgelegten Kodierungsstil, wie z.B. Airbnb's JavaScript-Styleguide oder Google's JavaScript-Styleguide. • **Gruppierung:** Gruppiere verwandte Funktionen und Variablen zusammen, um den Fluss des Codes zu verbessern. • **Klare Trennung:** Trenne verschiedene Aufgabenbereiche in verschiedene Module oder Dateien, um Wiederverwendbarkeit und Wartbarkeit zu verbessern. • **Konsistente Benennung**: Wie bei Variablen und Funktionen, sollte die Datei- und Ordnerbenennung klar und beschreibend sein. **Fehlerbehandlung** Eine robuste Fehlerbehandlung ist unerlässlich für stabile und zuverlässige Anwendungen. • **Antizipieren**: Erwarte mögliche Fehlerquellen und handle sie proaktiv ab, bevor sie auftreten. • **"try-catch"**: Verwende diese Blöcke, um Laufzeitfehler zu erkennen und angemessen darauf zu reagieren. • **Benutzerfreundliche Meldungen**: Statt technischer Fehlermeldungen, liefere klare, verständliche Hinweise für den Endbenutzer. • **Logging**: Protokolliere Fehler detailliert für Entwickler, um die Ursache schnell zu identifizieren und zu beheben. **Objekt- und Datenstrukturen** Gut durchdachte Strukturen sind das Rückgrat einer effizienten und wartbaren Anwendung. • **Einfache Datenstrukturen:** Verwende klare und intuitive Strukturen, wie Arrays oder Objekte, um Daten zu speichern und zu manipulieren. • **Dekomposition**: Teile komplexe Strukturen in kleinere, handhabbare Teile auf. • **Unveränderlichkeit**: Wo möglich, verwende unveränderliche Datenstrukturen, um Seiteneffekte und Dateninkonsistenzen zu verhindern. • **Zugriffssteuerung**: Stelle sicher, dass nur relevante Teile einer Struktur oder eines Objekts extern zugänglich sind. **Unit Tests und Testabdeckung** Unit Tests gewährleisten, dass der Code wie erwartet funktioniert und helfen, Fehler frühzeitig zu identifizieren. • **Grundlagen von Unit Tests**: Tests, die isoliert kleinste Code-Einheiten überprüfen, um ihre Funktionsfähigkeit sicherzustellen. • **Wichtigkeit**: Sie helfen, Fehler frühzeitig zu erkennen, fördern guten Code und erleichtern Refactorings. • **Frameworks**: Nutze Test-Frameworks wie Jest oder Mocha, um das Testen zu vereinfachen und zu automatisieren. • **Testabdeckung**: Ein Maß dafür, welcher Anteil des Codes durch Tests überprüft wird. Tools wie Istanbul können dies in JavaScript messen. • **TDD (Test Driven Development)**: Eine Methode, bei der zuerst Tests geschrieben und dann erst der Code entwickelt wird.

P**rinzipien des Clean Code** Die Prinzipien des Clean Code bieten Leitlinien, um qualitativ hochwertigen und wartbaren Code zu schreiben. • **DRY** (Don't Repeat Yourself): Vermeide Duplikationen und sorge für eine einzelne Informationsquelle im Code. • **KISS** (Keep It Simple, Stupid): Halte den Code einfach und geradlinig. Vermeide unnötige Komplexität. • **YAGNI** (You Ain't Gonna Need It): Schreibe keinen Code, den du aktuell nicht brauchst. Antizipiere nicht zu weit im Voraus. **Werkzeuge und Ressourcen** Die richtigen Werkzeuge und Ressourcen können den Weg zu Clean Code erheblich erleichtern. • **Linting-Tools**: Werkzeuge wie ESLint oder TSLint helfen, Code-Standards durchzusetzen und häufige Fehler zu identifizieren. • **Formatierer**: Prettier ist ein beliebter Code-Formatierer, der für Konsistenz in der Formatierung sorgt. • **Code Reviews**: Nutze Plattformen wie GitHub oder Bitbucket, um Code-Überprüfungen durchzuführen und Feedback von Kollegen zu erhalten. • **Dokumentation:** Ressourcen wie MDN (Mozilla Developer Network) bieten wertvolle Informationen und Best Practices für JavaScript.

Einführung in E2E-Testing Ein Testansatz, bei dem das gesamte Softwareprodukt in einer Umgebung getestet wird, die der tatsächlichen Nutzungsweise ähnelt. • **Zweck von E2E-Tests:** Sicherstellung, dass die Anwendung als Ganzes korrekt funktioniert und Benutzeranforderungen erfüllt werden. • U**nterschied zu anderen Testarten:** Während Unit-Tests einzelne Komponenten testen und Integrationstests die Verbindung zwischen Komponenten prüfen, simulieren E2E-Tests realistische Benutzerszenarien. • **Wichtige Anwendungsbereiche:** E2E-Tests sind besonders nützlich für kritische Workflows, wie etwa Bestellvorgänge oder Nutzerregistrierungen. • **Herausforderungen**: Sie erfordern oft komplexere Setups und können zeitaufwändiger sein als andere Testarten.**Übersicht über Cypress** Cypress ist ein modernes Frontend-Testwerkzeug für Webanwendungen, das Entwicklern ein schnelles, zuverlässiges und konsistentes Testing-Erlebnis bietet. • **Echtzeit-Feedback:** Tests werden automatisch neu geladen und ausgeführt, wenn Änderungen vorgenommen werden. • **Intuitiver Test Runner:** Eine benutzerfreundliche Oberfläche, die Tests in Echtzeit zeigt und bei Fehlern nützliche Informationen bereitstellt. • **Kein Asynchronitäts-Wirrwarr**: Durch automatisches Warten und Intelligente Queues reduziert Cypress flatternde Tests.**Cypress Test Runner** Das Herzstück von Cypress für Echtzeit-Tests und Debugging. • **Echtzeit-Feedbac**k: Tests aktualisieren sich automatisch bei CodeÄnderungen. • **Interaktives Debugging**: Tests pausieren und Schritt für Schritt durchlaufen. • **Fehlerdiagnose**: Klare Berichte, Screenshots bei Fehlern und Videoaufzeichnung. • **Erweiterbarkeit**: Anpassung mit Plugins und Erweiterungen.**Textstrukturierung** in Cypress Wie Sie Ihre Tests in Cypress effizient organisieren und strukturieren. • **Test-Suiten:** Gruppieren von verwandten Testfällen, um einen größeren Funktionsbereich abzudecken. • **Testfälle**: Schreiben von Einzelaufgaben, die ein bestimmtes Feature oder einen Workflow testen. • **Ordnerstruktur**: Nutzen des cypress/integration-Ordners für Tests und cypress/fixtures für Testdaten.**Best Practices für stabile Tests** Strategien in Cypress, um zuverlässige und konsistente Testergebnisse zu erzielen. • **Vermeidung von Flaky Tests:** Fokussieren Sie sich auf idempotente Aktionen und reduzieren Sie externe Abhängigkeiten. • **Assertions**: Nutzen Sie Assertions, um den Zustand und das Verhalten der Anwendung zu überprüfen. • **Testisolation**: Jeder Test sollte unabhängig sein und nicht auf den Ergebnissen eines anderen basieren.**Interaktion mit Web-Elementen** Die richtige Interaktion mit Web-Elementen ist entscheidend für realistische und effektive E2E-Tests. • **DOM-Zugriff**: Mit cy.get() Elemente auswählen und mit ihnen interagieren. • **Formulare**: Eingaben füllen mit cy.type() und Formulare absenden mit cy.submit(). • **Pop-ups & Dialoge**: Bestätigungsdialoge handhaben und Fensterwechsel erkennen. • **Dynamische Inhalte**: Mit cy.wait() auf das Laden von Inhalten warten oder cy.should() für Zustandsprüfungen nutzen.**CI/CD Integration** Die Integration von Cypress in CI/CD stellt sicher, dass Softwareänderungen kontinuierlich auf Qualität geprüft werden. • **Automatisierte Testausführung:** Cypress Tests als Teil des CI-Prozesses einrichten, um bei Codeänderungen automatisch zu laufen. • **Konfiguration**: Anpassen von cypress.json für CI-Umgebungen, z.B. unterschiedliche Basis-URLs oder Authentifizierungseinstellungen. • **Parallelisierung**: Mehrere Tests gleichzeitig in der CI-Umgebung ausführen, um die Testgeschwindigkeit zu erhöhen. • **Ergebnisberichte**: Integration von Testbericht-Generatoren für eine klare Übersicht der Testergebnisse.**Erweiterungen und Plugins** Erweitern Sie Cypress mit zusätzlichen Tools und Funktionen für verbesserte Testfähigkeiten. • **Community Plugins:** Nutzen Sie Plugins wie cypress-react-unit-test oder cypress-image-snapshot für spezifische Testanforderungen. • **Installation**: Plugins über npm installieren und in cypress/plugins/index.js konfigurieren. • **Custom Commands:** Erstellen Sie eigene Befehle mit Cypress.Commands.add(), um wiederkehrende Aufgaben zu vereinfachen. • **Dashboard Integration**: Nutzen Sie das Cypress Dashboard für erweiterte Analysen und Echtzeit-Feedback **Einführung in TDD** Ein Entwicklungsansatz, bei dem zuerst Tests geschrieben werden, bevor der eigentliche Code entsteht. • **Grundprinzipien**: **1**. Zuerst schreiben wir einen Test (RED) **2.** dann den Code, um den Test zu bestehen (GREEN) **3**. und schließlich wird der Code optimiert (Blue) • **Unterschied zu traditionellem Testen** • Bei TDD beginnt man mit dem Testen und gestaltet die Softwareentwicklung entsprechend, während traditionelles Testen oft nach der Entwicklung stattfindet. • **Ziel von TDD:** Förderung besserer Code-Qualität und einfacherer Wartbarkeit durch frühzeitiges Testen.**TDD-Regeln von Uncle Bob** Uncle Bob (Robert Cecil Martin) ist ein amerikanischer Softwareentwickler, Dozent und Autor. Er ist vor allem dafür bekannt, dass er zahlreiche Prinzipien des Softwaredesigns fördert und Autor und Unterzeichner des einflussreichen Agile Manifests ist. **Werke** • Clean Code • Clean Architecture • Agile Manifest **Grundlegende Schritte des TDD** TDD folgt einem bestimmten **Zyklus** • **Roter Test**: Zuerst wird ein Test geschrieben, der fehlschlägt, da die geforderte Funktion noch nicht implementiert ist. • **Grüner Test**: Nun wird der minimal notwendige Code geschrieben, um den Test zu bestehen. • **Refactoring**: Der Code wird verbessert, ohne dass seine Funktionalität verändert wird, um Sauberkeit und Effizienz zu gewährleisten. Wiederholung: Der Zyklus beginnt von neuem mit einem weiteren Test.**Vorteile von TDD** Der Ansatz des Test Driven Development in der Softwareentwicklung hat folgende Vorteile: • **Qualitätssicherung**: Durch das Schreiben von Tests vor dem Code werden Fehler frühzeitig erkannt und behoben. • **Vereinfachtes Refactoring**: Mit bestehenden Tests kann der Code ohne Angst vor unerwünschten Nebenwirkungen verbessert werden. • **Dokumentation durch Tests**: Tests dienen als lebende Dokumentation, die zeigt, wie der Code funktionieren soll. • **Verringertes Fehlerrisiko:** Regelmäßiges Testen minimiert die Wahrscheinlichkeit, dass Fehler in der Produktion auftreten. • **Förderung klarer Anforderungen:** Beim Schreiben von Tests werden unklare oder widersprüchliche Anforderungen oft frühzeitig identifiziert.Werkzeuge für TDD Folgende Tools können Test Driven Development in JavaScript unterstützen oder ermöglichen.•Jest •Mocha• Jasmine **Herausforderungen und Kritik** Folgende Schwierigkeiten und Bedenken können zu TDD angemerkt werden: • **Anfangsaufwand**: TDD erfordert anfänglich mehr Zeit, da zuerst Tests geschrieben werden müssen, bevor der eigentliche Code entwickelt wird. • **Lernkurve**: Es kann Zeit und Schulung erfordern, bis Entwickler den TDD-Prozess vollständig verinnerlichen. • **Mögliche Überkomplexität:** Manchmal können die Tests komplizierter werden als der eigentliche Code. • **False Sense of Security:** Nur weil Tests bestehen, bedeutet das nicht, dass es keine Fehler gibt. Es kann sein, dass nicht alle Fälle abgedeckt sind. • **Kritik am Wert:** Einige argumentieren, dass die Vorteile von TDD nicht die zusätzlichen Ressourcen rechtfertigen, die für seine Implementierung benötigt werden. **Mocking und Stubbing** Mocking sowie Stubbing ist im Bereich von TDD von grosser Bedeutung. **Definition von Mocking=** • Das Ersetzen eines realen Objekts durch ein simuliertes Objekt (Mock), um bestimmte Verhaltensweisen während des Tests zu überprüfen. Zum Beispiel API‘s oder Datenbanken. **Definition von Stubbing** • Das Ersetzen einer Funktion in einem Test durch eine "Stummel"-Version, die vorhersehbare Ergebnisse liefert, um andere Teile des Codes zu testen. • **Anwendungszwecke**: Mocks und Stubs werden verwendet, um externe Abhängigkeiten zu isolieren und die Testumgebung zu kontrollieren. • **Unterschiede**: Während Mocks das Verhalten überprüfen (z.B. ob eine Methode aufgerufen wurde), liefern Stubs feste Ergebnisse und prüfen nicht das Verhalten. • **Vorteile**: Sie erlauben es, Tests schneller, zuverlässiger und unabhängig von externen Faktoren durchzuführen.**Integration von TDD** in **den Entwicklungsprozess** Wie wird Test Driven Development nahtlos in den Softwareentwicklungszyklus integriert. • **Anforderungsanalyse**: Bevor der TDD-Prozess beginnt, ist es wichtig, klare Anforderungen zu definieren, die als Grundlage für Tests dienen. • **Testdesign**: Erstellen Sie zuerst Tests basierend auf den spezifizierten Anforderungen und Erwartungen. • **Entwicklungszyklus**: Im TDD-Zyklus (roter Test, grüner Test, Refaktorisierung) wird kontinuierlich entwickelt und getestet, um inkrementelle Fortschritte sicherzustellen. • **Dokumentation**: Da Tests auch als lebende Dokumentation dienen, sollten sie klar und verständlich sein und regelmäßig überprüft werden.**Ausblick in zukünftige Entwicklungen** Die Zukunft von TDD verspricht, spannend und innovativ zu sein, um den Anforderungen einer sich rasant entwickelnden Technologielandschaft gerecht zu werden. • **Automatisierte Testgenerierung:** Fortschritte in KI und ML könnten dazu führen, dass Tests automatisch basierend auf Anforderungen generiert werden. • **Erweiterte Integration in IDEs:** Integrierte Entwicklungsumgebungen könnten fortschrittliche TDD-Tools bieten, die Entwickler in Echtzeit bei der Testgenerierung unterstützen. • **Verfeinerte Metriken:** Zukünftige Tools könnten tiefere Einblicke und detailliertere Metriken zur Testabdeckung und Codequalität bieten. • **Verstärkter Fokus auf Sicherheit:** Angesichts wachsender Sicherheitsbedenken könnte TDD stärker auf Sicherheitstests und deren Integration in den Entwicklungszyklus abzielen.
Unit-Testing unterscheidet sich von anderen Testmethoden in Bezug auf den Umfang und den Fokus des Tests.**Jest wurde von Facebook eingeführt."Positiv"** Tests überprüfen, ob das System wie erwartet funktioniert, wenn es mit gültigen Eingabedaten gefüttert wird**. "Negativ"** Tests hingegen überprüfen, wie das System auf ungültige oder unerwartete Eingaben reagiert und ob es dabei angemessen fehlerresistent bleibt. **Code Coverage** ist ein Maß dafür, welcher Anteil des Codes durch Tests ausgeführt wird. Sie dient dazu: **wozu dient** Die Vollständigkeit von Tests zu bewerten und potenziell ungetestete Codebereiche zu identifizieren.**Welche Funktionen bietet Github als Plattform um Sie im Entwicklungsprozess zu unterstützen?Versionskontrolle:** Basierend auf Git ermöglicht GitHub das Verfolgen von Änderungen, das Erstellen von Branches und das Zusammenführen von Code. **Pull Requests, Code Reviews Ein «Commit»** in Git ist eine Momentaufnahme des aktuellen Zustands eines Projekts, die Änderungen am Code festhält. Commits bieten eine Historie der Entwicklung, ermöglichen das Rückgängigmachen von Änderungen **wozu dienen«Pull Request** PRs fördern die Kollaboration, indem sie eine strukturierte Plattform für Code Reviews bieten und sicherstellen, dass Änderungen qualitativ hochwertig und mit dem bestehenden Code kompatibel sind.**README-Dateien dienen daz**u, Informationen über ein Softwareprojekt bereitzustellen, einschließlich seiner Funktion, Installationsanweisungen und Nutzungshinweisen. **Worin sehen Sie den Vorteil, wenn Sie über einen «PR» die Änderungen von einem Branch in den anderen übertragen und nicht direkt lokal die Änderungen übertragen?**Es läuft somit über einen kontrollierten Prozess, welcher auch andere Entwickler beteiligen kann und somit über die jeweiligen Änderungen eine Diskussion geführt werden kann.**wozu diese Art des Codes schreiben von Vorteil sein kann.** Dieser Ansatz verbessert die langfristige Wartbarkeit, verringert Fehleranfälligkeit und erleichtert die Zusammenarbeit im Team.**Was versteht man unter Namenskonventionen, warum ist das in Zusammenhang mit Clean Code wichtig und welche wird in JavaScript meist angewendet?** Namenskonventionen sind vereinbarte Richtlinien für die Benennung von Programmierstrukturen wie Variablen, Funktionen und Klassen. Sie sind im Kontext von Clean Code wichtig, weil sie die Lesbarkeit und Verständlichkeit des Codes erhöhen und Missverständnisse vermeiden. In JavaScript wird häufig die camelCase-Konvention für Variablen und Funktionen und die PascalCase-Konvention für Klassen verwendet **Was versteht man unter Coding Styles oder Styleguides?**Coding Styles oder Styleguides sind Sammlungen von Konventionen und Richtlinien zur Formatierung und Strukturierung von Programmcode. Sie sorgen für eine einheitliche CodeDarstellung innerhalb eines Projekts oder Teams und erleichtern so die Lesbarkeit und Wartung.**Wie sollte man grundsätzlich mit Fehlern umgehen bezogen auf den Endbenutzer?**Bezogen auf Clean Code sollte man Fehler vor dem Endbenutzer möglichst abfangen und klar verständliche, nicht-technische Fehlermeldungen präsentieren, die dem Nutzer dabei helfen, das Problem zu verstehen.**welche Möglichkeit kennen Sie in JavaScript, um dieses Prinzip zu befolgen?** In JavaScript kann man das DRY-Prinzip befolgen, indem man Funktionen und Module erstellt, um wiederkehrende Aufgaben zu kapseln, und durch die Verwendung von Frameworks und Bibliotheken, die gängige Funktionen bereitstellen.**Blackbox-Tests:** Fokus auf die Funktionalität des Systems ohne Kenntnis des Codes (z. B. E2E-Tests, Integrationstests).**Whitebox-Tests:** Fokus auf die interne Implementierung des Codes, wobei der Quellcode bekannt ist (z. B. Unit-Tests). **Nachteile beim Einsatz von E2E Tests** Langsame Ausführung, Flakey Tests, Schwierig **Worum handelt es sich beim Cypress Test Runner und welche Funktionen erfüllt dieser?** Der Test Runner ist das Herz stück von Cypress. Es macht Tests in Echtzeit. Es aktualisiert die Tests automatisch bei Änderungen am Code, gibt uns klare berichte oder auch Screenshots bei Fehlern und ermöglicht es uns mit dem Interaktiven Debugger Schritt für Schritt durch die Tests zu gehen. **Was versteht man unter Flaky Tests?** Tests die nicht unabhängig sind und je nach dem ein anderes Resultat wiedergeben könnten.**warum ist Testisolation wichtig?.** Dies ist wichtig damit man gute, nachvollziehbare Tests hat und nicht Flaky Tests.**Warum ergibt es Sinn, E2E- Tests in den CI/CD Prozess einfliessen zu lassen?** So wird die Qualität der Software bei Änderungen immer getestet. **Welche Möglichkeiten gibt es mit Cypress, um die Funktionalität nach Belieben zu erweitern**? Community Plugins, Installationen über npm, Custom commands und die Integration des Dashboards **Welches Ziel soll mit TDD erreicht werden?** Fehlerreduktion: Bessere Codequalität Wartbarkeit
Darf in Test Driven Development Programmcode geschrieben werden für den es noch keinen Test gibt?: In Test-Driven Development (TDD) sollte kein Programmcode geschrieben werden, für den es noch keinen Test gibt.**Beschreiben Sie mindestens 3 Vorteile gegenüber der traditionellen Softwareentwicklung, wenn mit TDD entwickelt wird.** Höhere Codequalität Bessere Testabdeckung:: Schnelleres Feedback und leichteres Refactoring: **Was versteht man unter dem Begriff Mocking?**Mocking bezeichnet in der Softwareentwicklung eine Technik, bei der Scheinobjekte oder Ersatzobjekte (Mocks) verwendet werden, um die tatsächlichen Abhängigkeiten einer Komponente oder Funktion zu simulieren.**Was versteht man unter dem Begriff Stubbing?** Stubbing bezeichnet in der Softwareentwicklung eine Technik, bei der bestimmte Methoden oder Funktionen durch Ersatzmethoden (Stubs) ersetzt werden, um vordefinierte Rückgabewerte oder Verhalten zu simulieren.**Welche Art von Software sollte auf diese Art der Entwicklung umgesetzt werden und welche eher nicht?**TDD ist besonders gut für kritische, wartungsintensive Anwendungen geeignet, bei denen Zuverlässigkeit und Langfristigkeit entscheidend sind. Es eignet sich weniger für Projekte, die schnell iteriert werden müssen, stark UI-zentriert sind oder bei denen die Anforderungen unklar und in ständiger Bewegung sind.