# **Qualität von Softwarearchitekturen**

Definition: Wird gemessen indem man folgendes betrachtet:

die Architektur eines Softwaresystems

* erfüllt die Anforderungen und Erwartungen der Stakeholder
* und gleichzeitig die langfristige Wartbarkeit, Erweiterbarkeit, Performance und andere relevante Qualitätsattribute gewährleistet(sicherstellt).

A well-designed architecture is the foundation for high-quality software

(Gut gestaltete Architektur ist die Grundlage für eine qualitativ hochwertige Software)

# **Unterschiedliche Stakeholder haben unterschiedliche Prioritäten hinsichtlich Qualität**

Anwender: Zufriedenheit, Benutzerfreundlichkeit

Entwickler: Gut geschriebener und leicht verständlicher Code

Manager: Übergeordnete Sichtweise, Gesamteindruck der Qualität, Wartungskosten

# **Qualitätsmerkmale für Software nach ISO/IEC 25010**

9 Qualitätsmerkmale, die jeweils in Untermerkmale unterteilt sind

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

Descriere generată automat

# **Funktionelle Eignung**

Beschreibt, wie gut die Software die vorgesehenen Aufgaben erfüllt.

* **Vollständigkeit**: Alle notwendigen Funktionen sind vorhanden.
* **Richtigkeit**: Funktionen liefern korrekte Ergebnisse.
* **Angemessenheit**: Funktionen sind nützlich und zweckdienlich für die Aufgabe

# **Leistungseffizienz**

Beschreibt die Effizienz der Software in Bezug auf die Nutzung von Ressourcen.

* **Zeitverhalten**: Antwortzeiten und Verarbeitungsgeschwindigkeit der Software.
* **Ressourcennutzung**: Optimierung der Nutzung von Ressourcen wie CPU, Speicher und Bandbreite.
* **Kapazität**: Maximale Anzahl von Nutzern und Transaktionen, die die Software effizient verarbeiten kann.

In the context of software quality, Leistung refers to how well the software performs its intended functions in terms of efficiency, speed, and resource usage. Here are the key aspects of Leistung (performance) as defined in the ISO/IEC 25010 standard for software quality:

### 1. Zeitverhalten (Time Behavior)

* **Antwortzeit (Response Time)**: The time taken by the system to respond to a user's input or a request.
* **Verarbeitungsgeschwindigkeit (Processing Speed)**: The speed at which the software processes data and completes tasks.
* **Durchsatz (Throughput)**: The amount of work performed by the software in a given period of time.

### 2. Ressourcennutzung (Resource Utilization)

* **Effizienz der Ressourcennutzung (Resource Efficiency)**: How effectively the software uses available resources such as CPU, memory, disk space, and network bandwidth.
* **Ressourcenverbrauch (Resource Consumption)**: The amount of resources consumed by the software during its operation.

### 3. Kapazität (Capacity)

* **Maximale Nutzeranzahl (Maximum Number of Users)**: The highest number of concurrent users the software can support without performance degradation.
* **Maximale Transaktionsanzahl (Maximum Number of Transactions)**: The highest number of transactions the software can handle within a specified period.

# **Kompatibilität**

Bewertet, wie gut die Software mit anderen Produkten und Systemen zusammenarbeitet.

* **Koexistenz**: Fähigkeit der Software, ohne gegenseitige Beeinflussung neben anderen unabhängigen Produkten zu funktionieren.
* **Interoperabilität**: Fähigkeit der Software, Informationen und Dienste mit anderen Systemen auszutauschen und zu nutzen.

# **Interaktionsfähigkeit**

* Erlernbarkeit
* Benutzerengagement
* Inklusivität Benutzerassistenz
* Selbstbeschreibungsfähigkeit
* Zuverlässigkeit
* Schutz des Nutzers vor Fehleingaben

# **Zuverlässigkeit**

"Zuverlässigkeit" is a German term that translates to "reliability" in English. In the context of software quality, Zuverlässigkeit refers to the ability of the software to perform its required functions under stated conditions for a specified period of time.

Bewertet die Fähigkeit der Software, unter bestimmten Bedingungen konsistent zu funktionieren.

* **Reife**: Häufigkeit von Softwarefehlern und deren Auswirkungen.
* **Verfügbarkeit**: Fähigkeit der Software, bei Bedarf betriebsbereit zu sein.
* **Fehlertoleranz**: Fähigkeit der Software, bei Fehlern oder Ausfällen weiterhin korrekt zu funktionieren.
* **Wiederherstellbarkeit**: Fähigkeit der Software, nach einem Fehler wieder in den Normalbetrieb zurückzukehren.
*  **Reife (Maturity)**: The frequency and severity of software failures. Mature software has fewer and less severe bugs.
*  **Verfügbarkeit (Availability)**: The degree to which the software is operational and accessible when required for use. High availability means the system is up and running whenever needed.
*  **Fehlertoleranz (Fault Tolerance)**: The ability of the software to continue operating correctly even in the event of a failure. Fault-tolerant systems can handle errors without significant impact on functionality.
*  **Wiederherstellbarkeit (Recoverability)**: The capability of the software to recover data and resume functioning after a failure or unexpected event. This includes backup and restore procedures and the speed of recovery.

# **Sicherheit (Security)**

* Vertraulichkeit: Schutz von Daten vor unbefugtem Zugriff.
* Integrität: Sicherstellung der Korrektheit und Vollständigkeit von Daten.
* Verfügbarkeit: Sicherstellung, dass autorisierte Benutzer auf die Daten und Dienste zugreifen können.
* Nachweisbarkeit: Fähigkeit, Aktionen und Ereignisse nachzuverfolgen und zu überprüfen.
* Authentizität: Überprüfung der Identität von Benutzern und Systemen.

# **Wartbarkeit**

* Modularität
* Wiederverwendbarkeit
* Analysierbarkeit
* Modifizierbarkeit
* Testbarkeit

# **Flexibilität**

* Anpassbarkeit

Fähigkeit, die Software an neue oder sich ändernde Anforderungen anzupassen.

* Skalierbarkeit
* Installierbarkeit
* Austauschbarkeit

# **Sicherheit (Safety)**

 **Betriebssicherheit**: Fähigkeit der Software, in einem sicheren Zustand zu bleiben oder schnell zu einem sicheren Zustand zurückzukehren.

 **Risikominderung**: Maßnahmen zur Minimierung von Risiken und zur Vermeidung von Unfällen oder Schäden.

 **Fehlersicherheit**: Fähigkeit der Software, Fehler zu erkennen und darauf zu reagieren, um Gefahren zu vermeiden.

# **Qualitätsziel und Qualitätsanforderung**

# **Definition: Qualitätsziel**

Eine abstrakte Anforderung an ein Qualitätsmerkmal eines Softwareprodukts.

Beispiel: „Die Software soll möglichst wenig Ressourcen verbrauchen.“ (➔ Qualitätsmerkmal Leistungseffizienz)

# **Definition: Qualitätsanforderung**

Eine konkrete und messbare Anforderung an ein bestimmtes Produktmerkmal, die sich auf ein Qualitätsmerkmal eines Softwareprodukts auswirkt.

Beispiel: „Jeder Task soll immer in höchstens 2s abgeschlossen sein.“ (➔ Qualitätsmerkmal Leistungseffizienz, Untermerkmal Zeitverhalten)

# **Das GQM Modell**

**Definition:**

Framework zur Messung und Verbesserung der Softwarequalität durch strukturierte:

* Zielsetzung (Goals)
* Frageformulierung (Questions)
* Metrikdefinitionen (Metrics)

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

Descriere generată automat

1. **Ziele (Goals)** - Festlegung klarer Qualitätsziele für das Softwareprojekt

Beispiele: Verbesserung der Benutzerfreundlichkeit, Erhöhung der Systemzuverlässigkeit, Reduzierung der Fehlerquote

2. **Fragen (Questions)**- Basierend auf den Zielen: Formulierung von Fragen, die helfen sollen, den Fortschritt bei der Erreichung der Ziele zu bewerten.

Beispiele: Wie zufrieden sind die Benutzer mit der Benutzeroberfläche? Wie oft treten Fehler im System auf? Wie gut erfüllt die Software die funktionalen Anforderungen?

3**. Metriken (Metrics)**

Basierend auf den Fragen: **Auswahl geeigneter Metriken**, um den **Fortschritt zu messen** und die **Antworten auf die Fragen zu quantifizieren**.

Beispiele: Anzahl der Benutzerinteraktionen pro Stunde, Anzahl der behobenen Fehler pro Woche, durchschnittliche Antwortzeit des Systems

Messungen und Analysen in Softwareprojekten sollen so zielgerichtet und nachvollziehbar gestaltet werden.

# **Prinzipien der Qualitätssicherung und Bewertung**

# **Analytische Qualitätssicherung**

Ziel: Identifikation potenzieller Fehler oder Schwachstellen sowie Bewertung der Qualität

• Maßnahmen

* Code Reviews
* Architekturbewertungen (z.B. ATAM oder SAAM)
* Statische Code-Analyse
* Formale Verifikation
* Modellbasierte Analyse
* Prototyping und Simulation
* Metriken und Kennzahlen
* Security Audits und Penetration Testing

Analytical quality assurance is an essential component of the software development process, aiming to identify potential errors or weaknesses in a software product and evaluate its overall quality. Here's a detailed explanation of the mentioned measures:

1. **Code Reviews:** Code reviews involve systematic inspections of source code by developers or a team of developers. They help identify errors, violations of coding standards, and inefficient code, providing an opportunity to improve code quality, maintainability, and readability.
2. **Architecture Reviews (e.g., ATAM or SAAM):** Architecture evaluation methods like ATAM and SAAM enable structured analysis of a system's architecture. They help identify potential problems early, assess risks, and understand the impacts of architectural decisions.
3. **Static Code Analysis:** Static code analysis tools automatically scan source code to identify errors, security vulnerabilities, coding standards violations, and other potential issues. They offer an efficient way to find and fix code weaknesses.
4. **Formal Verification:** Formal verification is a mathematical process where the correctness of a system is verified based on formal specifications and logical proofs. It allows for rigorous verification of system properties and ensures confidence in the system's correctness.
5. **Model-Based Analysis:** Model-based analysis techniques use system models to analyze its behavior, performance, or other properties. They help identify potential problems before the system is implemented and facilitate the evaluation of changes or improvements.
6. **Prototyping and Simulation:** Prototyping and simulation allow various aspects of the system to be tested in a controlled environment. They help identify potential issues, evaluate performance, and enhance user experience before the system goes into production.
7. **Metrics and Key Performance Indicators (KPIs):** Metrics and KPIs quantify and monitor various aspects of software quality and performance. They enable objective assessment, trend identification, and continuous improvement.
8. **Security Audits and Penetration Testing:** Security audits and penetration tests are conducted to uncover security vulnerabilities and weaknesses in the system. They help identify potential attack vectors and improve the security of the system.

# **Konstruktive Qualitätssicherung**

Konstruktive Qualitätssicherung

• Ziel: Gewährleisten der geforderten Qualität bereits während der Entwicklung Konzentration auf präventive Ansätze

• Maßnahmen

* Modularisierung und Kapselung
* Architekturmuster und Design Patterns
* Coding Standards und Best Practices
* Automatisierte Tests
* Continuous Integration (CI) und Continuous Deployment (CD)
* Regelmäßige Refactorings
* Dokumentation
* Regelmäßige Architekturbewertungen und –prüfungen
* Prototyping
* Design-by-Contract

Constructive quality assurance focuses on ensuring the required quality of a software product during its development phase. It emphasizes preventive approaches to quality management, aiming to address potential issues before they manifest in the final product.

# **Quantitative Architekturbewertung**

Merkmale der quantitativen Architekturbewertung

* Messbarkeit - dass die Bewertungskriterien oder -metriken objektiv quantifizierbar sind.
* Objektivität - Die Bewertungskriterien sollten klar definiert und unabhängig von individuellen Meinungen sein.
* Vorhersagbarkeit - sollte zuverlässige Ergebnisse liefern, die es ermöglichen, potenzielle Probleme frühzeitig zu erkennen
* Automatisierbarkeit - die Bewertungsmethoden automatisiert werden können

Beispiele:

* Zyklomatische Komplexität (zwar eher Fokus auf Komplexität von Source Code, aber auch Indiz für Qualität von Softwarearchitektur)
* Messung der Antwortzeiten im System
* Maximale Anzahl an gleichzeitigen Nutzern, die das System bewältigen kann

# **Zyklomatische Komplexität**

**• Ansatz: Strukturelle Komplexität messen**

**• Quelle: Kontrollflussgraph G**

betrachten wir den Kontrollflussgraphen des Programms, der die verschiedenen Wege darstellt, die das Programm nehmen kann.

* Die zyklomatische Komplexität ist eine Metrik **zur Bewertung der strukturellen Komplexität eines Programms**.
* Sie wurde von Thomas J. McCabe entwickelt, um die Anzahl der unabhängigen Pfade in einem Programm zu messen. Diese Metrik wird oft verwendet, um die Testbarkeit und Wartbarkeit von Programmen zu bewerten.
* Der Ansatz zur Berechnung der zyklomatischen Komplexität besteht darin, den Kontrollflussgraphen (auch bekannt als Steuerflussgraph) des Programms zu analysieren.
* Ein Kontrollflussgraph stellt die möglichen Pfade durch ein Programm dar, wobei die Knoten die Anweisungen im Programm und die Kanten die Kontrollflussverzweigungen zwischen den Anweisungen darstellen.
* Die zyklomatische Komplexität gibt die minimale Anzahl von Tests an, die erforderlich sind, um alle Pfade im Programm zu durchlaufen und somit eine vollständige Pfadüberdeckung zu erreichen. Eine höhere zyklomatische Komplexität deutet auf ein komplexeres Programm hin, was oft zu höheren Anforderungen an das Testen und zur Schwierigkeit der Wartung führt.

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

Descriere generată automat

Die zyklomatische Zahl gibt die maximale Anzahl linear unabhängiger Zyklen auf dem Kontrollflussgraphen an.

O imagine care conține text, diagramă

Descriere generată automat

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

Descriere generată automat

# **Qualitative Architekturbewertung**

Die qualitative Architekturbewertung bezieht sich auf die Bewertung einer Softwarearchitektur anhand subjektiver Kriterien und Qualitätsmerkmale. Im Gegensatz zur quantitativen Bewertung, die auf messbaren Metriken basiert, konzentriert sich die qualitative Bewertung auf Aspekte wie Designprinzipien, Architekturmustern und Erfahrungswissen. Hier sind die Merkmale der qualitativen Architekturbewertung im Detail erklärt:

1. **Subjektivität:** Qualitative Architekturbewertungen sind oft subjektiv, da sie auf individuellen Einschätzungen, Meinungen und Erfahrungen basieren. Die Bewertungskriterien können je nach den Vorlieben, Erfahrungen und Perspektiven der beteiligten Personen variieren. Dies kann zu unterschiedlichen Einschätzungen führen, die von verschiedenen Personen vorgenommen werden.
2. **Komplexität:** Die qualitative Architekturbewertung berücksichtigt die Komplexität der Architektur und deren Auswirkungen auf die Wartbarkeit, Skalierbarkeit, Flexibilität und andere Qualitätsattribute des Systems. Es werden Aspekte wie die Struktur der Architektur, die Verständlichkeit des Designs und die Einfachheit der Implementierung bewertet.
3. **Erfahrung und Expertise notwendig:** Für eine qualitative Architekturbewertung sind Erfahrung und Expertise in Softwarearchitektur erforderlich. Die Bewertung basiert oft auf dem Fachwissen und der Erfahrung der Architekten, Entwickler und anderen beteiligten Personen. Erfahrene Architekten können Muster erkennen, bewährte Praktiken anwenden und potenzielle Probleme identifizieren, die für weniger erfahrene Personen möglicherweise nicht offensichtlich sind.
4. **Exploratives Vorgehen:** Bei der qualitativen Architekturbewertung wird oft ein exploratives Vorgehen verwendet, um die verschiedenen Aspekte der Architektur zu erkunden und zu verstehen. Dies kann beinhalten, dass verschiedene Alternativen und Ansätze betrachtet werden, um die Vor- und Nachteile zu bewerten und die bestmögliche Architekturlösung zu identifizieren.

• Merkmale qualitativer Architekturbewertung

* Subjektivität
* Komplexität
* Erfahrung und Expertise notwendig
* Exploratives Vorgehen - , um die verschiedenen Aspekte der Architektur zu erkunden und zu verstehen.

• Beispiel: ATAM – Architecture Tradeoff Analysis Method

# **ATAM – Architecture Tradeoff Analysis Method**

Übersetzt: „Verfahren zur Analyse von Architekturkompromissen“

• Strukturierte, Szenario-basierte Architekturbewertung

• Führende Methode im Bereich der Architekturbewertung

• Der Zweck des ATAM besteht darin, die Folgen von Architekturentscheidungen im Hinblick auf die Anforderungen an Qualitätsmerkmale zu bewerten.

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

Descriere generată automat

# **Annahme:**

* Nicht alle Qualitätsziele lassen sich gleichermaßen erfüllen
* Daher müssen Kompromisse bei Architekturentscheidungen gefunden werden

# **Grundsätzliches Vorgehen**

* Aktuelle Architektur und Geschäftsziele verstehen
* Darauf basierend die Architektur bewerten

# **Kompromisse / Tradeoffs – Beispiele**

• Leistung vs. Sicherheit

• Flexibilität vs. Einfachheit

• Skalierbarkeit vs. Konsistenz

• Wartbarkeit vs. Performance

• Benutzerfreundlichkeit vs. Sicherheit

• Kosten vs. Qualität

=> Die Identifizierung und Bewältigung von Tradeoffs ist ein wesentlicher Bestandteil von Softwarearchitektur

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

Descriere generată automat

# **ATAM wird in folgenden Situationen eingesetzt:**

* Entwicklung neuer Software
* Architekturänderungen
* Verbesserung und Optimierung von Qualitätseigenschaften
* Evaluierung von Architekturalternativen

# **9 Phasen**

1. ATAM präsentieren

* Methode erklären, Erwartungen ermitteln und Fragen klären
* Zweck: Gemeinsames Verständnis der Bewertungsmethode

2. Softwareproduktziele (Geschäftsziele) präsentieren

* Die wichtigsten Funktionen
* Technische, ökonomische, politische Rahmenbedingungen
* Welche Ziele können als primäre Architekturtreiber dienen?

Das sind typischerweise diejenigen Ziele, die die Architekturentscheidungen am stärksten beeinflussen.

* Zweck: Grundlage für die zu erfüllenden Ziele erkennen

3. Zu untersuchende Architektur präsentieren

* Softwarearchitekt erklärt die Architektur und deckt auf, wie die Geschäftsziele adressiert werden
* Zweck: Verständnis für die Architektur schaffen

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

Descriere generată automat

4. Architekturansätze identifizieren

* Welche Ansätze (Architekturmuster) zielen auf die Erfüllung bestimmter Qualitätsziele hin?
* Welche Ansätze wurden eingesetzt in der zu bewertenden Architektur?
* Nur identifizieren, (noch) keine Bewertung

5. Relevante Qualitätsattribute identifizieren

* Wichtigste Qualitätsattribute erkennen (als Fokus für 6.)
* Priorisierung, Ableiten von Szenarien
* Utility Tree erstellen
* Zweck: Grundlage für Analyse bilden

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

Descriere generată automat

6. Architektur untersuchen

Nach Phasen 4-5:

• Zusammenfassung der bisherigen Ergebnisse

• Identifizierung von Kompromissen

• Identifikation von Risiken

• Entwicklung von Empfehlungen

7. Modifikationen und tiefere Analyse

Verbesserungsvorschläge modifizieren und weiter analysieren, um eine

detailliertere Einsicht in potenzielle Auswirkungen und Trade-offs zu erhalten.

8. Architektur untersuchen

* Erneute Bewertung auf Basis der tieferen Analyse
* Ergebnis: Evtl. Vervollständigung der Daten aus Schritt 6

Zweck: Verbindung zw. Architektur und Szenarien herstellen

9. Ergebnisse Präsentieren

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

Descriere generată automat

# **Architekturmuster**

• Bewährte Lösungen für häufig auftretende Probleme beim Architekturentwurf

• Bieten strukturierte Herangehensweisen und vordefinierte Konfigurationen, um spezifische funktionale und nichtfunktionale Anforderungen eines Systems zu erfüllen

• Fokus auf Gesamtstruktur des Systems und das Zusammenspiel von Komponenten