Einführung

SWEBOK

SWEBOK ist die Abkürzung für Software Engineering Body of Knowledge. Es handelt sich dabei um eine Sammlung von Wissen, das für die Ausübung des Software Engineerings notwendig ist. Inhalt:

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Engineering Tools and Methods
- Software Quality
- Related Disciplines of Software Engineering

Wasserfall, V-Model, Rational Unified Process

Wasserfallmodell

Vorteile:

- Einfach zu verstehen
- Strukturiert
- Klare Phasen
- Vorteilhaft bei Projekten mit klaren Anforderungen
- Erlaubt eine genauere Kosten- und Zeitplanung

Nachteile:

- Keine Anpassung an sich ändernde Anforderungen
- Lange Entwicklungszeiten
- Hohe Kosten bei späteren Änderungen

Rational Unified Process

Vorteile:

- Iterativ und inkrementell
- Architekturzentriert
- Risiko- und qualitätsorientiert

Nachteile:

- Schwerfällig bezüglich Prozess und Dokumentation
- Hoher Aufwand
- Transition-Phase fehlt

Wann sinnvoll?

- Bei Projekten die gross und komplex sind
- Bei Projekten in regulierten Branchen
- Bei Projekten die auf wiederholbaren Prozessen basieren
- Bei Projekten mit fixen Anforderungen (Kosten, Zeit, Funktionalität)

Probleme von Software

- Die Rate an fehlgeschlagenen Projekten ist hoch
- Die meisten der implementierten Features werden nie benutzt (64%)

Gesetze der Softwareentwicklung

Humphy's Law

Menschen wissen nicht, was sie wollen, bis sie es sehen. Bedeutet dass Anforderungen sich ändern, sobald sie implementiert sind.

Ziv's Law

Softwareentwicklung ist unvorhersehbar und kann nie vollends verstanden werden.

Conway's Law

Software ist ein Spiegel der Firma und der Menschen, die sie entwickeln.

Brooks' Law

Adding manpower to a late software project makes it later.

Zawinski's Law

Every program attempts to expand until it can read mail. Those programs which cannot so expand are replaced by ones which can.

Parkinson's Law

Work expands so as to fill the time available for its completion.

Pareto's Fallacy

When you're 80% done, you think you only have 20% left.

Sturgeon's Revelation

90% of everything is crud.

The Peter Principle

In a hierarchy, every employee tends to rise to his level of incompetence. Thus, in time, every post tends to be occupied by an employee who is incompetent to carry out its duties.

Eagelson's Law

Any code of your own that you haven't looked at for six or more months might as well have been written by someone else.

Greenspun's Tenth Rule

Any custom developed system contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of the industry standard you refused to adopt.

The Iceberg Fallacy

The cost of development of a new system is only a small part of the total cost of ownership.

Agile Manifesto

Agile Werte

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

12 Prinzipein des Agilen Manifests

- Customer satisfaction through early and continuous software delivery
- Accommodate changing requirements throughout the development process
- Deliver working software frequently (weeks rather than months)
- Collaboration between the business stakeholders and developers throughout the project
- Support, trust, and motivate the people involved
- Enable face-to-face interactions
- Working software is the primary measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Self-organizing teams
- Regular reflections on how to become more effective

Bedeutung des agilen Manifests für die Softwareentwicklung

Agile Methoden sind eine Reaktion auf die Probleme der klassischen Softwareentwicklung. Sie sind darauf ausgelegt, flexibel auf sich ändernde Anforderungen zu reagieren und die Zusammenarbeit zwischen den Stakeholdern zu verbessern. Agile Methoden sind besonders geeignet für Projekte, bei denen die Anforderungen nicht vollständig bekannt sind oder sich während der Entwicklung ändern können.

Agile

Begriffe

- Risk: Risiken sind Ereignisse, die den Erfolg eines Projekts gefährden können.
- Cost of change: Die Kosten, die entstehen, wenn Anforderungen geändert werden.
- Iron Triangle: Scope, Resources, Schedule; in der Mitte ist Quality.
  - Zusätzlichen Scope führt zu mehr Zeit
  - Zusätzliche Qualität führt zu mehr Zeit
  - Zusätzliche Ressourcen führen manchmal zu weniger Zeit, aber sehr selten
- Scope ist die einzige Variable

Extreme Programming

Begriffe

- Core Values: Communication, Simplicity, Feedback, Courage, Respect
- Principles: Rapid Feedback, Assume Simplicity, Incremental Change, Embracing Change, Quality Work
- Practices: Planning Game, Small Releases, Metaphor, Simple Design, Testing, Refactoring, Pair Programming, Collective Code Ownership, Continuous Integration, 40-hour Week, On-site Customer, Coding Standards, Sustainable Pace
- Cost of Change Curve: Die Kosten für Änderungen steigen exponentiell mit der Zeit, bei XP sind die Kosten konstant
- Business Value: Der Mehrwert, den ein Feature für den Kunden bringt

Software Craft

Gründe für das Software Craft Manifest

- Mangel an Qualität und Professionalität in der Softwareentwicklung
- Technische Schulden
- Fehlender Fokus auf kontinuierliche Verbesserung
- Fehlende Zusammenarbeit zwischen Entwicklern und anderen Stakeholdern

Formate der Software Craft Community

- Coding Dojos: Treffen, bei denen Entwickler zusammenarbeiten, um ihre Fähigkeiten zu verbessern
- Code Katas: Übungen, bei denen Entwickler eine bestimmte Aufgabe lösen
- Pair Programming: Zwei Entwickler arbeiten zusammen an einem Computer und wechseln sich ab
- Mob Programming: Ein Team arbeitet zusammen an einem Computer
- Workshops und Meetups: Veranstaltungen, bei denen Entwickler zusammenkommen, um über Software Craft zu diskutieren
- Code Retreats: Intensive Workshops, die sich auf das Üben und Verbessern von Softwareentwicklungsfähigkeiten konzentrieren

Prinzipien des Software Craft Manifests

- Professionelles Verhalten
- Kontinuierliche Verbesserung
- Zusammenarbeit und Austausch
- Qualität im Vordergrund
- Reflexion und Verbesserung

Coding Dojo und Code Kata

Ein Coding Dojo ist ein kollaboratives Lernformat, bei dem Entwickler in einer lockeren Umgebung Programmieraufgaben lösen. Die Struktur eines Coding Dojos umfasst:

- Kata Auswahl
- Timeboxing
- Rotationen
- Reflexion

Eine Code Kata ist eine Übung, bei der Entwickler eine bestimmte Programmieraufgabe lösen. Code Katas sind eine Möglichkeit, Programmierfähigkeiten zu üben und zu verbessern.

Pyramid of Agile Competences

Die Pyramid of Agile Competences ist ein Modell, das die verschiedenen Kompetenzen beschreibt, die für die agile Softwareentwicklung erforderlich sind. Die Pyramide besteht aus drei Ebenen:

- Engineering Practices: Technische Fähigkeiten, die für die Softwareentwicklung erforderlich sind, wie z.B. Test-Driven Development, Refactoring und Continuous Integration.
- Management Practices: Managementfähigkeiten, die für die agile Softwareentwicklung erforderlich sind, wie z.B. Scrum, Kanban und Lean.
- Agile Values: Die Werte und Prinzipien des agilen Manifests, wie z.B. individuelle Interaktionen über Prozesse und Tools, funktionierende Software über umfassende Dokumentation und kontinuierliche Verbesserung.

User Stories

Ziele und Anwendung einer User Story

Ziele

- **Kundenorientierung**: User Stories stellen sicher, dass die Entwicklung auf die Bedürfnisse und Anforderungen der Endbenutzer ausgerichtet ist.
- **Kommunikation und Verständnis**: Sie fördern den Dialog zwischen Entwicklern, Kunden und anderen Stakeholdern, um ein gemeinsames Verständnis zu schaffen.
- **Fokus auf Mehrwert**: User Stories helfen dabei, den Mehrwert für den Benutzer zu identifizieren und zu priorisieren.
- **Iterative Entwicklung**: Sie unterstützen die iterative und inkrementelle Entwicklung, indem sie kleine, umsetzbare Einheiten liefern.
- **Flexibilität**: User Stories sind flexibel und können leicht angepasst werden, wenn sich Anforderungen ändern.

Anwendung

- **Anforderungsdefinition**: User Stories werden genutzt, um die Anforderungen aus Sicht des Endbenutzers zu beschreiben.
- **Planung und Priorisierung**: Sie dienen zur Planung und Priorisierung im Product Backlog.
- **Implementierung**: Entwickler verwenden User Stories, um spezifische Funktionen oder Features zu implementieren.
- **Akzeptanztests**: Akzeptanzkriterien in User Stories helfen bei der Definition der Tests, die sicherstellen, dass die Anforderungen erfüllt sind.

Aufbau einer User Story

Eine User Story ist typischerweise in einem einfachen Format geschrieben, das leicht verständlich ist. Ein gängiges Format lautet:

> **Als** [Rolle] **möchte ich** [Funktion], **damit** [Nutzen].

Beispiel:

> **Als** Benutzer **möchte ich** mich mit meiner E-Mail-Adresse und einem Passwort anmelden, **damit** ich auf meine persönlichen Daten zugreifen kann.

Komponenten einer User Story

- **Titel**: Kurze und prägnante Beschreibung.
- **Beschreibung**: Detaillierte Beschreibung im oben genannten Format.
- **Akzeptanzkriterien**: Bedingungen, die erfüllt sein müssen, damit die User Story als erledigt betrachtet wird.
- **Priorität**: Die Wichtigkeit der User Story im Kontext des gesamten Projekts.
- **Schätzung**: Der geschätzte Aufwand, um die User Story umzusetzen.

Zusammenhang zwischen Epics, Themes und Stories

- **Themes**: Ein übergeordnetes Thema oder eine Kategorie, die mehrere Epics und User Stories umfasst. Sie helfen, das Product Backlog in größere Bereiche zu unterteilen.
- **Epics**: Große User Stories, die zu umfangreich sind, um in einem einzigen Sprint umgesetzt zu werden. Sie werden in mehrere kleinere User Stories aufgeteilt.
- **User Stories**: Kleine, spezifische Anforderungen, die aus Sicht des Endbenutzers formuliert sind und in einem einzigen Sprint umgesetzt werden können.

Merkmale einer guten User Story

Eine gute User Story sollte die **INVEST**-Kriterien erfüllen:

- **Independent (Unabhängig)**: Sie sollte unabhängig von anderen User Stories sein, um Flexibilität und Priorisierung zu ermöglichen.
- **Negotiable (Verhandelbar)**: Sie sollte nicht zu detailliert sein und Raum für Diskussionen und Anpassungen bieten.
- **Valuable (Wertvoll)**: Sie sollte einen erkennbaren Nutzen für den Endbenutzer liefern.
- **Estimable (Schätzbar)**: Der Aufwand zur Umsetzung sollte abschätzbar sein.
- **Small (Klein)**: Sie sollte klein genug sein, um in einem Sprint abgeschlossen zu werden.
- **Testable (Testbar)**: Sie sollte klare Akzeptanzkriterien haben, die getestet werden können, um festzustellen, ob die Anforderungen erfüllt sind.

Estimation and Planning

Grundbegriffe

User Stories

User Stories sind kurze, einfache Beschreibungen einer Funktion aus der Perspektive des Endbenutzers. Sie folgen dem Format: *Als [Rolle] möchte ich [Funktion], damit [Nutzen].*

User Roles

User Roles definieren verschiedene Arten von Benutzern, die unterschiedliche Anforderungen und Bedürfnisse haben. Beispiele sind *Administrator*, *Endbenutzer* oder *Gast*.

Epics

Epics sind große User Stories, die zu umfangreich sind, um in einem einzigen Sprint umgesetzt zu werden. Sie werden in kleinere, detailliertere User Stories aufgeteilt.

Themes

Themes sind übergeordnete Kategorien, die mehrere Epics und User Stories umfassen. Sie helfen dabei, das Product Backlog in größere Bereiche zu unterteilen.

Story Points

Story Points sind eine Maßeinheit zur Schätzung des Aufwands, der zur Umsetzung einer User Story benötigt wird. Sie basieren auf relativen Bewertungen der Komplexität, des Risikos und des Umfangs.

Velocity

Velocity ist die Geschwindigkeit, mit der ein Team User Stories in einem Sprint abschließt. Sie wird in Story Points gemessen und hilft bei der Planung zukünftiger Sprints.

Planning Poker

Planning Poker ist eine Methode zur Schätzung von User Stories, bei der Teammitglieder Karten mit Schätzungen verdeckt legen und gleichzeitig aufdecken. Unterschiede werden diskutiert, um eine Einigung zu erzielen.

Conditions of Satisfaction

Conditions of Satisfaction sind klare, messbare Kriterien, die erfüllt sein müssen, damit eine User Story als abgeschlossen betrachtet wird. Sie dienen als Akzeptanzkriterien.

Levels of Planning

Levels of Planning umfassen verschiedene Planungsebenen im agilen Kontext, von der strategischen Planung auf Unternehmensebene bis zur täglichen Planung auf Teamebene.

Product Backlog

Das Product Backlog ist eine priorisierte Liste von Anforderungen und Funktionen, die in Form von User Stories, Epics und Themes dargestellt werden. Es wird kontinuierlich aktualisiert und priorisiert.

Priorisierung (der User Stories)

Priorisierung bedeutet, die Reihenfolge der User Stories im Product Backlog festzulegen, basierend auf ihrem Wert für den Kunden, den Kosten, dem Risiko und anderen Faktoren.

Techniques for Estimating

Techniques for Estimating umfassen Methoden wie Planning Poker, T-Shirt-Größen, und Dot-Voting, um den Aufwand und die Komplexität von User Stories zu schätzen.

Relative Schätzung

Relative Schätzung vergleicht den Aufwand für verschiedene User Stories relativ zueinander, anstatt absolute Zahlen zu verwenden. Dies erleichtert die Bewertung der Komplexität.

Weitere Begriffe

Planning for Value

Planning for Value konzentriert sich darauf, den größtmöglichen Nutzen für den Kunden zu liefern, indem Funktionen priorisiert werden, die den höchsten Mehrwert bieten.

Cost

Cost bezieht sich auf die Ressourcen, einschließlich Zeit und Geld, die benötigt werden, um eine User Story oder ein Projekt umzusetzen.

Financial Value

Financial Value misst den finanziellen Nutzen, den eine Funktion oder ein Projekt für das Unternehmen generiert, z.B. durch Umsatzsteigerung oder Kostensenkung.

Risk

Risk umfasst die Unsicherheiten und potenziellen Probleme, die die Umsetzung einer User Story oder eines Projekts beeinflussen können. Risiken müssen identifiziert und gemanagt werden.

New Knowledge

New Knowledge bezieht sich auf das neue Wissen und die Erkenntnisse, die während der Umsetzung eines Projekts gewonnen werden und zur Verbesserung zukünftiger Projekte beitragen.

Kano-Model of Customer Satisfaction

Das Kano-Model of Customer Satisfaction kategorisiert Kundenanforderungen in Basis-, Leistungs- und Begeisterungsmerkmale, um die Kundenzufriedenheit zu maximieren.

Planungsebenen im agilen Kontext

Strategische Planung

Strategische Planung auf Unternehmensebene setzt langfristige Ziele und Visionen. Zuständig sind die Geschäftsführung und das obere Management.

Taktische Planung

Taktische Planung auf Programmebene definiert mittelfristige Ziele und koordiniert mehrere Teams. Verantwortlich sind Programm-Manager und Product Owner.

Operative Planung

Operative Planung auf Teamebene umfasst die Planung einzelner Sprints und die tägliche Arbeit. Zuständig sind die Scrum Master und die Entwicklungsteams.

Build Automation, CI/CD, DevOps

Arten der Software Automation

On-Demand Automation

On-Demand Automation wird manuell vom Entwickler oder einem anderen Benutzer ausgelöst. Beispiele sind das manuelle Starten eines Build-Prozesses oder das Ausführen eines spezifischen Tests.

Scheduled Automation

Scheduled Automation wird zu festgelegten Zeiten oder Intervallen automatisch ausgeführt. Dies kann tägliche Builds, nächtliche Tests oder wöchentliche Berichte umfassen.

Triggered Automation

Triggered Automation wird durch ein Ereignis ausgelöst, wie z.B. das Einchecken von Code in ein Versionskontrollsystem oder das Erreichen eines bestimmten Meilensteins im Projekt.

Arten der Automation und deren Ziele

Build Automation

Build Automation automatisiert den Prozess des Kompilierens und Linkens von Quellcode zu ausführbaren Programmen. Ziel ist es, Konsistenz und Effizienz zu gewährleisten.

Test Automation

Test Automation umfasst die automatisierte Ausführung von Tests, um sicherzustellen, dass der Code korrekt funktioniert. Ziel ist es, die Testabdeckung zu erhöhen und die Qualität zu verbessern.

Deployment Automation

Deployment Automation automatisiert das Bereitstellen von Anwendungen in verschiedenen Umgebungen (z.B. Entwicklungs-, Test- und Produktionsumgebung). Ziel ist es, den Bereitstellungsprozess zu beschleunigen und Fehler zu minimieren.

Software Automation Pipeline

Die Software Automation Pipeline umfasst mehrere Schritte, die von der Code-Erstellung bis zur Bereitstellung in der Produktionsumgebung reichen. Jeder Schritt hat spezifische Ziele und Aufgaben.

Code Commit

Der Prozess beginnt mit dem Einchecken von Code in ein Versionskontrollsystem. Dies löst oft automatisch den nächsten Schritt in der Pipeline aus.

Build

In dieser Phase wird der Code kompiliert und zu einem ausführbaren Artefakt zusammengefügt. Build-Tools und Skripte automatisieren diesen Prozess.

Unit Testing

Nach dem Build werden Unit-Tests ausgeführt, um sicherzustellen, dass einzelne Codeeinheiten korrekt funktionieren. Diese Tests sind schnell und decken kleine Codeabschnitte ab.

Integration Testing

Integrationstests prüfen, ob verschiedene Module des Codes zusammenarbeiten. Diese Tests sind umfassender und dauern länger als Unit-Tests.

Deployment

Nach erfolgreichem Testen wird die Anwendung in eine Testumgebung oder direkt in die Produktionsumgebung bereitgestellt. Deployment-Tools und Skripte automatisieren diesen Schritt.

Monitoring

Nach dem Deployment wird die Anwendung überwacht, um sicherzustellen, dass sie korrekt funktioniert und um eventuelle Probleme frühzeitig zu erkennen.

Continuous Integration (CI)

Continuous Integration ist eine Praxis, bei der Entwickler ihren Code häufig (mehrmals täglich) in ein gemeinsames Repository integrieren. Jeder Commit löst automatische Builds und Tests aus. Ziel ist es, Integrationsprobleme frühzeitig zu erkennen und zu beheben.

Continuous Delivery (CD)

Continuous Delivery erweitert CI, indem sichergestellt wird, dass der Code jederzeit in die Produktionsumgebung bereitgestellt werden kann. Dies erfordert automatisierte Tests und Deployment-Prozesse. Ziel ist es, die Bereitstellung von Software schnell und zuverlässig zu machen.

DevOps

DevOps ist eine Kultur und Praxis, die Entwicklung (Dev) und Betrieb (Ops) integriert, um die Zusammenarbeit zu verbessern und die Softwarebereitstellung zu beschleunigen. Es umfasst Techniken wie CI, CD und Infrastructure as Code (IaC).

Ziele von DevOps

- **Schnellere Bereitstellung**: Verkürzung der Zeit von der Entwicklung bis zur Produktion.
- **Höhere Qualität**: Verbesserte Softwarequalität durch automatisiertes Testen und Überwachen.
- **Bessere Zusammenarbeit**: Förderung der Zusammenarbeit zwischen Entwicklungs- und Betriebsteams.
- **Erhöhte Effizienz**: Automatisierung wiederkehrender Aufgaben zur Reduzierung manueller Arbeit.

Scrum

Charakteristiken von Scrum

Scrum ist ein agiles Rahmenwerk zur Entwicklung und Lieferung komplexer Produkte. Es basiert auf iterativer und inkrementeller Entwicklung und fördert Transparenz, Inspektion und Anpassung.

Roles (Scrum Team)

- **Product Owner**: Verantwortlich für das Product Backlog und maximiert den Wert des Produkts. Er stellt sicher, dass das Team an den wichtigsten Aufgaben arbeitet.
- **Scrum Master**: Unterstützt das Team dabei, Scrum zu verstehen und umzusetzen. Entfernt Hindernisse und fördert die Selbstorganisation des Teams.
- **Development Team**: Ein interdisziplinäres Team von Fachleuten, das an der Umsetzung der Product Backlog-Einträge arbeitet. Es organisiert sich selbst und bestimmt, wie die Arbeit am besten erledigt wird.

Ceremonies (Scrum Events)

- **Sprint**: Eine feste Zeitspanne (meist 2-4 Wochen), in der ein inkrementelles Produkt erstellt wird.
- **Sprint Planning**: Ein Meeting, in dem das Team das Ziel und die Arbeit für den nächsten Sprint plant.
- **Daily Scrum**: Ein tägliches, kurzes Meeting (max. 15 Minuten), bei dem das Team den Fortschritt bespricht und den Plan für den Tag erstellt.
- **Sprint Review**: Ein Meeting am Ende des Sprints, in dem das Team das Inkrement präsentiert und Feedback einholt.
- **Sprint Retrospective**: Ein Meeting, in dem das Team reflektiert, was gut lief und was verbessert werden kann, um kontinuierlich zu lernen und sich zu verbessern.

Scrum Artifacts

- **Product Backlog**: Eine priorisierte Liste von Anforderungen und Funktionen, die das Produkt benötigt.
- **Sprint Backlog**: Eine Liste von Aufgaben, die im aktuellen Sprint erledigt werden sollen.
- **Increment**: Das fertige, potenziell auslieferbare Produkt, das am Ende jedes Sprints erstellt wird.

Scrum Values

- **Commitment (Engagement)**: Das Team verpflichtet sich, die Ziele des Sprints zu erreichen.
- **Courage (Mut)**: Das Team hat den Mut, die richtigen Entscheidungen zu treffen und sich Herausforderungen zu stellen.
- **Focus (Fokus)**: Das Team konzentriert sich auf die Arbeit im Sprint und die Ziele des Scrum Teams.
- **Openness (Offenheit)**: Das Team ist offen für Feedback und neue Ideen.
- **Respect (Respekt)**: Teammitglieder respektieren einander und erkennen die unterschiedlichen Fähigkeiten und Erfahrungen an.

Wichtige Begriffe

Sprint

Ein Sprint ist eine Zeitspanne von maximal einem Monat, in der ein fertiges, nutzbares und potenziell auslieferbares Produktinkrement erstellt wird.

Sprint Goal

Das Sprint Goal ist das Ziel, das während des Sprints erreicht werden soll. Es gibt dem Entwicklungsteam Orientierung und Motivation.

Retrospective

Die Sprint Retrospective ist ein Meeting am Ende eines Sprints, in dem das Team diskutiert, was gut gelaufen ist, was verbessert werden kann und wie zukünftige Sprints effizienter gestaltet werden können.

Task Board

Ein Task Board ist ein visuelles Werkzeug, das den Fortschritt der Aufgaben im Sprint Backlog darstellt. Es enthält typischerweise Spalten wie *To Do*, *In Progress* und *Done*.

Burndown Chart

Ein Burndown Chart ist ein Diagramm, das zeigt, wie viel Arbeit im Sprint noch zu erledigen ist. Es hilft dem Team, den Fortschritt zu überwachen und zu steuern.

Definition of Done

Die Definition of Done (DoD) ist eine Liste von Kriterien, die erfüllt sein müssen, damit eine User Story oder ein Produktinkrement als fertig betrachtet wird.

Definition of Ready

Die Definition of Ready (DoR) ist eine Liste von Kriterien, die eine User Story erfüllen muss, bevor das Entwicklungsteam sie in einen Sprint ziehen kann.

Daily Scrum

Das Daily Scrum ist ein tägliches, 15-minütiges Meeting, bei dem das Team den Fortschritt bespricht und den Plan für die nächsten 24 Stunden erstellt.

Increment

Ein Increment ist das fertige Produkt oder der Produktteil, der am Ende eines Sprints geliefert wird. Es muss in einem nutzbaren Zustand und potenziell auslieferbar sein.

Architektur Patterns

CQRS (Command Query Responsibility Segregation)

CQRS ist ein Architekturpattern, das Lese- und Schreiboperationen voneinander trennt. Dadurch können die jeweiligen Anforderungen besser erfüllt und die Leistung optimiert werden. Schreiboperationen (Commands) ändern den Zustand des Systems, während Leseoperationen (Queries) den Zustand des Systems abfragen.

Event Sourcing

Event Sourcing speichert den Zustand eines Systems als eine Sequenz von Ereignissen (Events) statt als aktuelle Werte. Jede Änderung am Zustand wird als Ereignis gespeichert. Dies ermöglicht die Wiederherstellung des Zustands durch Wiedergabe der Ereignisse und unterstützt historische Analysen und Debugging.

Strangler Pattern

Das Strangler Pattern ist eine Migrationsstrategie, bei der neue Funktionen als separate Module neben einem bestehenden System implementiert werden. Allmählich wird das alte System stranguliert, bis es vollständig ersetzt ist. Dies ermöglicht eine inkrementelle Modernisierung.

Online-Migration

Online-Migration ist die Strategie, Daten und Anwendungen im laufenden Betrieb von einem System auf ein anderes zu migrieren, ohne den Service zu unterbrechen. Dies erfordert sorgfältige Planung und Werkzeuge, um Konsistenz und Verfügbarkeit zu gewährleisten.

Circuit Breaker

Ein Circuit Breaker schützt ein System vor Überlastung und Ausfällen, indem es den Zugang zu einer fehlerhaften Komponente blockiert. Es schaltet den Stromkreisäb, wenn eine bestimmte Anzahl von Fehlern auftritt, und versucht nach einer Wartezeit automatisch eine Wiederherstellung.

Bulkhead (Schottwand)

Das Bulkhead-Pattern isoliert verschiedene Teile eines Systems, um den Ausfall eines Teils vom Rest des Systems zu trennen. Dies verhindert, dass ein Fehler in einem Teil des Systems andere Teile beeinträchtigt, ähnlich wie Schotten in einem Schiff, die das Eindringen von Wasser begrenzen.

Retry

Das Retry-Pattern versucht, fehlgeschlagene Operationen nach einer bestimmten Wartezeit automatisch erneut auszuführen. Dies ist nützlich in Systemen, in denen temporäre Fehler häufig sind und oft durch Wiederholung behoben werden können.

Serverless

Serverless ist ein Architekturstil, bei dem der Cloud-Anbieter die Serverinfrastruktur verwaltet und der Entwickler sich auf das Schreiben von Funktionen konzentriert, die als Reaktion auf Ereignisse ausgeführt werden. Dies reduziert die Betriebs- und Wartungskosten und skaliert automatisch.

Microservices

Microservices ist ein Architekturmuster, bei dem eine Anwendung in kleine, unabhängige Dienste aufgeteilt wird, die jeweils eine spezifische Funktionalität bieten. Diese Dienste können unabhängig entwickelt, bereitgestellt und skaliert werden, was Agilität und Resilienz erhöht.

Self-contained Systems

Self-contained Systems (SCS) sind eine Variante der Microservices, bei der jede Einheit eine vollständige Geschäftsfunktion abdeckt, einschließlich Benutzeroberfläche, Logik und Datenhaltung. SCS fördert die Unabhängigkeit und Modularität, indem jede Einheit in sich abgeschlossen ist.

Monolith (Modulith)

Ein Monolith ist eine Softwarearchitektur, bei der alle Funktionen und Komponenten einer Anwendung in einer einzigen, einheitlichen Codebasis enthalten sind. Ein Modulith ist ein modularer Monolith, bei dem der Monolith in gut strukturierte, lose gekoppelte Module unterteilt ist, um Wartbarkeit und Erweiterbarkeit zu verbessern.

Software-Architektur

Definition

Software-Architektur bezeichnet die grundlegende Struktur eines Softwaresystems, bestehend aus seinen Komponenten, deren Beziehungen zueinander und den Prinzipien, die dessen Design und Evolution leiten.

Rolle des Software-Architekten

Ein **Software-Architekt** ist eine Person, die für das Design und die Struktur eines Softwaresystems verantwortlich ist. Diese Rolle beinhaltet die Definition von Architekturentscheidungen, die Sicherstellung der Übereinstimmung mit den Geschäftsanforderungen und die Unterstützung des Entwicklungsteams bei der Implementierung der Architektur.

Unterschied zwischen Enterprise Architecture und Application Architecture

Enterprise Architecture

Enterprise Architecture definiert, wie ein Unternehmen viele Anwendungen nutzt. Sie ist vergleichbar mit der Stadtplanung, bei der es um die Organisation und Integration von IT-Systemen im gesamten Unternehmen geht.

Application Architecture

Application Architecture konzentriert sich auf die Struktur einer einzelnen Anwendung. Sie umfasst Komponenten, Schichten, Pakete und Namespaces, um die Organisation des Codes zu beschreiben. Sie ist vergleichbar mit der Architektur eines Gebäudes.

Schwierigkeiten des Software Designs

- **Complexity**: Die Komplexität großer Systeme ist schwer zu beherrschen.
- **Conformity**: Software muss oft mit bestehenden Systemen und Standards konform sein.
- **Changeability**: Software ist ständigem Wandel unterworfen und muss flexibel auf Änderungen reagieren können.
- **Invisibility**: Software ist immateriell und schwer visuell darstellbar, was das Verständnis und die Kommunikation erschwert.

Architectural Drivers

Architectural Drivers sind die zentralen Einflüsse auf die Architektur eines Systems. Diese können technische, geschäftliche, soziale oder organisatorische Faktoren sein, die die Architekturentscheidungen beeinflussen. Beispiele sind:

- **Qualitätsanforderungen** wie Leistung, Sicherheit und Skalierbarkeit.
- **Einschränkungen** wie Budget, Zeitplan und verfügbare Ressourcen.

Standard, Style und Pattern

Standard

Ein **Standard** ist ein festgelegter Satz von Regeln und Richtlinien, die bei der Softwareentwicklung befolgt werden sollen, um Konsistenz und Qualität zu gewährleisten.

Style

Ein **Architekturstil** ist eine abstrakte Beschreibung einer Familie von Architekturen, die nach einem gemeinsamen Muster organisiert sind, z. B. Schichtenarchitektur oder Microservices.

Pattern

Ein **Pattern** ist ein bewährtes Lösungsmuster für ein wiederkehrendes Problem in einem bestimmten Kontext, z. B. Singleton oder Factory Pattern in der Softwareentwicklung.

Unterschied zwischen Layer und Tier

Layer

Ein **Layer** ist eine logische Trennung innerhalb der Software, um Funktionalitäten zu organisieren und zu modularisieren. Ein Layer kann in derselben oder in verschiedenen physischen Maschinen existieren.

Tier

Ein **Tier** ist eine physische Trennung in einem verteilten System, wobei jede Tier auf einer eigenen Maschine oder einem eigenen Server ausgeführt wird, z. B. Präsentations-Tier, Logik-Tier, Datenbank-Tier.

Architekturstile: Vor- und Nachteile

Independent Components

- **Vorteile**: Hohe Modularität und Austauschbarkeit.
- **Nachteile**: Komplexität der Nachrichtenverwaltung.

Call-and-Return

- **Vorteile**: Einfachheit und direkte Kommunikation.
- **Nachteile**: Eingeschränkte Flexibilität.

Virtual Machine

- **Vorteile**: Plattformunabhängigkeit.
- **Nachteile**: Leistungseinbußen durch Interpretation.

Data Flow

- **Vorteile**: Einfach zu verstehen und zu testen.
- **Nachteile**: Schwer zu skalieren bei komplexen Anwendungen.

Data Centered

- **Vorteile**: Zentrale Datenverwaltung.
- **Nachteile**: Single Point of Failure und Performance-Engpässe.

Drei große Patterns

Transaction Script

Organisiert Geschäftslogik als eine Folge von Prozeduren oder Skripten, die Transaktionen durchführen.

- **Einsatzgebiet**: Einfache Anwendungen mit klaren, sequenziellen Prozessen.

Domain Model

Repräsentiert Geschäftslogik und Regeln durch ein reiches Objektmodell.

- **Einsatzgebiet**: Komplexe Anwendungen mit umfangreicher Geschäftslogik.

Table Module

Geschäftslogik wird in Modulen organisiert, die Tabellen der Datenbank repräsentieren.

- **Einsatzgebiet**: Anwendungen mit stark datenzentrierten Operationen und wenigen Geschäftsregeln.

C4-Modell

Das C4-Modell umfasst vier Ebenen:

1. **Context**: Überblick über das System und seine Interaktionen mit externen Entitäten.
2. **Container**: Aufteilung des Systems in Container (Anwendungen/Services).
3. **Component**: Detaillierte Darstellung der Komponenten innerhalb eines Containers.
4. **Code**: Detaillierte Darstellung des Codes innerhalb einer Komponente.

Dieses Modell hilft, die verschiedenen Abstraktionsebenen eines Systems zu verstehen und zu dokumentieren.

Architecture Canvas

Das **Architecture Canvas** ist ein Tool zur visuellen Darstellung und Planung der Softwarearchitektur. Es hilft, verschiedene Aspekte der Architektur auf einem einzigen Blatt darzustellen und dient als Kommunikationsmittel zwischen den Beteiligten. Vorteile sind:

- Übersichtlichkeit und klare Strukturierung.
- Unterstützung bei der Identifikation von Abhängigkeiten und Risiken.

Nachteile können sein:

- Möglicher Informationsverlust durch zu hohe Abstraktion.
- Erfordert regelmäßige Aktualisierung, um relevant zu bleiben.

Cynefin Framework / Codefin

Cynefin Framework

Definition

Das **Cynefin Framework** ist ein Modell zur Entscheidungsfindung und Problemlösung, das hilft, zu verstehen, welches Verhalten in bestimmten Kontexten zum Erfolg führt. Es basiert auf der Komplexitätstheorie und unterteilt Probleme in verschiedene Domänen, um passende Handlungsstrategien zu identifizieren.

Nutzen im Kontext der Softwareentwicklung

Das Cynefin Framework unterstützt Softwareentwickler und -teams dabei, die Natur der Probleme zu erkennen, mit denen sie konfrontiert sind, und entsprechende Lösungsansätze zu wählen. Es hilft, die richtige Methodik für verschiedene Arten von Problemen zu bestimmen und fördert ein besseres Verständnis für die Dynamik und Unsicherheiten in der Softwareentwicklung.

Begriffe und Erklärungen

Cynefin Ein Modell zur Entscheidungsfindung und Problemlösung, das Probleme in verschiedene Domänen einteilt.

Framework Ein strukturiertes Modell oder System, das zur Unterstützung und Strukturierung von Prozessen dient.

Exaptation Die Nutzung bestehender Strukturen oder Prozesse für neue, ursprünglich nicht vorgesehene Zwecke.

4 + 1 Domains

- **Simple** (Klar): Klare Ursache-Wirkungs-Beziehungen, beste Praktiken sind anwendbar.

- **Complicated** (Kompliziert): Ursache-Wirkungs-Beziehungen sind erkennbar, aber Expertise ist erforderlich, um Lösungen zu finden.
- **Complex** (Komplex): Ursache-Wirkungs-Beziehungen sind erst im Nachhinein erkennbar, experimentelle Ansätze und Adaptation sind notwendig.
- **Chaotic** (Chaotisch): Keine erkennbare Ursache-Wirkungs-Beziehung, schnelles Handeln zur Stabilisierung ist erforderlich.
- **Disorder** (Unordnung): Zustand der Ungewissheit, in dem nicht klar ist, welche der vier anderen Domänen anwendbar ist.

Causality Die Beziehung zwischen Ursache und Wirkung in einem bestimmten Kontext.

Correlation Die Beziehung und Wechselwirkung zwischen verschiedenen Variablen, ohne dass eine direkte Ursache-Wirkung-Beziehung impliziert wird.

Constraint Einschränkungen oder Bedingungen, die das Verhalten innerhalb eines Systems beeinflussen.

Codefin

Praktische Anwendung in der Softwareentwicklung

Das **Codefin** Framework basiert auf dem Cynefin Modell und adaptiert dessen Prinzipien für die Softwareentwicklung. Es bietet praktische Werkzeuge und Methoden zur Bewältigung der Komplexität in der Softwareentwicklung. Durch die Einteilung in verschiedene Domänen können Entwickler und Teams geeignete Techniken und Methoden auswählen, um spezifische Herausforderungen zu meistern. Codefin fördert iterative und adaptive Ansätze, um auf Veränderungen und Unsicherheiten in der Softwareentwicklung zu reagieren.

Kanban / Lean Software Development

Zusammenhang von Agile, Lean Software Development, Scrum, XP und Kanban

Herkunft

- **Agile**: Entstand 2001 durch das Agile Manifesto, das eine Reaktion auf schwerfällige Entwicklungsprozesse war.
- **Lean Software Development**: Basierend auf den Prinzipien der Lean Manufacturing von Toyota, adaptiert für die Softwareentwicklung.
- **Scrum**: Entwickelt in den 1990er Jahren, ein Rahmenwerk für die iterative und inkrementelle Softwareentwicklung.
- **Extreme Programming (XP)**: Von Kent Beck in den späten 1990er Jahren entwickelt, fokussiert auf technische Exzellenz und kontinuierliche Verbesserung.
- **Kanban**: Ursprünglich aus der Fertigungsindustrie von Toyota, adaptiert für die Softwareentwicklung zur Visualisierung und Optimierung von Workflows.

Kanban: Principles, Practices und Values

Principles

- Visualisiere den Workflow
- Begrenze die Anzahl der gleichzeitig bearbeiteten Aufgaben (WIP)
- Manage den Fluss
- Mache Prozessregeln explizit
- Implementiere Feedbackschleifen
- Verbessere gemeinsam in Zusammenarbeit und durch wissenschaftliche Methoden

Practices

- Visualisierung von Aufgaben auf einem Kanban-Board
- Limitierung der Work in Progress (WIP)
- Messung und Verwaltung von Durchlaufzeiten
- Kontinuierliche Verbesserung (Kaizen)

Values

- Transparenz
- Fokussierung
- Engagement
- Respekt
- Mut

Funktionsweise eines Kanban Boards und des Cumulative Flow Diagram (CFD)

Kanban Board

Ein **Kanban Board** visualisiert den Workflow, indem Aufgaben als Karten in verschiedenen Spalten dargestellt werden, die unterschiedliche Phasen des Prozesses repräsentieren. Es hilft, Engpässe zu identifizieren und den Fortschritt zu überwachen.

Cumulative Flow Diagram (CFD)

Ein **Cumulative Flow Diagram (CFD)** zeigt die Anzahl der Aufgaben in den verschiedenen Phasen des Workflows über die Zeit. Es hilft, den Fluss der Arbeit zu visualisieren und Engpässe zu erkennen.

Begriffe und Erklärungen

Pull System

Ein **Pull System** bedeutet, dass neue Aufgaben erst begonnen werden, wenn Kapazität im System frei wird, anstatt Arbeiten nach einem festen Plan zu starten.

Limited WIP

Limited WIP (Work In Progress) bezieht sich auf die Begrenzung der Anzahl gleichzeitig bearbeiteter Aufgaben, um Überlastung zu vermeiden und den Fluss zu verbessern.

Visualize the Workflow (Kanban Board)

Das **Kanban Board** visualisiert den Workflow durch Karten und Spalten, um den Fortschritt und Engpässe sichtbar zu machen.

Cumulative Flow Diagramm

Das **Cumulative Flow Diagramm** (CFD) zeigt die Anzahl der Aufgaben in verschiedenen Phasen des Workflows über die Zeit und hilft, den Fluss der Arbeit zu analysieren.

Lead time resp. Cycle time

Lead time ist die Gesamtzeit von der Aufgabenstellung bis zur Fertigstellung. **Cycle time** ist die Zeit, die eine Aufgabe benötigt, um durch den gesamten Workflow zu gelangen.

Kaizen

Kaizen steht für kontinuierliche Verbesserung durch kleine, inkrementelle Änderungen, die kontinuierlich vorgenommen werden, um Prozesse zu optimieren.

Waste in Software Development

Waste bezieht sich auf alle Aktivitäten, die keinen Mehrwert für den Kunden schaffen, wie unnötige Features, Verzögerungen, Fehler und ineffiziente Prozesse.

Principles of Lean

- **Eliminate Waste**: Entferne alle Aktivitäten, die keinen Mehrwert schaffen.
- **Amplify Learning**: Fördere kontinuierliches Lernen und Anpassung.
- **Decide as Late as Possible**: Triff Entscheidungen so spät wie möglich auf Basis fundierter Informationen.
- **Deliver as Fast as Possible**: Liefere so schnell wie möglich, um schnell Feedback zu erhalten.
- **Empower the Team**: Befähige das Team, Entscheidungen zu treffen und Verantwortung zu übernehmen.
- **Build Integrity In**: Integriere Qualität und Integrität von Anfang an.
- **See the Whole**: Optimiere das gesamte System und nicht nur Teile davon.

Unterschiede und Gemeinsamkeiten von Scrum und Kanban

Gemeinsamkeiten

- Beide fokussieren auf kontinuierliche Verbesserung.
- Transparenz und Sichtbarkeit des Arbeitsprozesses sind zentral.
- Beide verwenden ein Board zur Visualisierung der Arbeit.

Unterschiede

- **Scrum** arbeitet mit festen Iterationen (Sprints), während **Kanban** einen kontinuierlichen Fluss hat.
- **Scrum** hat definierte Rollen (Product Owner, Scrum Master, Entwicklungsteam), während **Kanban** flexibler in der Rollenverteilung ist.
- **Scrum** hat strukturierte Meetings (Sprint Planning, Daily Standup, Sprint Review, Sprint Retrospective), während **Kanban** Meetings nach Bedarf durchführt.