# **Entwurf von Softwarearchitekturen**

# **Ziele und Aufgaben des Architekturentwurfs**

• Finden eines Konstruktionswegs, mit dem die funktionalen und nicht-funktionalen Anforderungen aus dem Requirements Engineering umgesetzt werden können

• Unterstützung der Projektziele

O imagine care conține text, captură de ecran, Font

Descriere generată automat

# **Architekturprozesse und deren Interaktionen nach ISO/IEC/IEEE 42020:2019**

O imagine care conține text, captură de ecran, diagramă, Font

Descriere generată automat

O imagine care conține text, Font, captură de ecran, alb

Descriere generată automat

# **Architektur Governance**

• Festlegung von Richtlinien und Standards für die Architekturentwicklung

• Einrichten von Governance-Strukturen wie z.B. Architekturboards

• Überwachung der Einhaltung von Architekturstandardsund richtlinien

This component is responsible for establishing clear rules and standards regarding architecture. This includes defining design principles, technical standards, security policies, and more. These standards are essential to ensure consistency and quality in architecture throughout the project or organization.

Instituting a governance structure, such as an architecture committee, is crucial to oversee and make decisions regarding architecture. This committee is responsible for evaluating and approving architecture proposals, managing risks and conflicts, as well as maintaining compliance with established standards.

# **Architektur Management**

• Planung und Priorisierung von Architekturaktivitäten und – ressourcen

• Risiko- und Änderungsmanagement für die Architektur

• Kommunikation und Koordination mit Stakeholdern

Architecture management involves managing activities and resources related to architecture within the organization or project. This includes planning and prioritizing architectural activities, allocating resources, and tracking progress against established objectives.

# **Architektur Enablement**

• Schulung und Weiterbildung von Entwicklungsteams und Stakeholdern

• Bereitstellung von Tools und Ressourcen zur Unterstützung der Architekturentwicklung

• Aufbau und Pflege eines Wissensmanagements für Architekturwissen

• Förderung einer Kultur der kontinuierlichen Verbesserung und Innovation

This component refers to **providing the necessary resources, tools**, and support to **enable the development and implementation of architecture** within the organization or project.

An important aspect of architecture enablement is the **training and educati**on of **development teams and stakeholders** regarding architectural principles and practices.

Additionally, it's important to develop and manage a knowledge base and knowledge management system to collect, organize, and distribute relevant architecture-related information.

# **Kernprozesse**

# **Konzeptionierung**

• Sammlung und Analyse von Anforderungen und Randbedingungen

• Dokumentation der Architekturkonzepte und – entscheidungen

• Erstellung von Prototypen und Modellen zur Veranschaulichung der Architektur

# **Erarbeitung**

• Detaillierung der Architekturkonzepte

• Unterstützung der Entwicklungsteams bei der Umsetzung der Architektur

• Erstellung von Implementierungsrichtlinien und -vorgaben

# **Evaluierung**

• Durchführung von Architekturbewertungen und –analysen

• Einholung von Feedback von Stakeholdern und Experten

• Identifikation und Bewertung von Risiken und Schwachstellen

# **Zusammenspiel zwischen Architekturentwurf und Requirements Engineering**

O imagine care conține diagramă, Grafică, Font, captură de ecran

Descriere generată automat

|  |  |
| --- | --- |
| Verfeinerung…  • Ergänzung…  • Validierung…  • Priorisierung…  • Dokumentation…  … der Anforderungen | O imagine care conține text, captură de ecran, Font, carte de vizită  Descriere generată automat |

# **Twin Peaks Model**

O imagine care conține diagramă, linie, model

Descriere generată automat

# RE Peak

• Repräsentiert Aktivitäten zur Erhebung und Analyse von Anforderungen

# Architecture Peak

• Repräsentiert Aktivitäten zur Entwicklung und Verfeinerung der Architektur

# Grundgedanke:

RE & Architekturentwurf dürfen nicht isoliert voneinander betrachtet werden.

# Wesentliche Konzepte

• Parallele Entwicklung

• Iterativer Ansatz

• Eng verzahnte Aktivitäten

• Berücksichtigung von Risiken und Unsicherheiten

# **Bottom-Up Architekturentwurf**

O imagine care conține text, captură de ecran, Font, linie

Descriere generată automat

* In contrast, the bottom-up approach starts with the individual components or details and gradually builds up to create a larger, more comprehensive system or solution.
* The process begins with identifying and understanding the specific elements, data points, or components that make up the system. Then, you incrementally combine these smaller components to create larger structures or systems.
* This approach is often used in tasks such as software development, where individual modules or functions are developed independently and later integrated to form the complete system. It can also be applied in problem-solving, where solutions are built iteratively from the ground up based on empirical observations or experimentation.

# **Top-Down Architekturentwurf**

O imagine care conține text, captură de ecran, Font

Descriere generată automat

* In the top-down approach, you start with a broad overview or high-level perspective and then gradually break down the problem or system into smaller, more manageable components or details.
* The process begins with defining the overall goals, objectives, or requirements of the system or project. Then, you decompose these high-level goals into smaller, more specific tasks, sub-systems, or components.
* This approach is often used in project planning, system design, and software development. It allows for a clear understanding of the big picture before diving into the details. It also facilitates communication and alignment of objectives across different levels of an organization or project.

# **Middle-Out / Hybrid-Ansatz**

O imagine care conține text, captură de ecran, Font, diagramă

Descriere generată automat

# **Design by Contract**

**Definition:**

Design by Contract enthält eine Reihe von Prinzipien zur komponentenbasierten Softwareentwicklung

* Die Komponenten schließen untereinander Verträge ab
* Ein Vertrag beinhaltet das Versprechen einer Programmkomponente, eine bestimmte Leistung zu erbringen, wenn bestimmte Voraussetzungen erfüllt sind.
* Setze Programm so aus Komponenten zusammen, dass es richtig funktioniert, wenn alle nur ihre Verträge einhalten.

These contracts specify what each component promises to do and under what conditions.

# **Vertrag für eine Methode**

Ein Vertrag für eine Methode kann zum Beispiel aus folgenden Elementen bestehen:

**Vorbedingung**

Bedingungen, die erfüllt sein müssen, bevor eine Methode aufgerufen werden kann.

Beispiel: Methode zur Berechnung einer Wurzel darf nur aufgerufen werden, wenn der Parameter eine nicht-negative Zahl ist.

**Nachbedingung**

Welche Leistung erbringt die Methode, d.h. welche Eigenschaft gilt nach ihrer Ausführung.

Beispiel: Methode stellt sicher, dass das Quadrat des Ergebnisses gleich der Eingabe ist

**Klasseninvariante**

Zu jedem Zeitpunkt erfüllt der Zustand der Klasse eine bestimmte Eigenschaft (die Invariante).

# **Vor- und Nachteile**

### Vorteile:

1. **Klare Spezifikation**:
   * Durch die explizite Definition von Vorbedingungen, Nachbedingungen und Invarianten wird die Funktionalität von Softwarekomponenten klar und eindeutig beschrieben.
2. **Frühe Fehlererkennung**:
   * DbC ermöglicht die frühzeitige Erkennung von Fehlern, da Vertragsverletzungen während der Entwicklung oder im Teststadium identifiziert werden können.
3. **Verbesserte Wartbarkeit**
4. **Erleichtertes Testen**:
   * Die Spezifikationen der Verträge dienen als Grundlage für Testfälle, was das Testen erleichtert und die Testabdeckung verbessert.

### Nachteile:

1. **Erhöhter Aufwand(efort)**:
   * Das Implementieren und Überprüfen von Verträgen erfordert zusätzlichen Aufwand während der Entwicklungsphase. Es müssen Vorbedingungen, Nachbedingungen und Invarianten definiert und überwacht werden.
2. **Laufzeitüberprüfungen**
3. **Komplexität**:
   * Die Einführung von Design by Contract kann die Komplexität des Codes erhöhen, insbesondere wenn die Verträge sehr detailliert sind oder wenn es viele Komponenten gibt, die miteinander interagieren.

# **Conway‘s Law**

“Organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations.”

Die Organisationsstruktur nimmt Einfluss auf die Softwarearchitektur!

Inverse Conway Maneuve

r • Z.B. Aufteilung der Teams nach Komponenten

O imagine care conține text, hartă

Descriere generată automat

O imagine care conține text, captură de ecran, diagramă, linie

Descriere generată automat

O imagine care conține text, hartă

Descriere generată automat

O imagine care conține text, captură de ecran, diagramă, Font

Descriere generată automat

O imagine care conține text, hartă

Descriere generată automat

O imagine care conține text, captură de ecran, Font, linie

Descriere generată automat

# **Kopplung**

**Definition**

Maß der Abhängigkeit zwischen verschiedenen Modulen oder Komponenten eines Softwaresystems

–wie stark Änderungen an Komponenten andere beeinflussen

**Ziel**

Ziel ist eine geringe Kopplung zwischen Komponenten

Eine geringe Kopplung verbessert die • Wartbarkeit • Flexibilität • Testbarkeit

# **Arten von Kopplung**

O imagine care conține text, diagramă, captură de ecran, Plan

Descriere generată automat

# **O imagine care conține text, captură de ecran, Font Descriere generată automat**

# **Efferente Kopplung/Efferent Coupling (Ce)**

• Anzahl von Komponenten, die eine bestimmte Komponente referenziert

• Ein hoher Ce-Wert deutet darauf hin, dass die Komponente viele andere Komponenten nutzt

• Indiz für starke Kopplung

O imagine care conține text, captură de ecran, proiectare, linie

Descriere generată automat

# **Afferente Kopplung/Afferent Coupling (Ca)**

• Anzahl der anderen Komponenten, die auf eine bestimmte Komponente direkt zugreifen

• Ca zählt, wie viele andere Komponenten direkt von der betrachteten Komponente abhängen

• Ein höherer Ca deutet darauf hin, dass die betrachtete Komponente von vielen anderen Komponenten genutzt wird

• Indiz für gute Wiederverwendung

• Ggf. Single-point-of-Failure

• Ggf. hohe Kopplung

O imagine care conține captură de ecran, text, proiectare, linie

Descriere generată automat

# **Instability (I) / Instabilitätswert**

Verhältnis von efferenter Kopplung zu der Summe aus efferenter und afferenter Kopplung

O imagine care conține Font, alb, linie, număr

Descriere generată automat

O imagine care conține text, Font, chitanță, alb

Descriere generată automat

# **Kohäsion**

**Definition:**

Bezieht sich auf das Maß, in dem die Elemente oder Funktionen innerhalb einer Komponente zusammengehören oder gemeinsam ein klares und zusammenhängendes Ziel verfolgen

**Ziel:**

Ziel ist eine hohe Kohäsion innerhalb einer Komponente.

Eine hohe Kohäsion verbessert die: Wartbarkeit • Flexibilität • Testbarkeit

Hohe Kohäsion

• Die in einer Komponente enthaltenen Elemente oder Funktionen erfüllen gemeinsam eine spezifische Aufgabe

Niedrige Kohäsion

• Die in einer Komponente enthaltenen Elemente oder Funktionen sind weniger zusammengehörig und erfüllen möglicherweise unterschiedliche Aufgaben oder haben unterschiedliche Verantwortlichkeiten

O imagine care conține text, captură de ecran, Font, linie

Descriere generată automat

# **Messung von Kohäsion**

# **LCOM - Lack of Cohesion Metric**

**Definition:**

Die LCOM-Metrik versucht, den Grad des Zusammenhalts innerhalb einer Klasse zu quantifizieren.

O imagine care conține Font, alb, diagramă, linie

Descriere generată automat

O imagine care conține text, Font, captură de ecran, alb

Descriere generată automat

**Beispiel**

**O imagine care conține text, diagramă, captură de ecran, număr

Descriere generată automat**

# **Domain -Driven Design (DDD)**

“The information model is a fundamental tool for managers in the digital change.”

Auch Domänenmodell genannt

**Definition**

Domain Driven Design (DDD) ist eine Herangehensweise an die Modellierung komplexer Software.

* Wenn wir Software entwickeln, sollte unser Fokus nicht primär auf den Technologien liegen, die wir verwenden.
* Stattdessen sollte sich unser Hauptaugenmerk auf die geschäftliche Seite richten, also auf den fachlichen Bereich, den wir mit unserer Software unterstützen wollen – kurz: auf die Domäne.

### Kernkonzepte:

1. **Domain**:
   * Die Domäne repräsentiert den **Anwendungsbereich oder das Geschäftsgebiet**, in dem die Software eingesetzt wird. Sie besteht aus den Problemen, Regeln und Prozessen, die in diesem Bereich vorhanden sind.
2. **Modeling the Domain**:
   * DDD legt den Schwerpunkt auf das Modellieren der Domäne und die Schaffung eines gemeinsamen Verständnisses zwischen Entwicklern und Domänenexperten. **Dies geschieht durch die Erstellung eines Domänenmodells, das die relevanten Konzepte und Beziehungen innerhalb der Domäne abbildet.**
3. **Ubiquitous Language**:
   * Eine zentrale Idee von DDD ist die Verwendung einer einheitlichen, durchgängigen Sprache, die von allen Beteiligten, einschließlich Entwicklern und Domänenexperten, verstanden wird.
   * Diese Sprache ermöglicht eine präzise Kommunikation und trägt dazu bei, Missverständnisse zu vermeiden.
4. **Bounded Context**:
   * Ein abgegrenzter Kontext definiert den Umfang und die Grenzen eines bestimmten Modells innerhalb der Domäne.
   * Jeder abgegrenzte Kontext kann seine eigene einheitliche Sprache und Modelle haben, die innerhalb dieses Kontextes gültig sind.

O imagine care conține text, captură de ecran, Font, document

Descriere generată automat

Beispiel Domänenmodell

O imagine care conține text, diagramă, captură de ecran, Plan

Descriere generată automat

# **Context Mapping:**

# **Identifiziere die Kontextgrenzen und definiere klare Schnittstellen zwischen verschiedenen Kontexten, um die Integration und Kommunikation zu erleichtern.**

# **Weitere Konzepte von DDD**

• Kontextgrenzen (Bounded Context)

* Zentraler Begriff im Domain-Driven Design Domäne wird in klar abgegrenzte Bereiche unterteilt
* Jeder Bounded Context definiert eine Grenze, innerhalb derer eine spezifische Modellierung der Domäne gilt und die Regeln, Begriffe und Konzepte konsistent sind

• Kontextübersicht (Context Map)

* Visualisiert die Beziehungen und Interaktionen zwischen verschiedenen Bounded Contexts innerhalb einer Domäne.
* Hilft, ein besseres Verständnis der Gesamtdomäne zu entwickeln, indem sie zeigt, wie verschiedene Teile eines Systems miteinander verbunden sind und wie sie miteinander kommunizieren.

• Antikorruptionsschicht (Anticorruption Layer)

* Eine Anti-Corruption Layer (ACL) ist ein Architekturkonzept im Domain-Driven Design (DDD)
* Schafft eine schützende Schicht zwischen zwei unterschiedlichen Bounded Contexts oder Systemen
* Diese Schicht stellt sicher, dass der eigene Kontext vor den Einflüssen und den möglicherweise unpassenden Modellen oder Konzepten eines anderen Kontexts geschützt wird.

O imagine care conține text, captură de ecran, diagramă, Font

Descriere generată automat

# **Event Storming**

**Definition**

* Workshop -basierte Methode, um schnell herauszufinden, was in einer Domäne „passiert“.
* Methode, um Fachexperten und Softwareentwickler zusammenzubringen.

**Ablauf**

1. Vorbereitung

**Stakeholder einladen** (z.B. Entwickler, Domänenexperten, Produktmanager) und Moderator bestimmen (Choose a facilitator to guide the session and keep the discussion on track.)

Große Arbeitsfläche/Wand bereitstellen, ebenso viele bunte Haftnotizen und Marker

2. Identifikation von Domänenereignissen

Teilnehmer schreiben wichtige Ereignisse auf **orangene Haftnotizen**.

Ein Ereignis ist etwas, das passiert und für das Geschäft von Bedeutung ist

„Order Placed“, “Payment Processed“

Ereignisse in chronologischer Reihenfolge an die Wand kleben

3. Erweitern und Vertiefen **Kommandos definieren** (**blaue Haftnotizen**), die Ereignisse auflösen

Ein Kommando ist eine Benutzeraktion oder eine Entscheidung im System

„Place Order“, „Process Payment“

# **Weitere Designprinzipien**

# **Postel‘s Law (Robustheitsprinzip)**

"Be conservative in what you do, be liberal in what you accept."

Sei streng bei dem, was du tust, und offen bei dem, was du von anderen akzeptierst.

O imagine care conține text, Font, captură de ecran, alb

Descriere generată automat

**Rechne mit Service-Ausfällen (Expect Service Outages)**:

* As an API consumer, you should anticipate that the service you are interacting with may experience downtime or interruptions. This means your application should be designed to handle such situations gracefully

**Mit fehlerhaftem Verhalten rechnen (Expect Faulty Behavior)**:

* APIs might sometimes return unexpected results or errors. Your application should be resilient to these issues by validating responses, handling different types of errors appropriately, and not assuming that the API will always behave perfectly.

**Biete nicht nur die aktuelle API an, sondern auch mind. eine vergangene Version an (Offer Not Only the Current API Version but Also at Least One Previous Version)**:

* Maintain backward compatibility by supporting older versions of the API alongside the current version. This practice, often referred to as “Expand and Contract,” allows users time to transition from older to newer versions without immediate disruption. It provides a smoother upgrade path and reduces the risk of breaking existing integrations when new versions are released.

**"Be conservative in what you send"**:

* When your software generates and sends data (e.g., messages, requests, responses), it should strictly adhere to the established protocols and standards.

**"Be liberal in what you accept"**:

* When your software receives data, it should be prepared to handle a wide range of input, including non-standard or slightly incorrect data. This means being tolerant of variations and minor deviations from the protocol, as long as the data can still be interpreted and processed correctly.

# **Interface Segregation Principle**

* Clients sollten nicht dazu gezwungen werden, von Schnittstellen (und deren Methoden) abzuhängen, die sie gar nicht brauchen.
* Damit würden sich Änderungen an nicht benötigten Schnittstellen (und deren Methoden) unnötigerweise auf Verwender auswirken.
* Schnittstellen sollten fachlich strukturiert sein.
* Kleine und fokussierte Schnittstellen sind leichter implementierbar und wartbar

O imagine care conține text, diagramă, captură de ecran, Font

Descriere generată automat

# **Offen-Geschlossen-Prinzip**

* Ein Modul soll für Erweiterungen offen sein.
* Ein Modul soll für Änderungen geschlossen sein.

# **So-einfach-wie-möglich-Prinzip**

„Mache die Dinge so einfach wie möglich – aber nicht einfacher“

# **Information Hiding (Geheimnisprinzip)**

* Kapselung von Komplexität in Komponenten
* Verwenden von Schnittstellen

# **Regelmäßiges Refactoring und Redesign**

# **Nutzung von Architekturmustern**