Zurich University of Applied Sciences

School of Engineering

Bachelor of Science (BSc) in Informatik

Modul Advanced Software Engineering 1 (ASE1)

Software Architektur

Software Architekturen und Qualität

Institut für Angewandte Informationstechnologie (InIT)

Walter Eich (eicw) / Matthias Bachmann (bacn)

https://www.zhaw.ch/de/engineering/institute-zentren/init/

Agenda

- 5.1. Lernziele
- 5.2. Bewertung von Softwarearchitekturen
- 5.3. Prototyp und technischer Durchstich
- 5.4. Architekturanalyse

Lernziele

- Begriff der Qualität anhand des Q-Modells 9126/25010 inkl. Haupt- und Untermerkmale erklären
- Taktiken und Praktiken zur Erreichen wichtiger Q-Ziele von Softwaresysteme erklären können
  - Effizienz/Performance/Wartbarkeit, Änderbarkeit, Erweiterbarkeit, Flexibilität
- Qualitative Bewertung von Softwarearchitekturen nach ATAM durchführen
  - Vorgehen bei qualitativer Bewertung erklären und durchführen
  - Erstellung von Szenarien und Qualitätsbäumen erklären und durchführen
  - Analyse von Softwarearchitekturen hinsichtlich Szenarien und Identifikation entsprechender Risiken selbstständig durchführen
- Metriken und andere Messinstrumente kennen, um Architektur beurteilen zu können

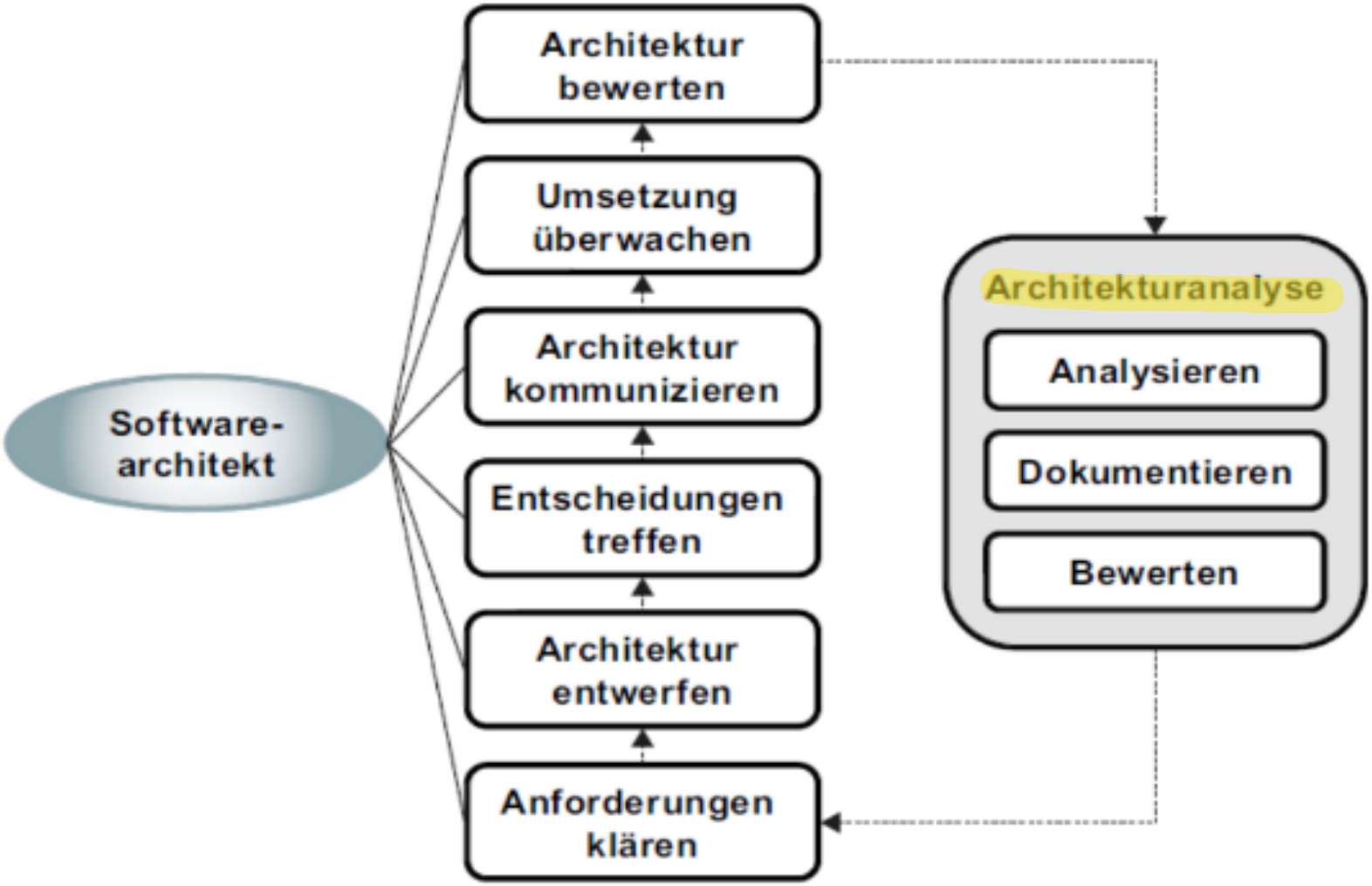
Qualität: Klassische Vorgehensweisen

- Statische Analysen
  - Reviews
  - Code Reviews
  - Architekturreview
  - Dokumenten Review
- Dynamische Analysen
  - Testing (V-Modell) auf verschiedenen Stufen
  - Performance-, Last- oder Stresstests

Architekturanalyse

- Gibt Feedback in Bezug auf
  - Qualitätssicherung
  - Architekturbewertung
  - Überwachung der Umsetzung
  - Effektivität der Entwicklung
  - Verbesserung der Erweiterbarkeit

5. Eingliederung der Architekturanalyse

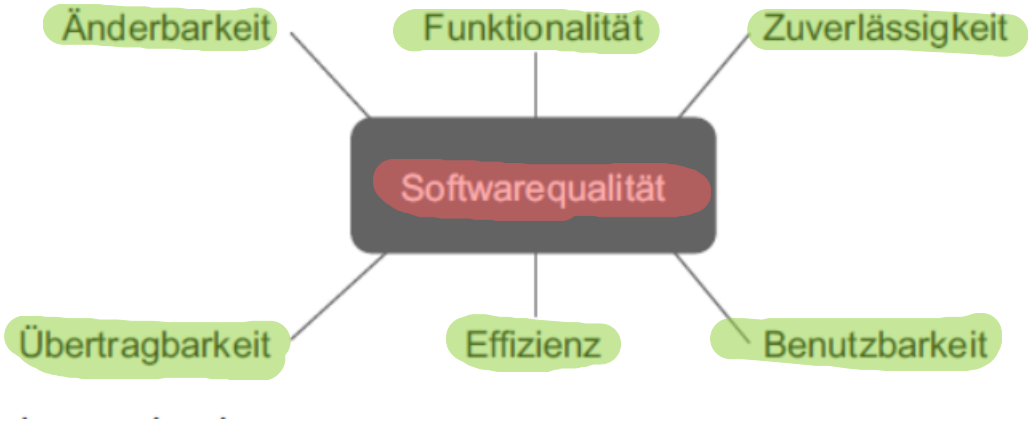


Bewertung

- Zwei Arten können bewertet werden
  - Prozesse wie Entwicklungs- oder Betriebsprozesse
  - Artefakte wie Anforderungen, Quellcode oder andere Dokumente

5.2.1. Qualitative Bewertung

- Qualitätsmodelle bilden die Grundlage für die Bewertung und Analysen
- Qualitätsmodelle sind in Haupt- und Untermerkmalen organisiert.
- ISO 9126 im 2005 abgelöst durch ISO 25000
  - Bezieht sich ausschliesslich auf die Produktqualität



5.2.1.2. Qualitätsmerkmale (nach ISO 9126)

- **Qualitätsmerkmale**
  - **Funktionalität:** Besitzt die Software die verlangten Funktionen?
  - **Zuverlässigkeit:** Kann die Software ihr Leistungsniveau unter festgelegten Bedingungen über einen bestimmten Zeitraum aufrechterhalten?
  - **Benutzbarkeit:** Lässt sich das Programm leicht bedienen und erlernen? Wie reagiert die Software auf Fehleingaben? Benutzbarkeit steht auch für Benutzerfreundlichkeit.
  - **Leistungs-Effizienz:** Wie sparsam ist die Software zur Lösung eines festgelegten Problems bezüglich der Ressourcen, Zeitverhalten bei Anfragen und Bearbeitungen sowie Speicherplatz?
  - **Änderbarkeit:** Wie hoch ist der Aufwand zur Fehlerbeseitigung, zur Umsetzung von Verbesserungen oder zur Anpassung an Veränderungen?
  - **Übertragbarkeit:** Ist die Software auch auf anderen Systemen (Hard- und Software) einsetzbar?

Qualitätsmerkmale nach ISO 9126

Funktionalität	■ Angemessenheit ■ Richtigkeit ■ Interoperabilität ■ Sicherheit ■ Ordnungsmäßigkeit
Zuverlässigkeit	■ Reife ■ Fehlertoleranz ■ Wiederherstellbarkeit
Benutzbarkeit	■ Verständlichkeit ■ Erlernbarkeit ■ Bedienbarkeit
Effizienz	■ Zeitverhalten ■ Verbrauchsverhalten
Änderbarkeit	■ Analysierbarkeit ■ Modifizierbarkeit ■ Stabilität ■ Prüfbarkeit
Übertragbarkeit	■ Anpassbarkeit ■ Installierbarkeit ■ Austauschbarkeit ■ Konformität

5.2.1.2. Qualitätsmerkmale nach ISO 25010

Funktionale Eignung	Funktionale Vollständigkeit Funktionale Richtigkeit Funktionale Angemessenheit
Zuverlässigkeit	Reife Fehlertoleranz Wiederherstellbarkeit Verfügbarkeit
Benutzbarkeit	Erkennbarkeit der Angemessenheit Bedienbarkeit Erlernbarkeit Ästhetik der Benutzeroberfläche Erreichbarkeit Schutz des Benutzers vor Fehleingaben
Leistungseffizienz	Zeitverhalten Verbrauchsverhalten Kapazität
Sicherheit	Vertraulichkeit Integrität Nachweisbarkeit Verantwortlichkeit Authentizität
Kompatibiliät	Koexistenz Interoperabilität
Wartbarkeit	Modularität Wiederverwendbarkeit Analysierbarkeit Modifizierbarkeit Testbarkeit
Übertragbarkeit	Anpassbarkeit Installierbarkeit Austauschbarkeit

Erweiterung ISO 25010 zu 9126

Sicherheit:
Wie gut ist der Datenschutz? Können nicht autorisierte Personen oder Systeme die Daten lesen oder verändern? Wird autorisierten Personen oder Systemen der Zugang zu den Daten auch nicht verwehrt?

Kompatibilität:
Können zwei oder mehr Systeme oder Komponenten Informationen untereinander austauschen bzw. ihre erforderlichen Funktionen ausführen bei Nutzung der gleichen Hardware- oder Softwareumgebung?

5.1.2.5. Taktiken und Praktiken (1)

- Es gibt keine allgemeingültige Methode zur Entwicklung von Lösungsstrategien für bestimmte Qualitätsmerkmale
- Beispiel Performance erhöhen: mögliche Taktiken
  - Lasttests durchführen
  - Zusätzliche Hardware (z.B. für mehr Speicher) bereitstellen
  - Auf Verteilung verzichten
  - Redundanzen einführen
  - Kommunikation der Systemkomponenten verringern
  - Flexibilität des Systems verringern

5.1.2.5. Taktiken und Praktiken (2)

- Flexibilität und Anpassbarkeit konkurriert oft mit der Forderung nach hoher Performance. In welcher Hinsicht muss das System flexibel sein:
  - Funktionalität
  - Datenstrukturen oder Datenmodell
  - Eingesetzte Fremdsoftware
  - Schnittstellen zu anderen Systemen
  - Benutzerschnittstellen
  - Zielplattform

5.1.2.5. Taktiken und Praktiken (3)

- Verbesserung der Flexibilität eines Systems
  - »Information Hiding« betreiben
    - Interne Details einer Komponente vor anderen verbergen
    - Interne Abstraktionsschichten einführen
  - **Abhängigkeiten** verringern
  - Änderungen möglichst lokal und auf wenige Bausteine begrenzt halten
  - Systembestandteile möglichst voneinander **entkoppeln**
    - Bausteine immer über Schnittstellen kommunizieren lassen
    - Adapter, Fassaden oder Proxys verwenden, um Bausteine zu entkoppeln
  - Verständlichkeit des Codes erhöhen

Taktiken und Praktiken (4)

- Zur Realisierung der Nachvollziehbarkeit ist Folgendes erforderlich:
  - eine eindeutige Identifizierung aller Anforderungen,
  - die Auswahl der Art, des Zeitpunkts, der Verantwortung und der Werkzeugunterstützung der Informationserhebung und -verwaltung.

5.2.2. Quantitative Methoden

- Quantitative Bewertung, messen die Artefakte einer Architektur in Zahlen (Metriken)
- Quantitative Methoden zur Qualitätssicherung von Architekturen sind z.B.:
  - Metriken zur Vermessung und Bewertung von Architekturaspekten
  - Analysen, die stärker den Prozess betrachten, wie z.B. Änderungshäufigkeiten
  - Analysen, die die korrekte Umsetzung der Architektur überprüfen, z.B. durch Überprüfung von Architekturregeln im Code
  - Analysen, die das Software-Repository betrachten, wie z.B. Codedubletten (statische Codeanalyse)

5.2.2.1. Überprüfung von Architekturregeln

- Die Architektur bildet gewissermassen den Rahmen eines Softwaresystems und schränkt Design und Implementierung ein.
  - Z.B. Schichten
  - Elemente
  - Beziehung der Elemente
- Zur Prüfung der Architekturregeln gibt es eine Reihe von Methoden (ggf. unterstützt durch Werkzeuge), mit denen sich eine Architekturprüfung durchführen lässt.
  - Statische Analyse (z.B. Sonar oder manuell)
  - Überprüfung ob der Code der Architektur entspricht

5.2.2.2. Metriken (1)

- **Anforderungen:**
  - Anzahl der geänderten Anforderungen pro Zeiteinheit
- **Quellcode:**
  - Abhängigkeitsmasse (Kopplung)
  - Anzahl der Codezeilen (LoC – Lines of Code)
  - Anzahl der Kommentare im Verhältnis zur Anzahl der Programmzeilen
  - Anzahl statischer Methoden
  - Komplexität (der möglichen Ablaufpfade, **zyklomatische** Komplexität, Anzahl der Methoden pro Klasse
  - Vererbungstiefe
  - Anzahl der Testfälle pro Anforderung, **Testabdeckung**

5.2.2.2. Metriken (2)

- **Erstellungsprozess einer Software:**
  - Anzahl der implementierten/getesteten Features pro Zeiteinheit
  - Anzahl der neuen Codezeilen pro Zeiteinheit
  - Zeit für Meetings in Relation zur gesamten Arbeitszeit
  - Verhältnis der geschätzten zu den benötigten Arbeitstagen (pro Artefakt)
  - Verhältnis von Manager zu Entwickler zu Tester
- **Fehler:**
  - Mittlere Zeit bis zur Behebung eines Fehlers
  - Anzahl der gefundenen Fehler pro Paket
- **Test:**
  - Anzahl der Testfälle
  - Anzahl der Testfälle pro Klasse/Paket

5.2.2.2. Metriken (3)

- **Design (von Robert C Martin):**
  - Eingehende Abhängigkeiten (afferent couplings, C_a)
  - Ausgehende Abhängigkeiten (efferent couplings, C_e).
  - Instabilität $I = C_e : (C_e + C_a)$
  - Abstraktheit (Verhältnis abstrakte zu konkrete Klassen) $A = N_a : (N_a + N_c)$

  https://pdfs.semanticscholar.org/18ce/e8682c75f8ed069891f67731d159d3b8c5ff.pdf
  Robert Martin: OO Design Quality Metrics – An Analysis of Dependencies, 1994

- **System (ganz oder teilweise fertig):**
  - Performance-Eigenschaften wie Ressourcenverbrauch oder benötigte Zeit für die Verarbeitung bestimmter Funktionen oder Anwendungsfälle

5.2.2.3. Zyklomatische Komplexität

- Die zyklomatische Komplexität wird auch **M**cCabe-Metrik genannt
- Sie zeigt die **Anzahl voneinander unabhängiger Pfade** eines Softwaremoduls :

$$v(G) = \mathbf{M} = e - n + 2p$$

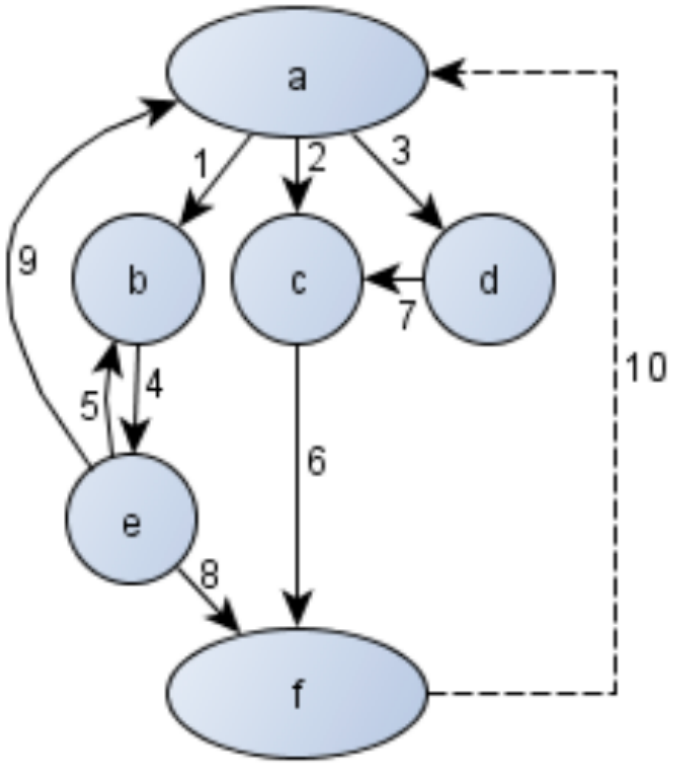
e die Anzahl der Kanten im Graphen,
n die Anzahl der Knoten im Graphen
p die Anzahl der einzelnen Kontrollflussgraphen (ein Graph pro Funktion/Prozedur/Komponente) bezeichnet.

- Die ermittelte McCabe-Zahl ist ein Mass für die Komplexität eines Moduls
- Bestimmt die Anzahl Testfälle -> hohe Zahl = viele Testfälle

5.2.2.3. Beispiel Zyklomatische Komplexität

$M = e - n + 2p$

e die Anzahl der Kanten im Graphen,
n die Anzahl der Knoten im Graphen
p die Anzahl der einzelnen Kontrollflussgraphen (ein Graph pro Funktion/Prozedur/Komponente) bezeichnet.



e = 9 (gestrichelte Linie zählt nicht)
n = 6 (a-f)
p = 1

$M = 9 - 6 + (2 * 1) = 5$

-> 5 Pfade: abef, beb, abea, acf, adcf

-> ab 10 ist die Testbarkeit ausserordentlich schwierig

5.3. Prototyp und technischer Durchstich

Zurich University of Applied Sciences
zh aw School of Engineering

- Relevante Anforderungen bzw. Entwicklungsprobleme können besser geklärt und veranschaulicht werden.
  - In der Softwareentwicklung nennt man diese Vorgehensweise auch Prototyping.
  - Sie führt schnell zu Ergebnissen und ermöglicht ein frühzeitiges Feedback
  - Es gibt Spezielle Werkzeuge für »Rapid Prototyping« (z.B. DSL), zu Deutsch »schneller Prototypenbau«.
- Nachteil:
  - höherer Entwicklungsaufwand
  - geplanter »Wegwerf-Prototyp« wird doch nicht weggeworfen

5.3.2.2. Arten von Softwareprototypen

- Der **Demonstrationsprototyp**
  - … dient zur Auftragsakquisition und verschafft den Beteiligten eine Vorstellung davon, wie das Produkt am Ende aussehen könnte. Wichtig hierbei: Dieser Prototyp wird später weggeworfen!
- Der **Prototyp im engeren Sinne**
  - … wird parallel zur Modellierung des Anwendungsbereichs erstellt und veranschaulicht verschiedene Aspekte der Benutzungsschnittstelle oder Teile der Funktionalität. Dieser Prototyp dient zur Analyse.
- Das **Labormuster**
  - … gehört in den Bereich des experimentellen Prototypings und dient zur Beantwortung konstruktionsbezogener Fragen und Alternativen.
- Das **Pilotsystem**
  - … aus dem evolutionären Prototyping ist dagegen schon ein Kern des Produkts. Die Weiterentwicklung vom Prototyp zum Produkt erfolgt schrittweise in Zyklen unter Beteiligung der Anwender, Entwicklung.

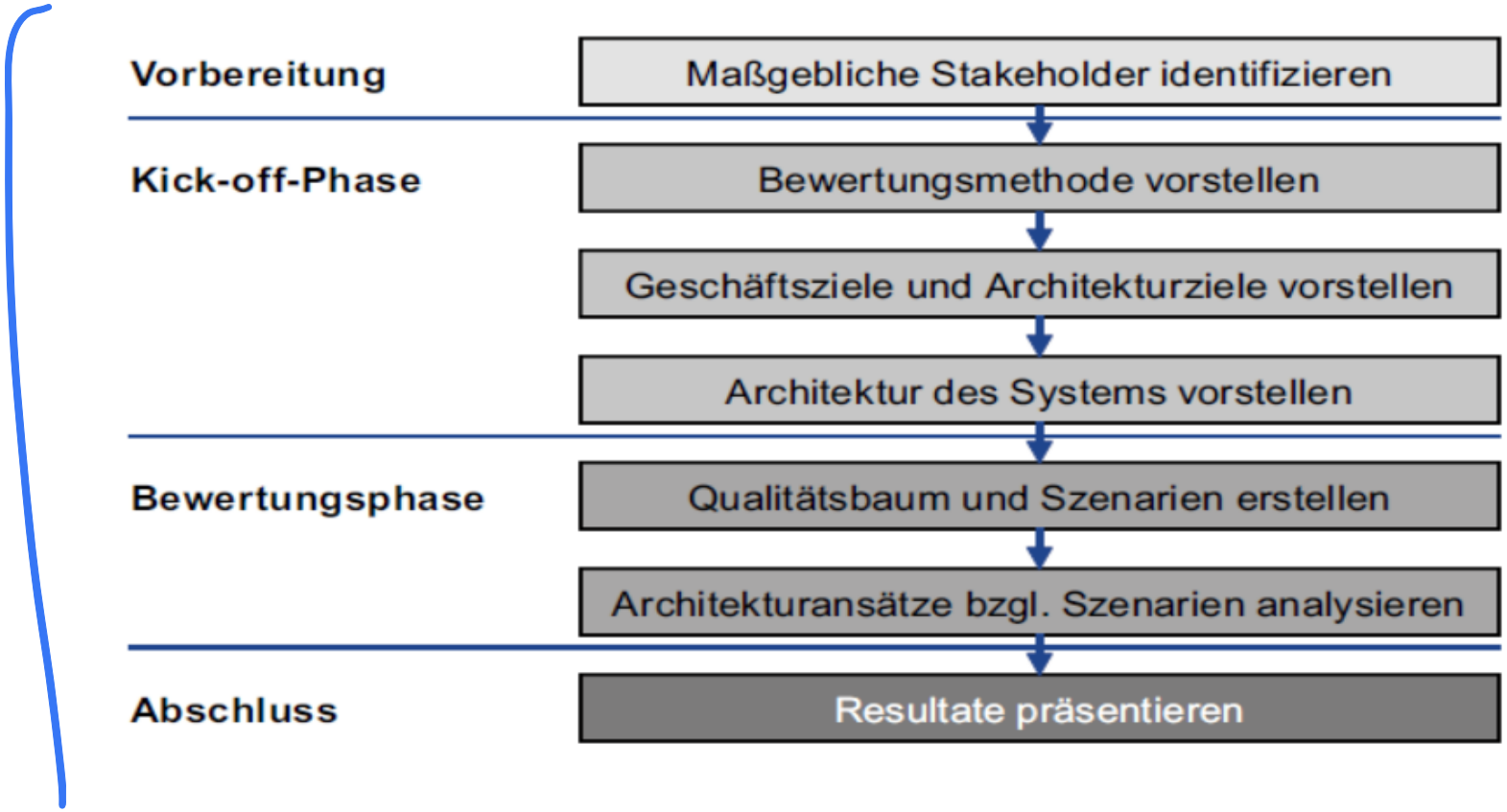
5.4.1. Architekturanalyse - ATAM
Architecture Tradeoff Analysis Method

Zurich University of Applied Sciences
School of Engineering

- Eine Methode zur Architekturanalyse ist die ATAM. ATAM steht für *Architecture Tradeoff (Kompromiss) Analysis Method* und ist ein methodisches Vorgehen zur qualitativen Architekturbewertung.
- **Vorteile von ATAM:**
  - eindeutige Qualitätsanforderungen
  - verbesserte Architekturdokumentation
  - dokumentierte Grundlage für architektonische Entscheidungen
  - frühzeitig identifizierte Risiken
  - verbesserte Kommunikation zwischen den Beteiligten
- **Voraussetzungen für ATAM:**
  - Architekt des Systems (oder technischer Ansprechpartner)
  - Architekturdokumentation
  - Verantwortlicher fachlicher Ansprechpartner oder Auftraggeber

5.4.1. Vorgehen bei der Bewertung nach ATAM (1)

- ATAM teilt die Bewertung einer Softwarearchitektur in vier Phasen ein:

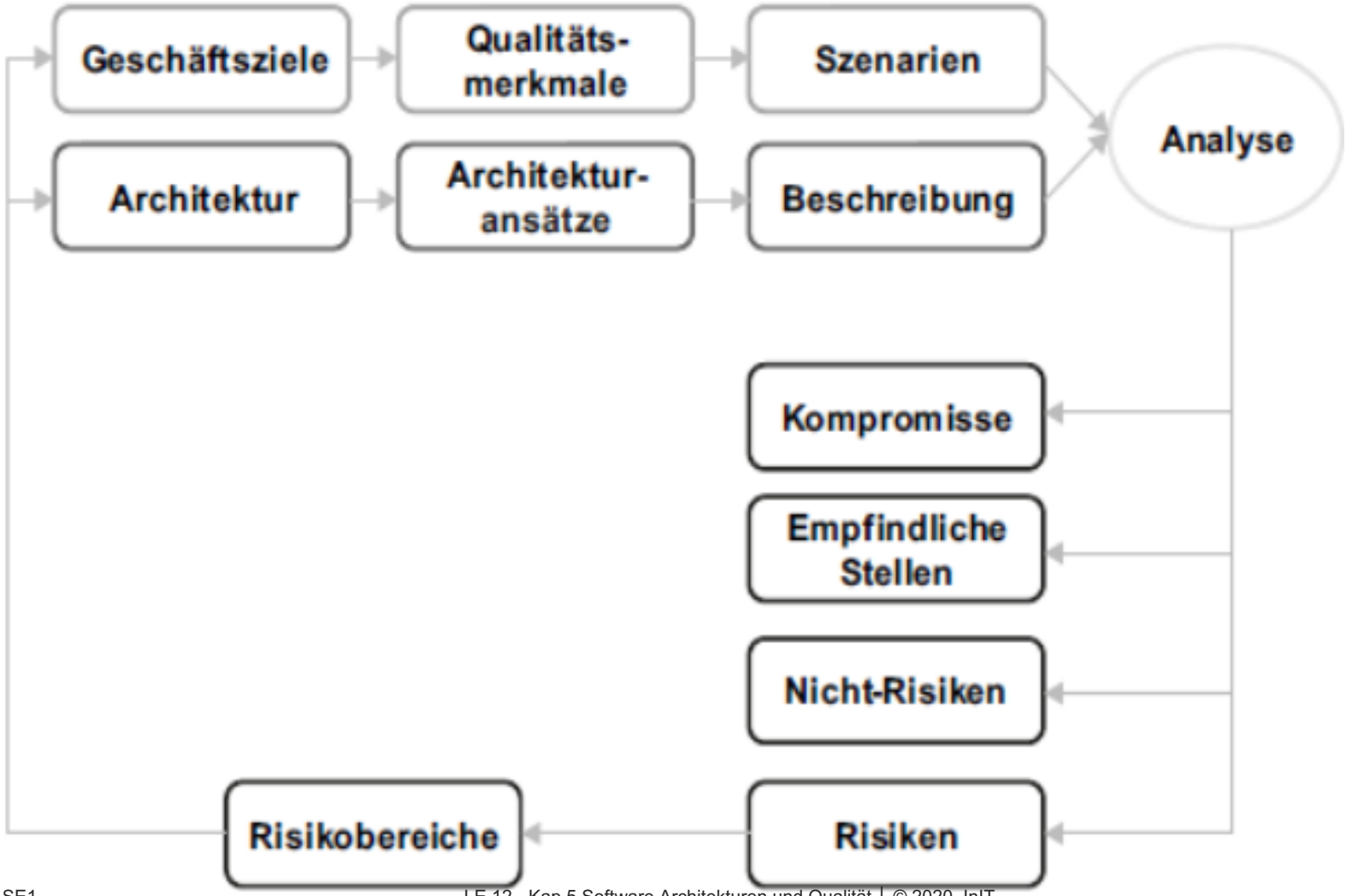


Vorgehen bei der Bewertung nach ATAM (2)

Zurich University of Applied Sciences
School of Engineering

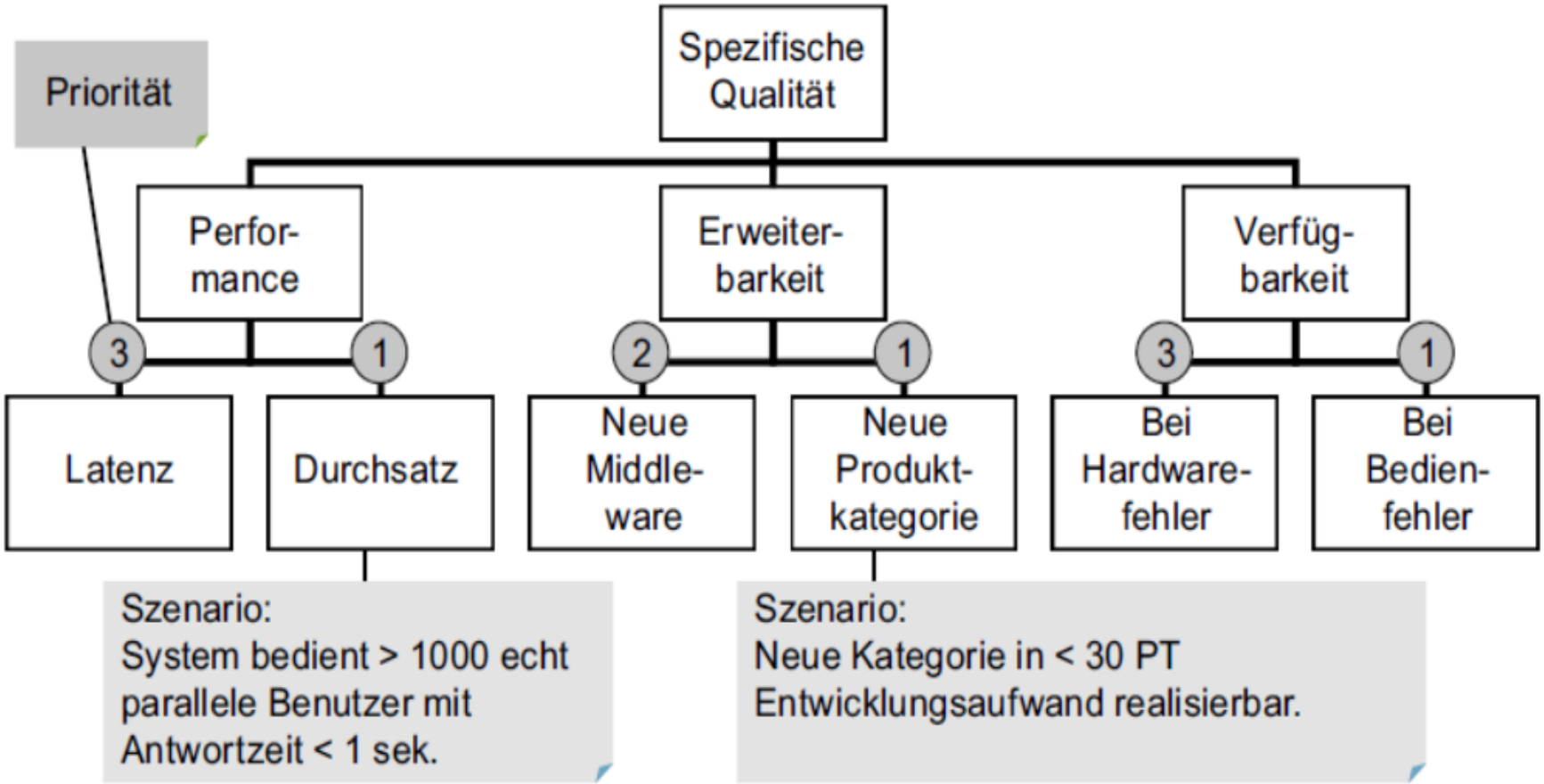
- Stakeholder erarbeiten die wesentlichen geforderten Qualitätsmerkmale
  - hierarchisch in einem Qualitätsbaum anordnen
  - sowie für die wichtigsten Qualitätsziele Szenarien beschreiben
- Nach Analyse der Szenarien sollten alle Entscheidungen in vier Aspekte eingeteilt werden:
  - **Risiken:** Die Risiken sind Teile der Architektur, die je nach Verlauf die Erfüllung von Geschäftszielen gefährden und Probleme bereiten können.
  - **Empfindliche Stellen:** Bei den sogenannten »empfindlichen Stellen« (engl. Sensitivity Points) der Architektur können geringe Änderungen bereits weitreichende Folgen haben.
  - **Kompromisse:** Die Kompromisse oder Trade-offs geben an, ob und wie eine Entwurfsentscheidung eventuell mehrere Qualitätsmerkmale wechselseitig beeinflussen kann.
  - **Nicht-Risiken:** Welche Szenarien werden auf jeden Fall (d.h. risikolos) erreicht?

5.4.1. Vorgehen bei der Bewertung nach ATAM (3)



ATAM - Qualitätsbaum

- Beispiel von Szenarien dargestellt im Qualitätsbaum:



ATAM – Bewertung (1)

- Die Bewertung der Architektur hinsichtlich der Qualitätsmerkmale erfolgt meist in einer kleinen Gruppe gemeinsam mit dem Architekten, und zwar in der Reihenfolge gemäss den Prioritäten.
- Dabei geht es um die Beantwortung einer Reihe verschiedener Fragen:
  - Welche Architekturentscheidungen wurden getroffen, um ein Szenario zu erreichen?
  - Welcher Architekturansatz unterstützt die Erreichung des Szenarios?
  - Welche Kompromisse wurden eingegangen?
  - Wurden andere Qualitätsmerkmale oder Architekturziele beeinflusst?
  - Welche Risiken bestehen?
  - Welche Analysen, Prototypen oder Untersuchungen stützen diese Entscheidung?

ATAM – Bewertung (2)

- Am Ende der Bewertung sollte man einen guten Überblick über verschiedene wichtige Punkte gewonnen haben:
  - Qualität der Architektur in Bezug auf gegebene Szenarien und spezifische Architekturziele
  - Risiken bei der Umsetzung der wichtigsten Szenarien
  - Massnahmen zur Verhinderung der Risiken
  - Szenarien, die ohne Risiken erreicht werden können

ATAM – Szenarien (1) - Arten

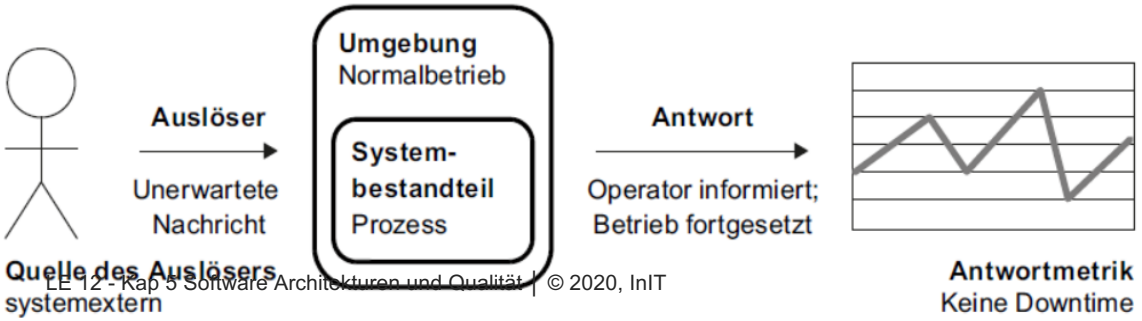
Zurich University of Applied Sciences
School of Engineering

Arten von Szenarien: Es gibt 3 verschiedene Arten von Szenarien:

- **1)** Die Anwendungsszenarien beschreiben, wie das System zur Laufzeit auf bestimmte Auslöser reagieren soll. Hierzu gehören auch Szenarien zur Beschreibung der Effizienz bzw. Performance.
- **2)** Änderungsszenarien beschreiben, was bei einer Modifikation des Systems oder seiner direkten Umgebung passiert, z.B. wenn eine zusätzliche Funktion implementiert werden soll.
- **3)** Die Stress- oder Grenzszenarien beschreiben, wie das System auf Extremsituationen wie z.B. einen Stromausfall reagieren soll.

ATAM – Szenarien (2) - Bestandteile

- Szenarien bestehen aus folgenden Bestandteilen:
  - Der **Auslöser** (stimulus) ist ein bestimmtes Ereignis, z.B. wenn ein Benutzer eine Funktion aufruft oder ein Systemteil ausfällt.
  - Die **Quelle des Auslösers** (source): woher der Auslöser kommt.
  - In der **Umgebung** (environment) wird dargestellt, in welchem Zustand sich das System zum Zeitpunkt des Auslösers befindet.
  - Weiterhin zählt dazu der **Systembestandteil** (artifact), der durch den Auslöser betroffen ist.
  - **Antwort** (response): wird Reaktion auf den Auslöser zurückgeliefert.
  - Die **Antwortmetrik** (response measure) ist ein Bewertungsmodell, um die Qualität der Antwort des Systems zu messen bzw. zu bewerten.



ATAM – Szenarien (3) – Beispiele

Zurich University of Applied Sciences
School of Engineering

- **1) Anwendungsszenarien:**
  - Die Antwort auf eine Angebotsanfrage muss Endbenutzern im Regelbetrieb in weniger als 5 Sekunden angezeigt werden. Im Betrieb unter Hochlast (Jahresendgeschäft) darf eine Antwort bis zu 15 Sekunden dauern, in diesem Fall ist vorher ein entsprechender Hinweis anzuzeigen.
  - Bei der erstmaligen Benutzung des Systems muss ein Benutzer ohne Vorkenntnisse innerhalb von 15 Minuten in der Lage sein, die gewünschte Funktionalität zu lokalisieren und zu verwenden.
  - Bei Eingabe unzulässiger oder fehlerhafter Daten in die Eingabefelder muss das System entsprechende spezifische Hinweistexte ausgeben und danach im Normalbetrieb weiterarbeiten.

ATAM – Szenarien (4) – Beispiele

Zurich University of Applied Sciences
School of Engineering

- **2) Änderungsszenarien:**
  - Die Entwicklung neuer Funktionalitäten muss in weniger als 30 Personentagen möglich sein.
  - Die Unterstützung einer neuen Browser-Version muss in weniger als 30 Personentagen programmiert und getestet werden können.
- **3) Stress- oder Grenzszenarien:**
  - Im Normalbetrieb muss bei einem CPU-Ausfall das Ersatzsystem innerhalb von 15 Minuten verfügbar sein.
  - Bei Ausfall eines Datenbanksystems läuft das System mit der geforderten Performance und Leistung weiter.

ATAM – Szenarien (5) – Beispiele

- Beispielbestandteile für Szenarien zur **Effizienz** (Performance):

Quelle des Auslösers	intern oder extern
Auslöser	Beliebiges Ereignis: periodisch, sporadisch, zufällig, gezielt
Umgebung	Normalbetrieb, Hochlast, Überlast
Systembestandteil	Gesamtsystem
Antwort	Auslöser beeinflusst Ausführungsverhalten (Kann das System nach dem Auslöser vollständig oder nur noch eingeschränkt genutzt werden? Sind Funktionen und Daten eingeschränkt oder vollständig nutzbar?) oder Laufzeitverhalten des Systems (Wie ändern sich Zeitverhalten oder Ressourcennutzung durch den Auslöser?).
Antwortmetrik	Latenz, Reaktionszeit, Durchsatz Fehlerrate, Menge verlorener Daten oder nicht mehr verfügbarer Funktionen, Schwankungen in der Möglichkeit des Systemzugriffs

ATAM – Szenarien (6) – Beispiele

- Beispielbestandteile für Szenarien zur **Zuverlässigkeit**:

Quelle des Auslösers	intern oder extern
Auslöser	Fehler, Aus- oder Wegfall eines Systemteils, korrekte oder inkorrekte Nutzung eines Systemdienstes
Umgebung	Normalbetrieb oder besondere (eingeschränkte) Betriebsbedingungen
Systembestandteil	Gesamtsystem oder beliebige Bestandteile
Antwort	■ Das System entdeckt und behebt den Fehler, benachrichtigt Verantwortliche, schaltet fehlerhafte Bestandteile ab oder ersetzt sie. ■ Das System oder ein Bestandteil geht durch den Auslöser in einen Fehlerzustand über oder stellt die Funktion ein.
Antwortmetrik	■ Zeiten, in denen das System verfügbar oder wiederhergestellt sein muss ■ Zeit, die zwischen der Entdeckung eines Fehlers/Fehlverhaltens und seiner Behebung vergeht ■ Zeit, die das System in eingeschränktem Betriebszustand oder Fehlerzustand verbleiben darf

Fragen