

Vorlesung 5

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 qualitative 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 → 9 Merkmale

Funktional Eignung

- Vollständigkeit
- Korrektheit
- Angemessenheit

• Leistungseffizienz

- Zeitreihalter
- Ressourcerauslastung
- Kapazität

• Kompatibilität

- Koexistenz
- Interoperabilität

• Interaktionsfähigkeiten

- Erkennbarkeit der Eignung
- Sehbarkeit
- Bedienbarkeit
- Schutz des Nutzers vor Fehlern
- Benutzerengagement
- Inklusivität
- Benutzerassistenz
- Selbstbeschreibungsfähigkeit

• Zuverlässigkeit

- Reife
- Verfügbarkeit
- Fehlertoleranz
- Wiederherstellbarkeit

Sicherheit (Security)

- Vertraulichkeit
- Integrität
- Nachweisbarkeit
- Verantwortlichkeit
- Analysierbarkeit
- Modifizierbarkeit
- Testbarkeit

Flexibilität

- Anpassbarkeit
- Skalierbarkeit
- Installierbarkeit
- Austauschbarkeit

Sicherheit (Safety)

- Betriebesbeschränkung
- Risikoidentifikation
- Ausfallsicherheit
- Gefahrenmeldung
- Sicherer Einbindung

Qualitätsziel

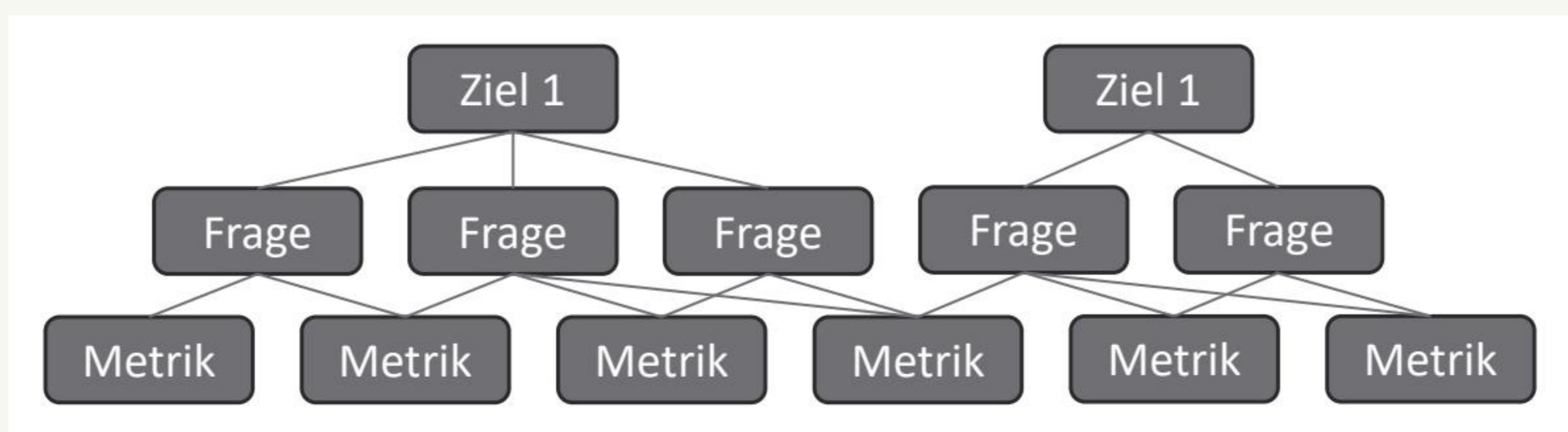
= eine abstrakte Anforderung an ein Qualitätsmerkmal eines Softwareprodukts
z.B.: „Die Software soll möglichst wenig Ressourcen verbrauchen“,
→ Qualitätsmerkmal Leistungseffizienz

Qualitätsonforderung

= eine konkrete und messbare Anforderung an ein bestimmtes Produktmerkmal, die sich auf ein Qualitätsmerkmal eines Softwareprodukts auswirkt
z.B.: „Jeder Task soll immer in höchstens 24 abgeschlossen sein“,
→ Qualitätsmerkmal Leistungseffizienz,
Untermerkmal Zeitverhalter

Das GQM Modell

- Framework zur Messung und Verbesserung der Softwarequalität durch strukturierte Zielsetzung (Goals)
 - Zielformulierung (Questions)
 - Frageformulierung (Questions)
 - Metriksdefinitionen (Metrics)



- Messungen und Analysen in Softwareprojekten sollen so zielgerichtet und nachvollziehbar gestaltet werden
"you cannot control what you cannot measure" Tom DeMarco

1. Ziele (Goals)

- Festlegung klarer Qualitätsziele für das Softwareprojekt
- Beispiel: → Verbesserung der Benutzerfreundlichkeit, Erhöhung der Systemzuverlässigkeit, Reduziert der Fehlerquote

2. Frager (Questions)

- Basierend auf den Zielen: Formulierung von Fragen, die helfen sollen, der 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. Metriker (Metrics)

- Basierend auf den Fragen: Auswahl geeigneter Metriker, um den Fortschritt zu messen und die Antworten auf die Frage zu quantifizieren
- Beispiele:
 - Anzahl der Benutzerinteraktionen pro Stunde, Anzahl der behobenen Fehler pro Woche, durchschnittliche Antwortzeit des Systems

Prinzipien der Qualitätssicherung und Bewertung

Analytische Qualitätssicherung

• Ziel:

- > Identifizierung potentieller Fehler oder Schwachstellen sowie Bewertung der Qualität

• Maßnahmen:

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

Konstruktive Qualitätssicherung

- Ziel:
 - Gewährleister der geforderter Qualität bereits während der Entwicklung
 - Konzentration auf proaktive Ansätze

- Maßnahmen:
 - Modularisierung und Kapselung
 - Architekturmuster und Design Patterns
 - Coding Standards und Best Practices
 - Automatisierte Tests
 - Continuous Integration und Continuous Deployment
 - Continuous Testing und Refactorings
 - Dokumentation
 - Regelmäßige Architekturentwicklungen und -prüfungen
 - Prototyping
 - Design-by-Contract

Quantitative Architekturbeurteilung

• Merkmale der quantitativer Architekturbeurteilung

- Messbarkeit
- Objektivität
- Verhessbarkeit
- Automatisierbarkeit

• Beispiele:

- Zyklometrische 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

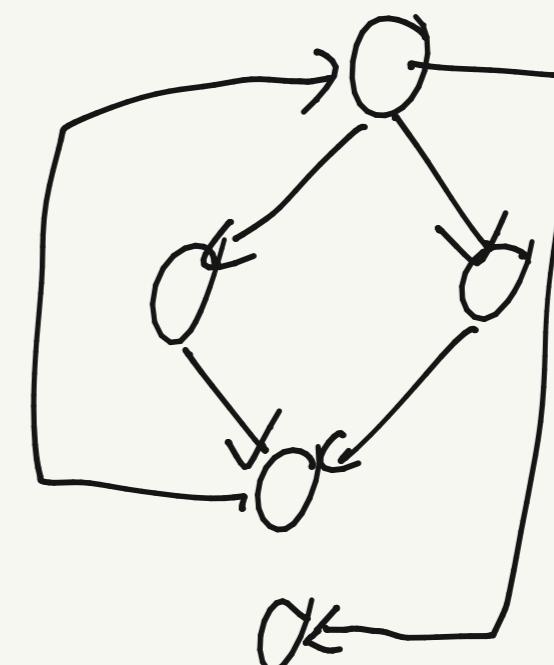
- Zyklomatische Komplexität $V(G) = e - n + 2p$

$\rightarrow e$ = Anzahl der Kanten

$\rightarrow n$ = Anzahl der Knoten

$\rightarrow p$ = Anzahl der Zusammenhangskomponenten
(unabhängige Teilgraphen, z.B. Methoden)

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



- 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

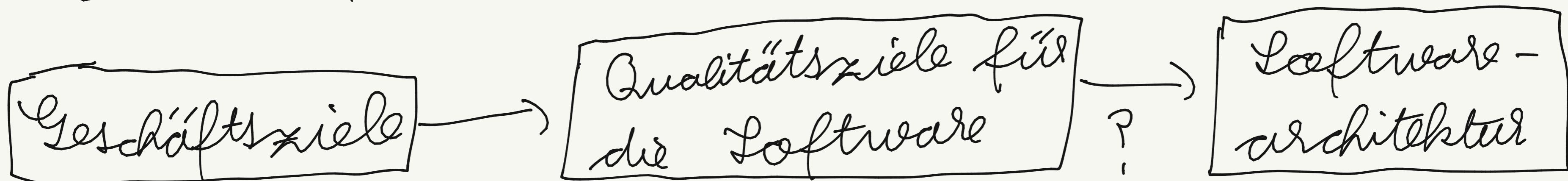
- Wenn eine Umstrukturierung entsteht, dann beginne mit der Komponente, die die höchste zyklomatische Komplexität besitzt

Qualitative Architekturbeurteilung

- Merkmale qualitativer Architekturbeurteilung
 - Subjektivität
 - Komplexität
 - Erfahrung und Expertise notwendig
 - Exploratives Vorgehen
- Beispiel: ATAM - Architecture Tradeoff Analysis Method

ATAM - Architecture Tradeoff Analysis Method

- Strukturierte, Szenario-basierte Architekturbeurteilung
- Führende Methode im Bereich der Architekturbeurteilung
- 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?

- 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 Geschäftsziel verstehen
 - Darauf basierend die Architektur bewertet

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 Architektonal Qualitätsziele erfüllt,
 - welche Qualitätsziele sich entgegenstehen (tradeoffs) und
 - wie die Zusammenhänge zwischen (Geschäfts-) Zielen und Softwarearchitektur sind
- An der Methode sollten alle wesentlichen Stakeholdern beteiligt werden:
 - Projektleitern
 - Architekten
 - Entwickler
 - Kundenvertreter
 - und ggf. weitere Stakeholder

ATAM wird in folgender Situationen eingesetzt:

- Entwicklung neuer Software
- Architekturänderungen
- Verbesserung und Optimierung von Qualitätseigenschaften
- Evaluierung von Architekturalternativen
- Rückblick und Lessons Learned

9 Phasen:

1. ATAM präsentieren
2. Softwareproduktziile (Geschäftsziele) präsentieren
3. Zu untersuchende Architektur präsentiert
4. Architektursätze 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
 - Methoden erklären, Erwartungen ermitteln und Fragen stellenZweck: Gemeinsames Verständnis der Bewertungsmethode
2. Softwareproduktziile (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 addresiert werdenZweck: 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

Phaser 4 & 5 - Architekturansätze & Qualitätsattribute identifizieren

4. Architekturansätze identifizieren

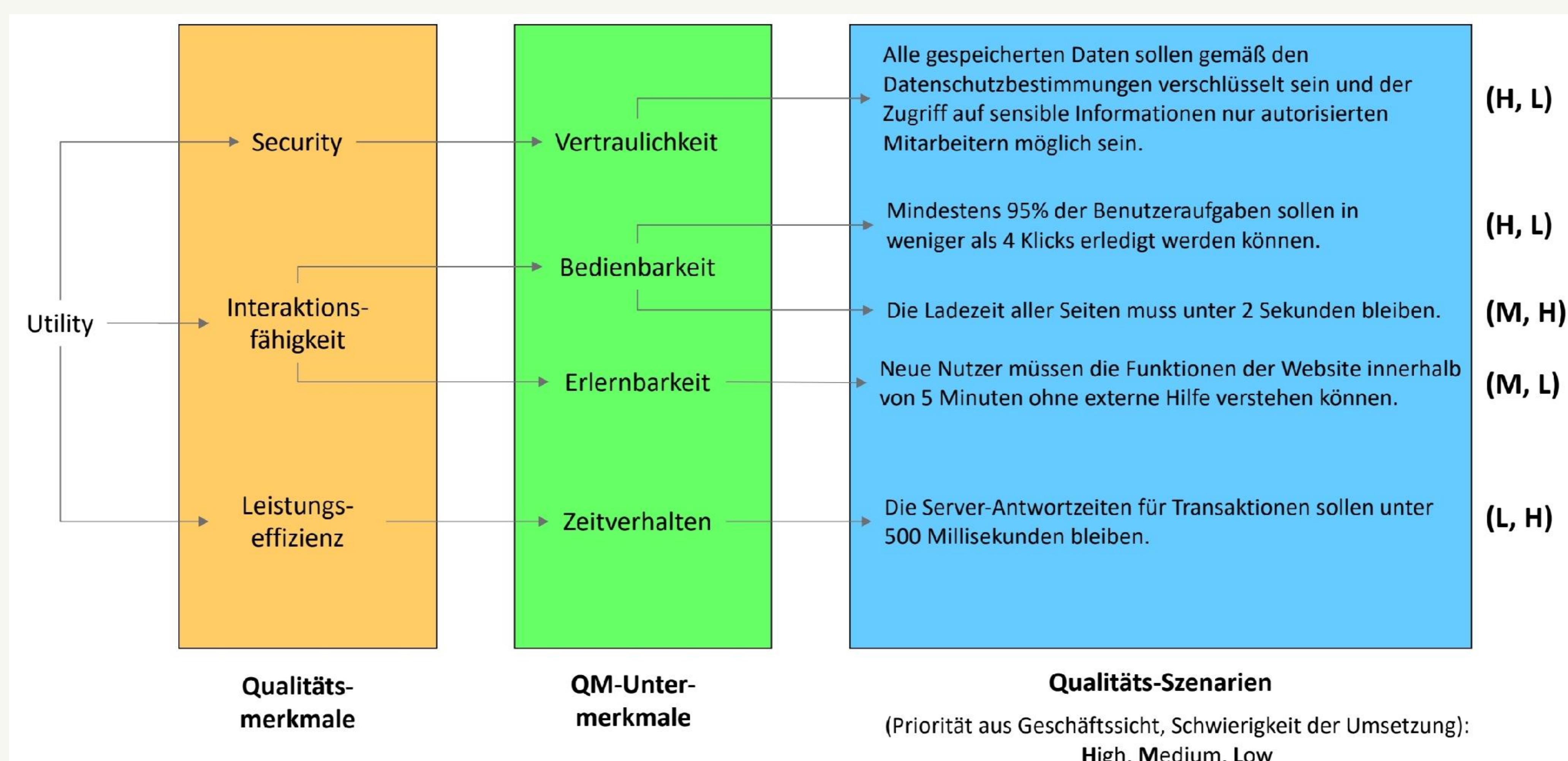
- Welche Ansätze (Architekturmuster) zielen auf die Erfüllung bestimmter Qualitätsziele hin?
- Welche Ansätze werden eingesetzt in der bewertenden Architektur?
- Nur identifizierer, (noch) keine Bewertung
- Zweck: Der Qualitätsanspruch erfüllende Ansätze / Muster erkennen

5. Relevante Qualitätsattribute identifizieren

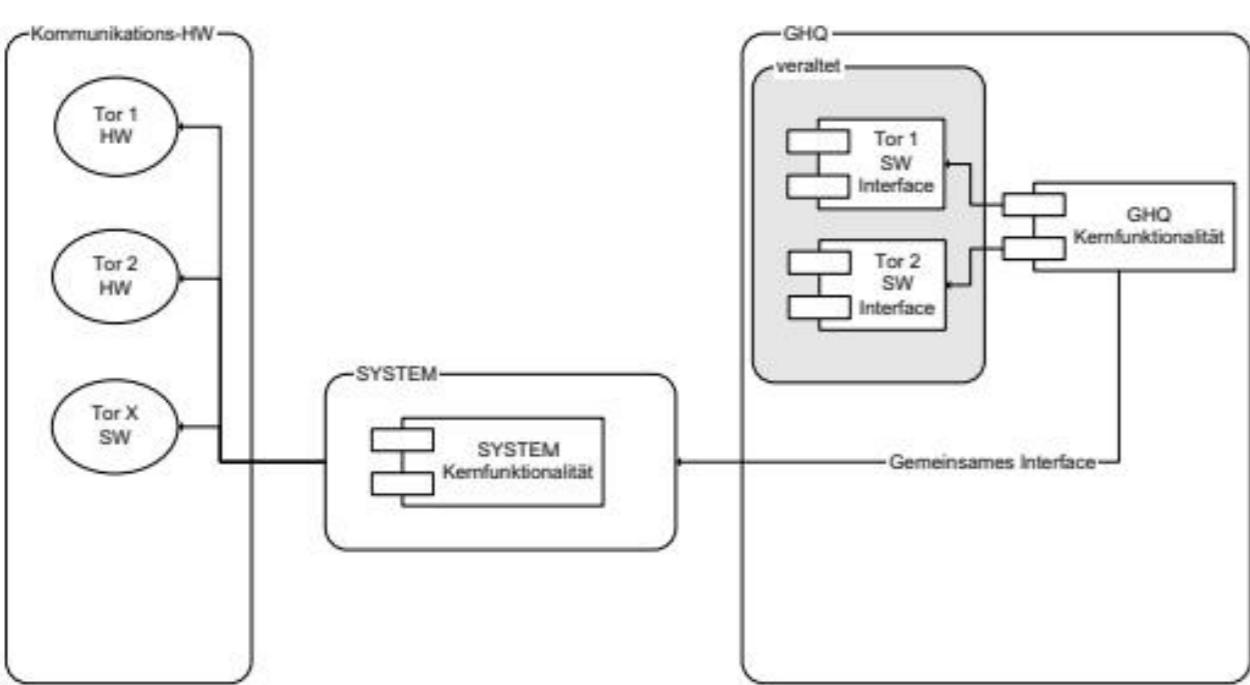
- Wichtigste Qualitätsattribute erkennen (als Fokus für)
- Priorisierung, Ableiter von Szenarien

→ Utility Tree erstellen

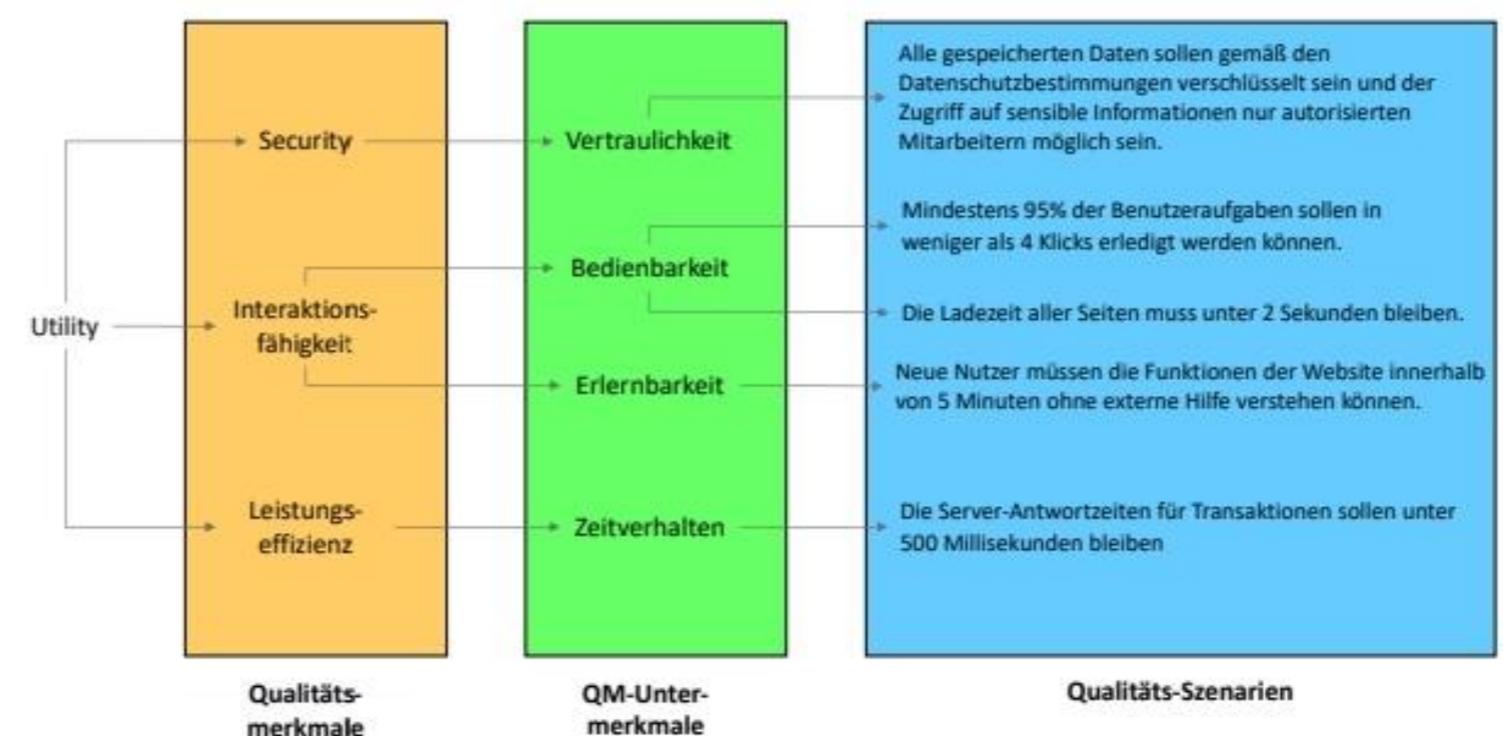
Zweck: Grundlage für Analyse bilden



Phase 6 – Architektur untersuchen



&



Architektur (Phase 4)

Utility Tree (Phase 5)

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

Phasen 7-9: Verfassung und Ergebnisse präsentieren
Phasen 7-9: Verfassung und Ergebnisse präsentieren

7. Modifikationen und tiefere Analyse
→ Verbesserungsvorschläge modifizieren und weiter analysieren, um eine detaillierte Einsicht in potenzielle Auswirkungen und Trade-offs zu erhalten

8. Architektur untersuchen

→ Erneute Bewertung auf Basis der tiefener Analyse

→ Ergebnis: Eventuell Vervollständigung der Daten aus Schritt 6
Zweck: Verbindung zwischen Architektur und Szenarien Hersteller

9. Ergebnisse Präsentieren