# **Grundlegende Konzepte**

Das magische Viereck erfolgreicher Softwareprojekte

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

Descriere generată automat

Das "Magische Viereck" erfolgreicher Softwareprojekte beschreibt vier grundlegende Dimensionen, die im Gleichgewicht sein müssen, um ein Softwareprojekt erfolgreich zu gestalten. Diese Dimensionen sind Qualität, Zeit, Kosten und Umfang. Jede dieser Dimensionen beeinflusst die anderen, und Änderungen in einer Dimension haben Auswirkungen auf die anderen. Hier ist eine detaillierte Erklärung:

### 1. Qualität

**Beschreibung**: Hohe Qualität bedeutet, dass die Software den Anforderungen und Erwartungen der Benutzer entspricht und keine Fehler oder Sicherheitslücken aufweist. **Beispiele für Maßnahmen zur Qualitätssicherung**:

* **Code-Reviews und Testing**: Regelmäßige Überprüfung des Codes und umfassende Tests (Unit-Tests, Integrationstests, automatisierte Tests).

### 2. Zeit (Time)

**Beschreibung**: Zeit bezieht sich auf die Dauer, die benötigt wird, um das Softwareprojekt zu vollenden. Dies umfasst alle Phasen des Projekts, von der Planung über die Entwicklung bis hin zur Auslieferung. **Beispiele für Maßnahmen zur Zeitsteuerung**:

### 3. Kosten

**Beschreibung**: Kosten beziehen sich auf die finanziellen Aufwendungen, die mit der Entwicklung und Implementierung der Software verbunden sind. Dazu gehören Personalkosten, Infrastrukturkosten, Lizenzkosten und andere Ausgaben. **Beispiele für Maßnahmen zur Kosteneffizienz**:

### 4. Umfang (Scope)

**Beschreibung**: Der Umfang bezieht sich auf die Gesamtheit der Funktionen und Anforderungen, die das Softwareprojekt erfüllen soll. Ein gut definierter Umfang stellt sicher, dass alle notwendigen Funktionen implementiert werden, ohne dass es zu unnötigen Erweiterungen oder Änderungen kommt

### Zusammenfassung

Das Magische Viereck verdeutlicht, dass die erfolgreiche Durchführung eines Softwareprojekts ein ausgewogenes Management von Qualität, Zeit, Kosten und Umfang erfordert. Änderungen in einer Dimension wirken sich zwangsläufig auf die anderen aus. Zum Beispiel kann eine Erhöhung der Qualitätsanforderungen zu höheren Kosten und längeren Entwicklungszeiten führen, oder eine Verkürzung der Zeit kann die Qualität beeinträchtigen oder den Umfang reduzieren. Ein effektives Projektmanagement sorgt dafür, dass diese Dimensionen im Gleichgewicht bleiben, um die Projektziele zu erreichen

# **Warum sind Kenntnisse über Softwarearchitektur wichtig?**

* **Komplexität von Softwareprojekten**: Moderne Softwareprojekte sind oft sehr komplex und setzen sich aus vielen Teilen zusammen, die reibungslos interagieren müssen. Eine durchdachte Softwarearchitektur hilft dabei, diese Komplexität zu bewältigen, indem sie klare Strukturen schafft. Dadurch wird nicht nur die Entwicklung erleichtert, sondern auch die Wartung und Weiterentwicklung.
* **Skalierbarkeit und Leistung**: Skalierbarkeit bedeutet, dass ein System mehr Last bewältigen kann, ohne an Leistung einzubüßen. Eine solide Softwarearchitektur ermöglicht es, Anwendungen effizient zu skalieren, wenn die Nutzerzahlen steigen. Dabei wird darauf geachtet, dass zusätzliche Ressourcen wie Server oder Datenbanken problemlos hinzugefügt werden können.
* **Sicherheit und Datenschutz**: In der heutigen digitalen Welt sind Sicherheit und Datenschutz von größter Bedeutung. Eine gute Softwarearchitektur integriert Sicherheitsmaßnahmen von Anfang an, um Daten zu schützen und die Privatsphäre zu wahren. Dazu gehören sichere Codierungspraktiken, Verschlüsselung und Mechanismen zur Authentifizierung und Autorisierung.
* **Technologische Vielfalt**: Die Technologielandschaft ist sehr vielfältig und ändert sich ständig. Kenntnisse über Softwarearchitektur helfen dabei, die richtigen Technologien auszuwählen und sie effektiv zu integrieren. Das bedeutet auch, zu verstehen, wie verschiedene Frameworks, Sprachen und Tools zusammenarbeiten können, um ein kohärentes System zu schaffen.
* **Agilität und kontinuierliche Bereitstellung**: In einer schnelllebigen Umgebung sind Agilität und die Fähigkeit zur kontinuierlichen Bereitstellung von großer Bedeutung. Eine flexible Softwarearchitektur unterstützt agile Methoden, die es ermöglichen, schnell neue Funktionen und Updates bereitzustellen und dabei dennoch qualitativ hochwertige Software zu entwickeln.

### 1. **Complexity of Software Projects**

Modern software projects are often highly complex, consisting of numerous components that need to interact seamlessly. A well-designed software architecture helps manage this complexity by providing clear structures and hierarchies. This not only makes development easier but also facilitates maintenance and further development.

### 2. **Scalability and Performance**

Scalability refers to a system's ability to handle increased load without compromising performance. A robust software architecture ensures that applications can scale efficiently as user demand grows. This involves designing the system in a way that can accommodate additional resources, such as servers or databases, without major overhauls.

### 3. **Security and Privacy**

Security and privacy are paramount in today’s digital landscape. A strong software architecture incorporates security measures from the ground up, ensuring that data is protected and privacy is maintained. This includes implementing secure coding practices, encryption, authentication, and authorization mechanisms to safeguard against breaches and vulnerabilities.

### 4. **Technological Diversity**

The technological ecosystem is vast and constantly evolving. Knowledge of software architecture helps developers make informed decisions about which technologies to use and how to integrate them effectively. This includes understanding how different frameworks, languages, and tools can work together to create a cohesive and functional system.

### 5. **Agility and Continuous Delivery**

In today’s fast-paced environment, agility and the ability to continuously deliver updates are critical. A flexible software architecture supports agile methodologies, allowing for iterative development and frequent releases. This ensures that new features and updates can be delivered quickly and efficiently, meeting user needs and staying competitive in the market.

# **Big Ball of Mud**

Begriff, um eine unstrukturierte und chaotische Softwarearchitektur zu beschreiben

Typische Merkmale sind

* Mangel an Struktur
* Schlechte Modularisierung
* Spaghetti-Code
* Technologische Vielfalt
* Wachsende Komplexität

# **Technische Schulden**

• Entstehen, wenn Teams bewusst Entscheidungen treffen, die kurzfristig Vorteile bringen, aber langfristig Auswirkungen auf die Qualität haben.

• Ähnlich wie finanzielle Schulden müssen auch technische Schulden irgendwann „zurückgezahlt“ werden durch Verbesserung des Source Codes.

Technical debt is a concept in software development that describes the consequences of taking shortcuts or making expedient decisions in the development process. Here's an explanation:

### What is Technical Debt?

* **Origin**: Technical debt arises when development teams knowingly make decisions that prioritize short-term benefits, such as faster delivery or cost savings, but may lead to long-term issues with the quality or maintainability of the software.
* **Similarity to Financial Debt**: Just as financial debt involves borrowing money that must be repaid with interest, technical debt involves borrowing time or sacrificing code quality, which eventually needs to be "repaid" through improvements to the source code.

# **Ursachen und Arten technischer Schulden**

• Schnelle Lösungen

• Veraltete Technologien

• Fehlende Tests

• Schlechte Codequalität

• Nicht behandelte Probleme

Rückzahlung technischer Schulden erfordert Zeit, Ressourcen und Engagement des Entwicklungsteams.

"Wartbarkeit" is a German term that translates to "maintainability" in English. In software development, "wartbarkeit" refers to the ease with which a software system or application can be maintained or modified over time. A highly maintainable system is one that is easy to understand, update, and extend without introducing errors or causing disruptions to existing functionality. Factors that contribute to maintainability include clean and modular code, comprehensive documentation, adherence to coding standards, and the presence of automated tests.

# **Entwicklung und Effekt von technischen Schulden – Qualitätsmerkmal Wartbarkeit**

O imagine care conține text, diagramă, linie, Interval

Descriere generată automat

# **Arten von IT-Architekten & Aufgaben**

**Enterprise Architekt**

• Starke strategische Ausrichtung

• IT-Business-Alignment

• Technologie-Roadmaps

**Solution / System Architekt**

• Leadership

• Eingebunden in Anforderungsspezifikation

• Systemintegration

**Software Architekt**

• Erstellen und Managen von Softwarearchitekturen

• Enge Zusammenarbeit mit Entwicklungsteam

• Kooperation, Teamwork, Netzwerken

# **Typische Aufgaben von Softwarearchitekten**

• Architektonische Planung

• Technologieauswahl

• Code-Reviews und Best Practices

• Skalierung und Performance

• Risikomanagement

• Kommunikation und Zusammenarbeit

• Wartbarkeit und Erweiterbarkeit

• Qualitätssicherung

# **Wann ist eine Softwarearchitektur erfolgreich?**

Der Erfolg einer Softwarearchitektur bemisst sich am Erfolg der Software!

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

Descriere generată automat

# **Softwarearchitektur nach ISO/IEC/IEEE 42020:2019**

**Definition:**

„Grundlegende Konzepte oder Eigenschaften einer Entität in ihrer Umgebung und leitende Prinzipien für die Realisierung und Entwicklung dieser Entität und der damit verbundenen Lebenszyklusprozesse.“

# **Fundamental concepts or properties**

Sie sollen in der Regel durch die Komponenten der Entität, die Beziehungen zwischen den Komponenten und durch die Beziehungen zwischen der Entität und ihrer Umgebung verkörpert werden.

# **Architecture entity (Architektureinheit)**

thing being considered, described, discussed, studied or otherwise addressed during the architecting effort

# **Zentrale Aspekte der Definition von Softwarearchitektur**

Fundamentale Konzepte und Eigenschaften eines Systems

* Die Komponenten des Systems
* Die Beziehungen/Schnittstellen zwischen den Komponenten
* Die Beziehungen zwischen dem System und seiner Umgebung

# **Schnittstelle**

**Definition**

Eine Schnittstelle repräsentiert einen wohldefinierten Zugangspunkt zum System oder zu dessen Komponenten.

* Dabei beschreibt eine Schnittstelle die Eigenschaften dieses Zugangspunkts, wie z.B. Attribute, Daten und Funktionen.

# **Angebotene und benötigte Schnittstellen**

Angebotene Schnittstelle

* Wird von einer Komponente bereitgestellt
* Definiert, welche Dienste oder Funktionen verfügbar sind
* Andere Komponenten können diese Schnittstelle nutzen, um Dienste in Anspruch zu nehmen

Angeforderte Schnittstelle

* Wird von einer Komponente benötigt
* Definiert Abhängigkeiten und Anforderungen an andere Komponenten
* Komponente kann nur dann funktionieren, wenn alle angeforderten Schnittstellen von anderen Komponenten bereitgestellt werden

# **Verantwortung für Schnittstellen**

Die Verantwortung für Schnittstellen kann unterschiedlich verteilt sein

# **Schnittstellenanbieter**

* Schnittstellenanbieter kann verantwortlich für die Definition, Entwicklung, Wartung und Dokumentation der Schnittstelle sein (ist meistens der Fall)
* Beispiel: Google Maps API Google ist Anbieter und übernimmt auch die Verantwortung für die Definition, Entwicklung, Wartung und Dokumentation dieser Schnittstelle

# **Schnittstellenverwender**

* Es kann vorkommen, dass ein Verwender der Schnittstelle gleichzeitig die Gesamtverantwortung oder Teile der Verantwortlichkeiten für angebotene Schnittstellen übernimmt.
  + Beispiel: Entwicklung einer maßgeschneiderten API durch ein Softwareentwicklungsunternehmen für ein E-Commerce-Unternehmen
  + E-Commerce-Unternehmen
    - Trägt inhaltliche Verantwortung für die Schnittstellen
    - Definiert Anforderungen an die Schnittstellen
  + Softwareentwicklungsunternehmen
    - Implementiert die Schnittstellen, ist ggf. zusätzlich verantwortlich für Dokumentation, Wartung und weiterer Aspekte

# **Normungsorganisationen**

* Für allgemein akzeptierte Schnittstellen
* Meist standardisiert und definiert durch Organisationen wie zum Beispiel ISO, IEEE, etc. Beispiele für Schnittstellen: USB, HDMI, TCP/IP
* Normungsorganisationen definieren Schnittstellen; Implementierung und Wartung werden meist durch Drittanbieter vorgenommen

# **Komponenten vs Module**

Beide Konzepte, Komponenten und Module, dienen dazu, den Code in überschaubare, wiederverwendbare Einheiten zu organisieren, um die Wartbarkeit, Lesbarkeit und Skalierbarkeit von Software zu verbessern.

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

Descriere generată automat

# **Kerneigenschaften einer Komponente**

* Export und Import von Schnittstellen
* Kapselung und Austauschbarkeit
* Konfigurierbarkeit

# **Export und Import von Schnittstellen**

• Eine Komponente bietet Schnittstellen an, die sie im Sinne eines Vertrages garantiert.

• Diese Garantie gilt unter der Bedingung, dass die von ihr benötigten Schnittstellen im Rahmen einer entsprechenden Konfiguration bereitgestellt werden.

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

Descriere generată automat

# **Austauschbarkeit und Kapselung**

• Über die Schnittstellen kapselt eine Komponente die Implementierung

• Solange alle Schnittstellenverträge erfüllt werden, können Komponenten ersetzt werden.

# **Konfigurierbarkeit**

**Arten von Konfigurierbarkeit**

**• Konfiguration von Eigenschaften und Verhalten**

Allgemeine Einstellungen, Aussehen, Verhalten, …

**• Komposition von Komponenten**

Verbindungen zwischen Komponenten zur Erstellung eines vollständigen Anwendungssystems

**• Dynamische vs. statische Konfiguration**

* Dynamic configuration allows adjustments during the runtime of the system, while static configuration requires changes to be made before the system starts.

Voraussetzung von Konfigurierbarkeit: Effektives Konfigurationsmanagement

* Konfigurationsdateien
* Versionierung
* Fehlermanagement

**Effective Configuration Management**:

* Effective configuration management involves **managing configuration files, versioning configurations**, error management, and other aspects to **ensure that the system is properly configured** and changes are managed effectively.

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

Descriere generată automat

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

Descriere generată automatO imagine care conține text, diagramă, Font, Plan

Descriere generată automat

O imagine care conține text, captură de ecran, Font

Descriere generată automat

O imagine care conține text, Font, captură de ecran

Descriere generată automat

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat