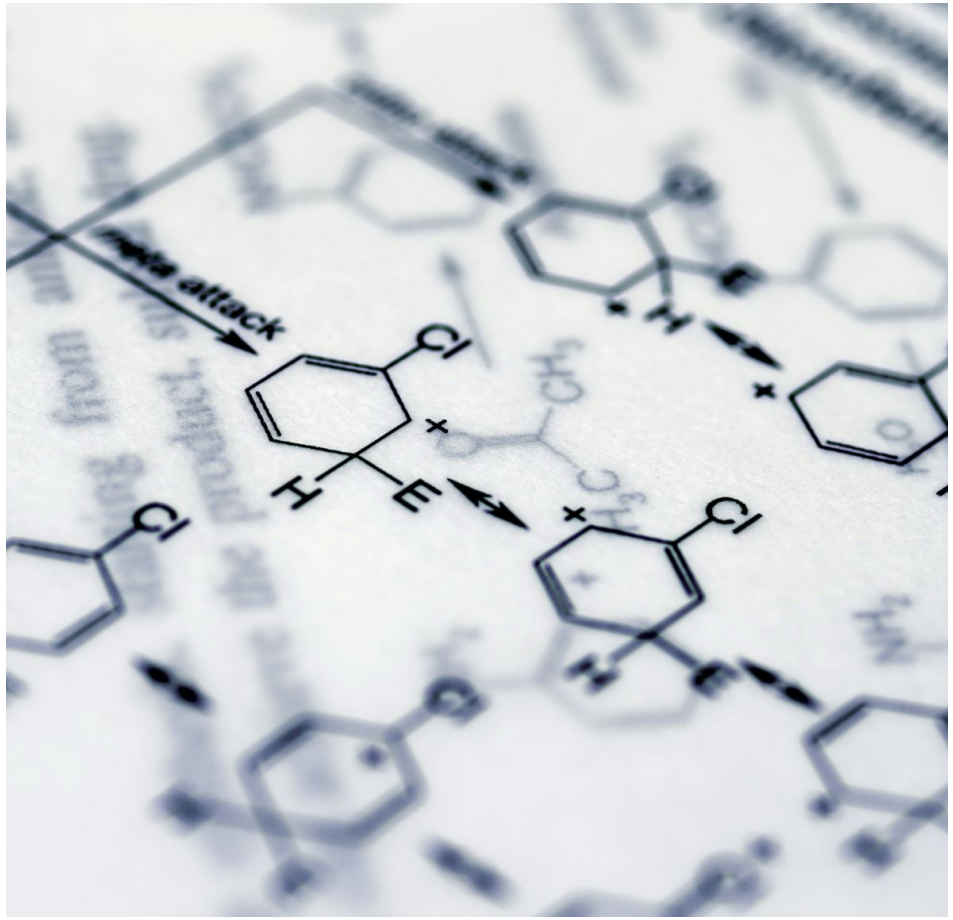
Themenbereiche der Vorlesung



Themenbereiche

A – Grundlegende Konzepte

B – Dokumentation und Kommunikation

C – Entwurf von Softwarearchitekturen

D – Architekturmuster

E – Qualität von Softwarearchitekturen

Qualität von Softwarearchitekturen

Themenbereich E

Inhalte

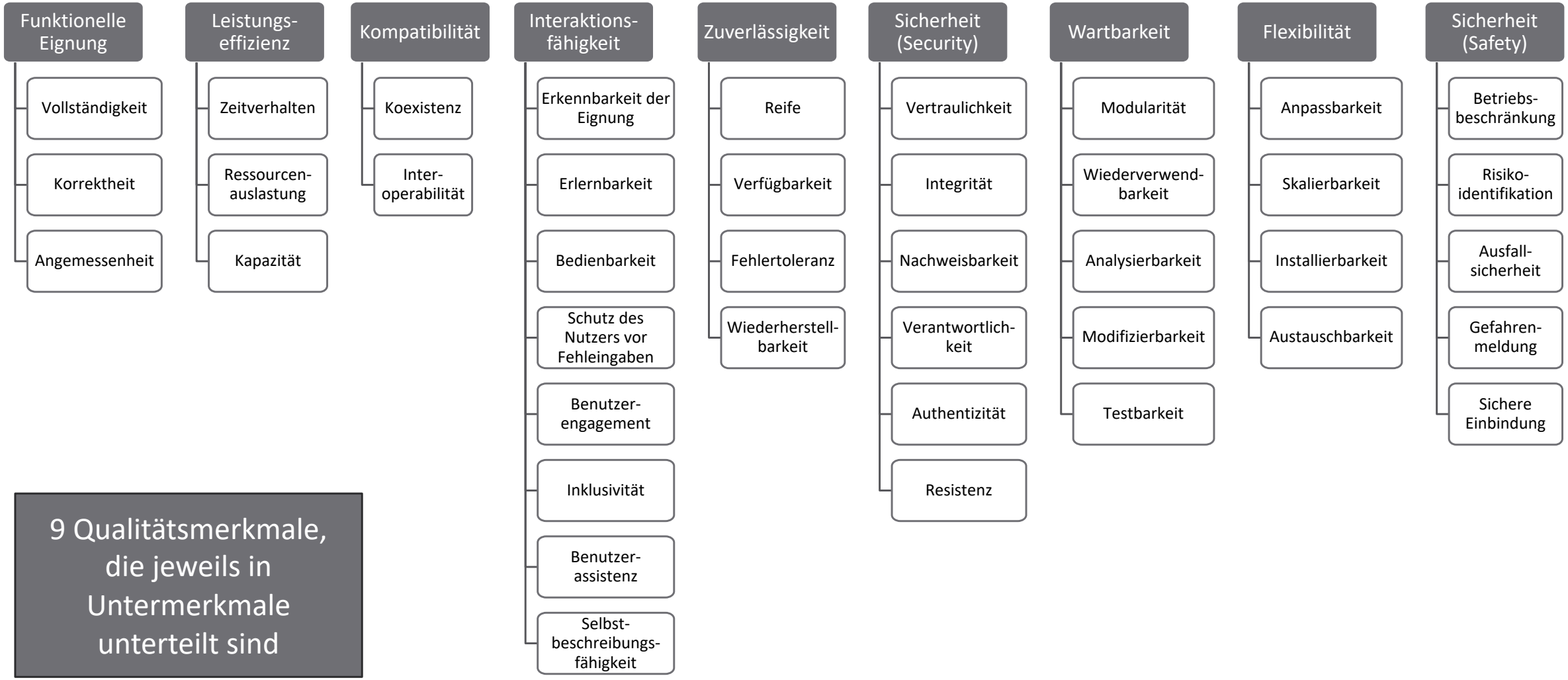
- Qualitätsmerkmale
- Prinzipien der Qualitätssicherung- und Bewertung
- ATAM – Architecture Tradeoff Analysis Method

Qualität von Softwarearchitektur

- Maß, in dem die Architektur eines Softwaresystems die Anforderungen und Erwartungen der Stakeholder erfüllt und gleichzeitig die langfristige Wartbarkeit, Erweiterbarkeit, Performance und andere relevante Qualitätsattribute gewährleistet.
- 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

Qualitätsmerkmale für Software nach ISO/IEC 25010



9 Qualitätsmerkmale, die jeweils in Untermerkmale unterteilt sind

Qualitätsziel und Qualitätsanforderung

Definition: Qualitätsziel

Eine abstrakte Anforderung an ein Qualitätsmerkmal eines Softwareprodukts.

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

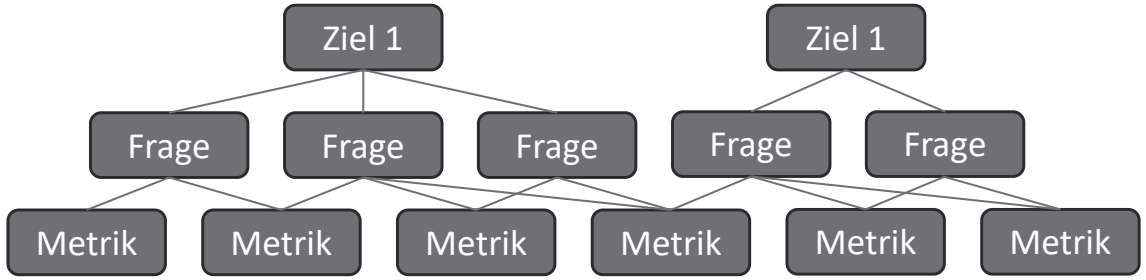
Definition: Qualitätsanforderung

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)

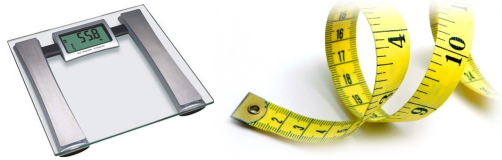
Das GQM Modell

- Framework zur Messung und Verbesserung der Softwarequalität durch strukturierte
  - Zielsetzung (**G**oals)
  - Frageformulierung (**Q**uestions)
  - Metrikdefinitionen (**M**etrics)



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

“You cannot control what you cannot measure“
Tom DeMarco



Quelle: V. R. Basili, G. Caldiera, H. D. Rombach: The goal question metric approach. Encyclopedia of software engineering, S. 528-532, 1994.

Das GQM Modell

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

Quelle: V. R. Basili, G. Caldiera, H. D. Rombach: The goal question metric approach. Encyclopedia of software engineering, S. 528-532, 1994.

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

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

Quantitative Architekturbewertung

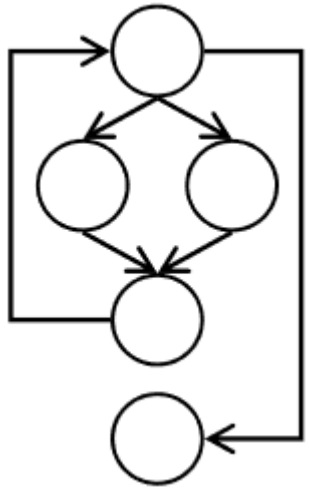
- Merkmale der quantitativen Architekturbewertung
  - Messbarkeit
  - Objektivität
  - Vorhersagbarkeit
  - Automatisierbarkeit

- 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

- Zyklomatische Komplexität V(G) = e – n + 2p
  - e = Anzahl der Kanten
  - n = Anzahl der Knoten
  - p = Anzahl der Zusammenhangskomponenten (unabhängige Teilgraphen, z.B. Methoden)



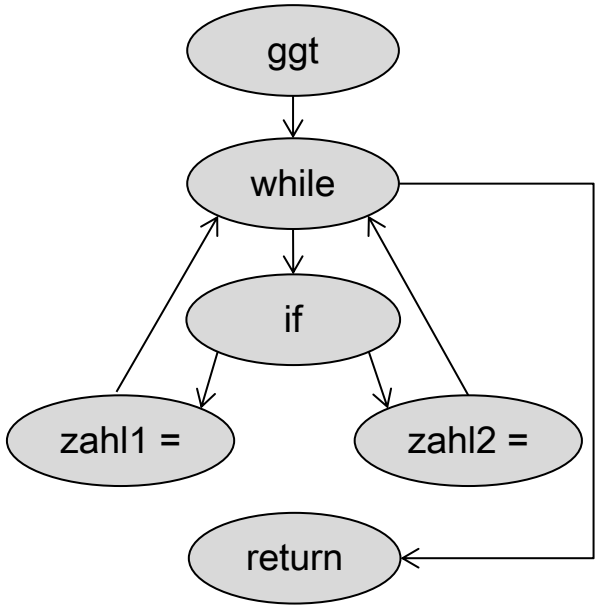
Die zyklomatische Zahl gibt die maximale Anzahl linear unabhängiger Zyklen auf dem Kontrollflussgraphen an.

Quelle: T. J. McCabe: A Complexity Measure. IEEE Transactions on Software Engineering, Band SE-2, S. 308-320, 1976.

Zyklomatische Komplexität

- Beispiel:

```
public static int ggt(int zahl1, int zahl2) {
    while(zahl2 != 0) {
        if(zahl1 > zahl2) {
            zahl1 = zahl1 - zahl2;
        }
        else {
            zahl2 = zahl2 - zahl1;
        }
    }
    return zahl1;
}
```



- Die zyklomatische Komplexität $V(G) = e - n + 2p$ ist hier:

  $7 - 6 + 2 = 3$

Zyklomatische Komplexität

- Klassifikation der zyklomatischen Komplexität *V(G)*:

V(G)	Risiko
1-10	Einfaches Programm, geringes Risiko
11-20	komplexeres Programm, erträgliches Risiko
21-50	komplexes Programm, hohes Risiko
>50	untestbares Programm, extrem hohes Risiko

- *Wenn eine Umstrukturierung ansteht, dann beginne mit der Komponente, die die höchste zyklomatische Komplexität besitzt.*

Qualitative Architekturbewertung

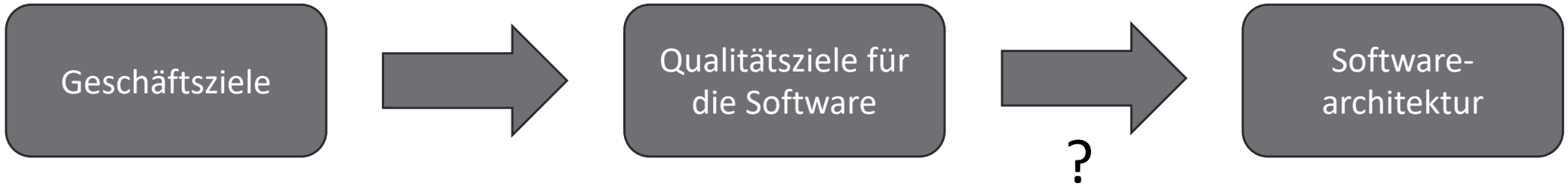
- Merkmale qualitativer Architekturbewertung
  - Subjektivität
  - Komplexität
  - Erfahrung und Expertise notwendig
  - Exploratives Vorgehen

- Beispiel: ATAM – Architecture Tradeoff Analysis Method

ATAM – Architecture Tradeoff Analysis Method

ATAM – Architecture Tradeoff Analysis Method

- 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.



- Erfüllt die Software die Qualitätsziele?
- Was sind kritische Entscheidungspunkte?

ATAM – Architecture Tradeoff Analysis Method

- Übersetzt: „Verfahren zur Analyse von Architekturkompromissen“
- 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

ATAM – Architecture Tradeoff Analysis Method

- Zeigt auf,
  - wie gut die Architektur Qualitätsziele erfüllt,
  - welche Qualitätsziele sich entgegenstehen (tradeoff) und
  - wie die Zusammenhänge zwischen (Geschäfts-)Zielen und Softwarearchitektur sind.
- An der Methode sollten alle wesentlichen Stakeholder beteiligt werden:
  - Projektleiter
  - Architekten
  - Entwickler
  - Kundenvertreter
  - und ggf. weitere Stakeholder

Ausgangssituationen

- ATAM wird in folgenden Situationen eingesetzt:
  - Entwicklung neuer Software
  - Architekturänderungen
  - Verbesserung und Optimierung von Qualitätseigenschaften
  - Evaluierung von Architekturalternativen
  - Rückblick und Lessons Learned

9 Phasen

1. ATAM präsentieren
2. Softwareproduktziele (Geschäftsziele) präsentieren
3. Zu untersuchende Architektur präsentieren
4. Architekturansätze identifizieren
5. Relevante Qualitätsattribute identifizieren
6. Architektur untersuchen
7. Modifikationen und tiefere Analyse
8. Architektur untersuchen
9. Ergebnisse präsentieren

Phasen 1-3: Präsentationen

1. ATAM erklären
   - 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?

     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

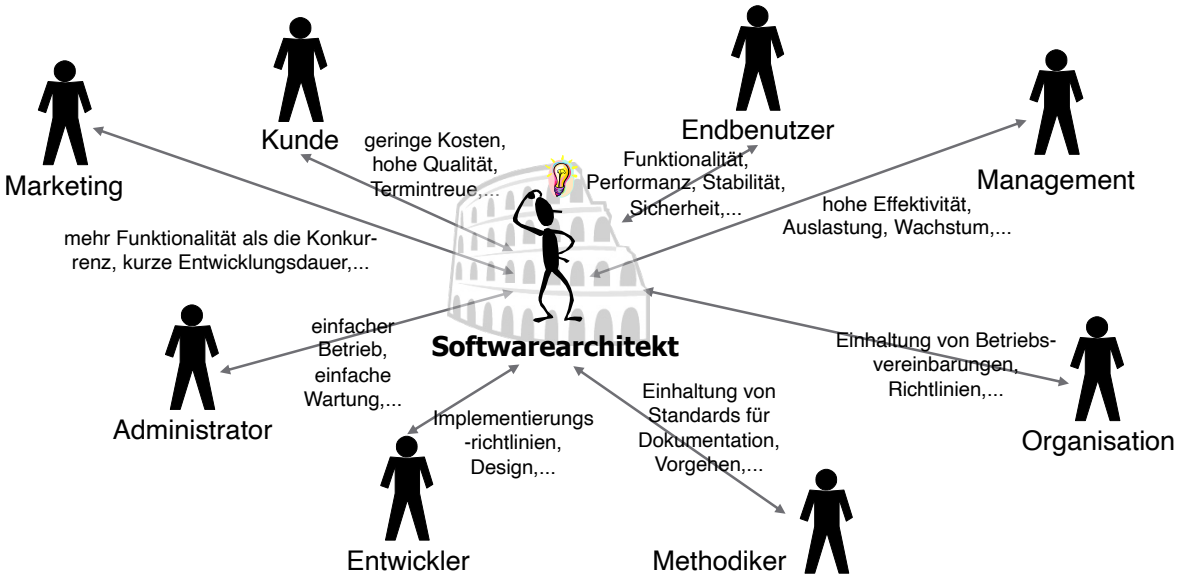
     Zweck: Verständnis für die Architektur schaffen

Geschäftsziele - Beispiele

- **Produktivität**: Die Effizienz der Arbeitsabläufe soll durch die Software gesteigert und die Produktivität der Benutzer verbessert werden.
  - **Qualitätsmerkmale**: Benutzerfreundlichkeit, Leistung, Skalierbarkeit, Zuverlässigkeit
- **Kundenbindung und –loyalität**: Es soll eine starke Kundenbindung aufgebaut werden und die Kundenloyalität erhöht werden, um langfristige Beziehungen aufzubauen und wiederkehrende Einnahmen zu generieren.
  - **Qualitätsmerkmale**: Zuverlässigkeit, Sicherheit, Benutzerfreundlichkeit, Leistung
- **Innovation und Technologieführerschaft**: Es sollen innovative Technologien und Lösungen eingeführt werden, um sich als Technologieführer in der Branche zu positionieren und Wettbewerbsvorteile zu erlangen.
  - **Qualitätsmerkmale**: Flexibilität, Performance, Zuverlässigkeit, Skalierbarkeit

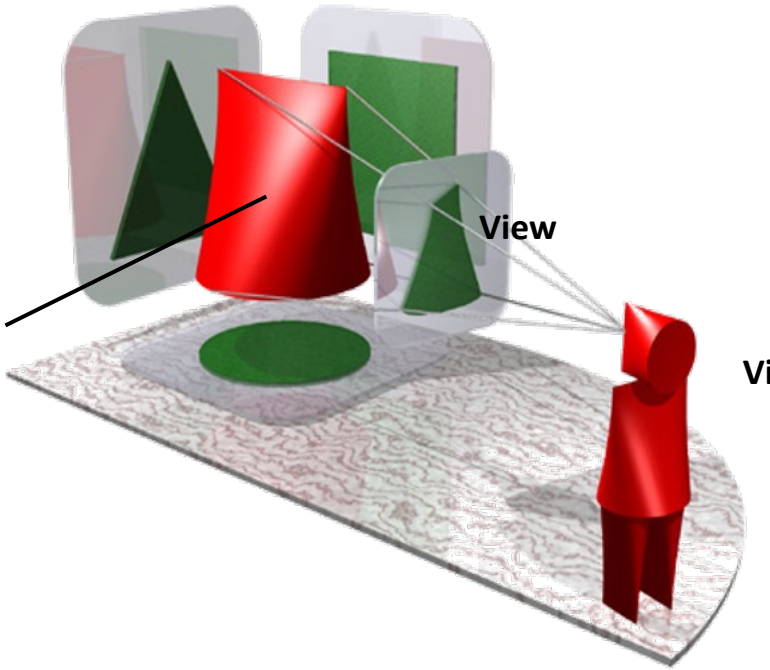
Architekturpräsentation(en)

[ISO/IEC/IEEE 42020:2019]



Zielgruppenorientierte Präsentationen!

Entity of Interest
(zum Beispiel ein Softwaresystem)



Phasen 4 & 5 – Architekturansätze & Qualitätsattribute identifizieren

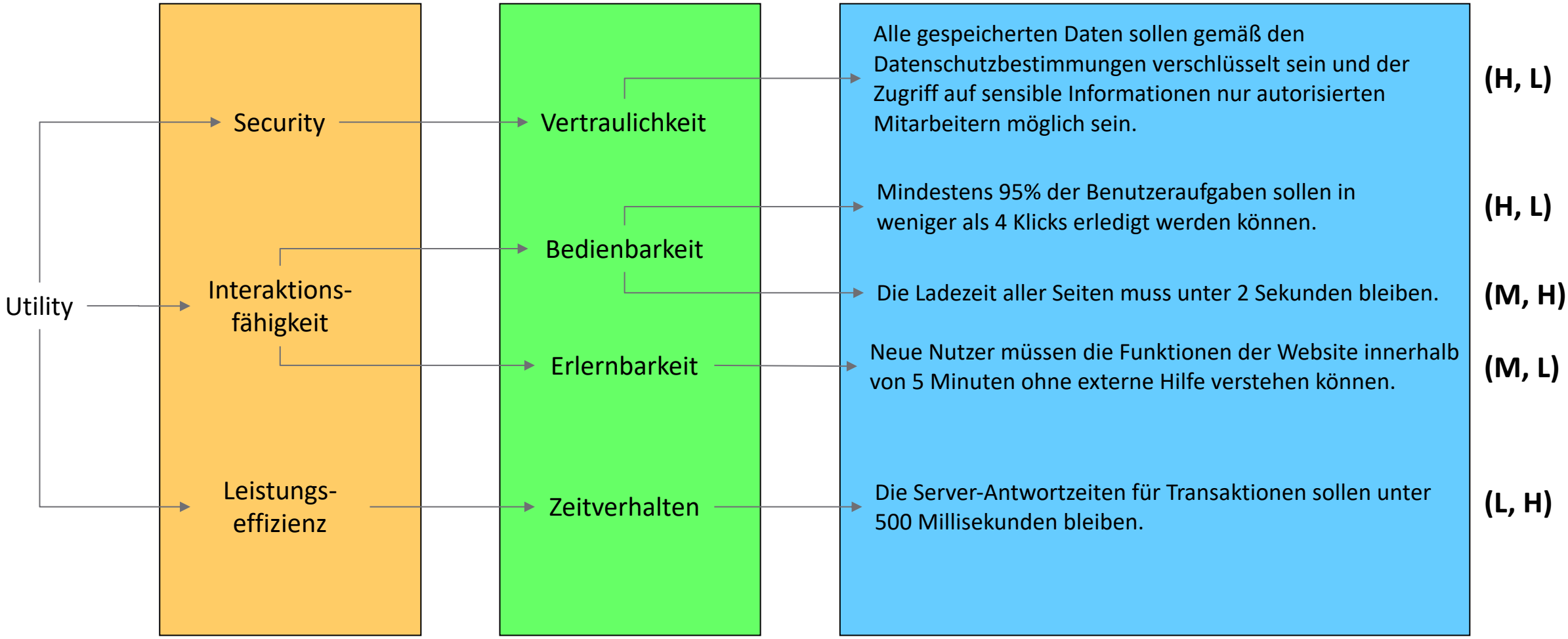
4. Architekturansätze identifizieren
   - Welche Ansätze (Architekturmuster) zielen auf die Erfüllung bestimmter Qualitätsziele hin?
   - Welche Ansätze wurden eingesetzt in der zu bewertenden Architektur?
   - Nur identifizieren, (noch) keine Bewertung

     Zweck: Den Qualitätsanspruch erfüllende Ansätze/Muster erkennen

5. Relevante Qualitätsattribute identifizieren
   - Wichtigste Qualitätsattribute erkennen (als Fokus für 6.)
   - Priorisierung, Ableiten von Szenarien
   - Utility Tree erstellen

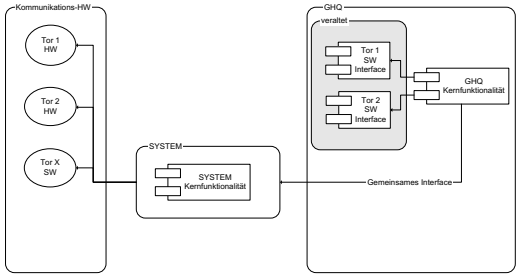
     Zweck: Grundlage für Analyse bilden

Phase 5: Utility Tree als Ergebnis



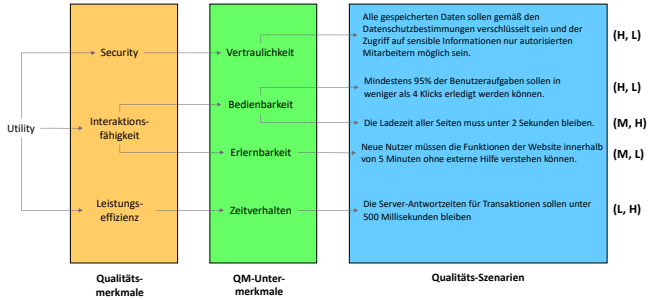
(Priorität aus Geschäftssicht, Schwierigkeit der Umsetzung):
High, **M**edium, **L**ow

Phase 6 – Architektur untersuchen



Architektur (Phase 4)

&



Utility Tree (Phase 5)

- Zusammenfassung der bisherigen Ergebnisse
- Identifizierung von Kompromissen
- Identifikation von Risiken
- Entwicklung von Empfehlungen

Phasen 7 - 9 – Verfeinerung & Ergebnisse präsentieren

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