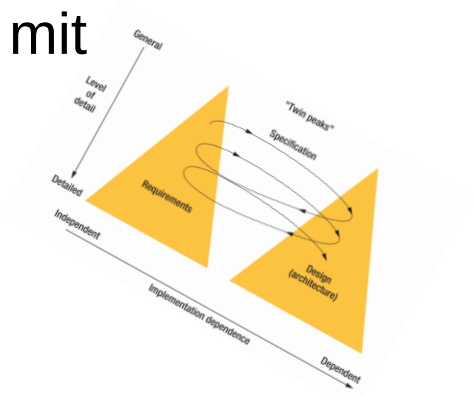
Die Analyse

Phase II

Komponenten in dieser Phase nicht unmittelbar erkennbar …

Analyzes user needs and develops user requirements. Create a detailed functional requirements document.

- Die Analyse beschäftigt sich eher mit dem „**Was** wollen wir bauen“ und **nicht** mit dem „**Wie** wollen wir es bauen“
- Wie wir aus dem letzten Kapitel aber wissen: es besteht eine **Wechselwirkung** (Stichwörter wie Machbarkeit)
- Es kann daher schon in dieser Phase ein Software-Architekt (Meister der Komponenten) zeitweise mit einbezogen werden



Aus der Cafe-Küche: Need, Requirement?

- Großes Internet, viele Meinungen
  - „Requirements are what need to be done in order to achieve the need or goal.“
  - „Needs are high-level requirements that are segregated into lower-level and more detailed requirements.“
- *„Business analysis is the practice of enabling **change in an enterprise** by defining **needs** and recommending solution**s** that deliver value to stakeholders.”*
- *“The need is the objective, and the requirement is the decision about whether to do something to achieve that objective. A **need turns into a requirement** when someone recognizes that having the unmet need is unacceptable and decides he requires the need to be met.”* (Kupe Kupersmith, Paul Mulvey, Kate McGoey. Business Analysis For Dummies. John Wiley & Sons, Inc. USA, 2013)

Exkurs: Probleme lösen

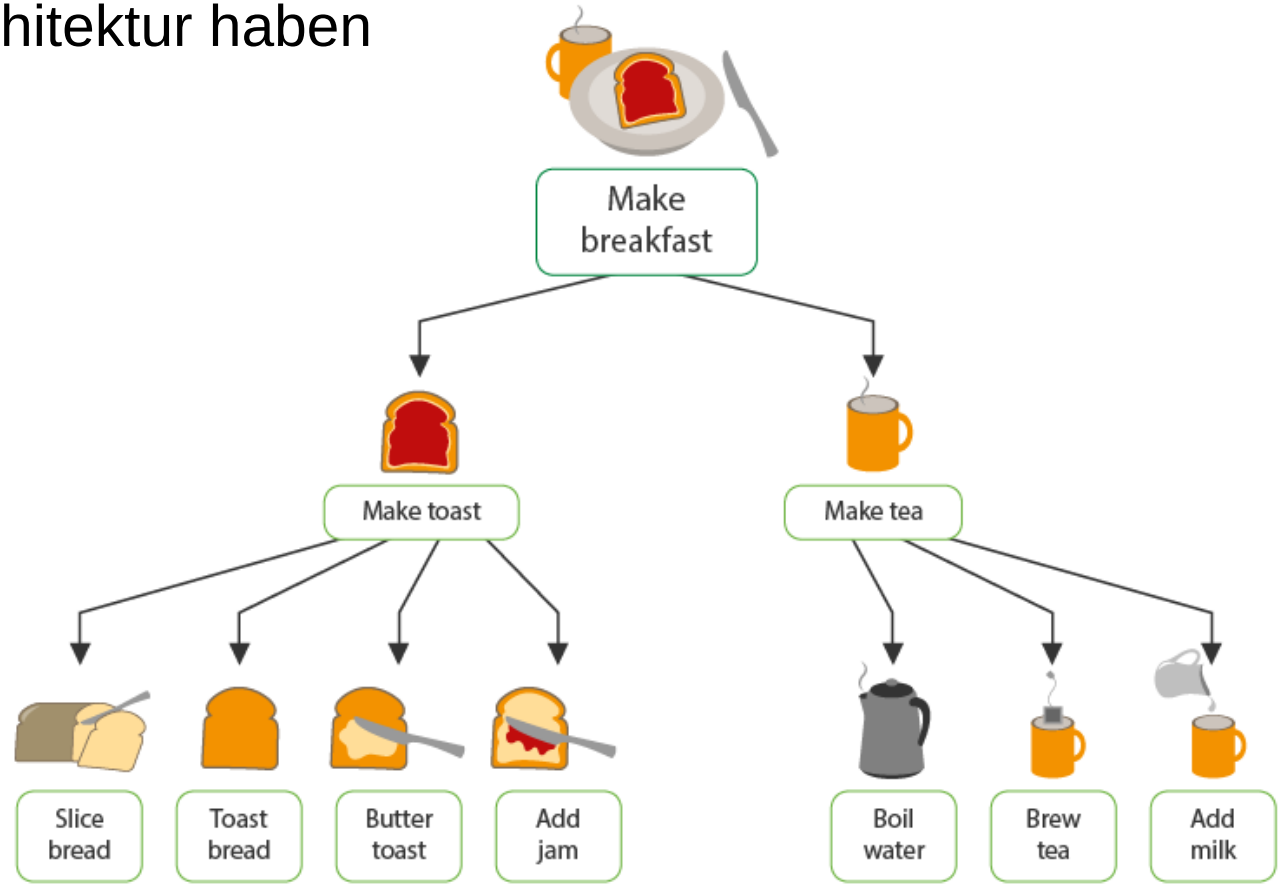
- Anmerkung: persönliche Erfahrung gemischt mit Literatur-Empfehlungen

Exkurs: Probleme lösen

- Stakeholder werden selten strukturiert das „Problem“ wiedergeben (können) → der Versuch, Anforderungen einfach zu transkribieren wird scheitern
- Um eine technische Lösung aus einem realen „Problem“ kreieren zu können → tiefes fachliches Verständnis erforderlich
- Hilfsmittel:
  - Keine Features sondern End-2-End Szenarien (hier schon erste Anzeichen von Komponenten)
  - Beobachten → Iterativ an das Problem herangehen (Feedback Cycle)
  - Stakeholder Gruppen bewusst mischen und auch nicht (kann zu chaotischen Diskussionen führen „weil meine Anforderungen sind wichtiger“ oder auch Sichtweisen verschmelzen)
  - Problem Decomposition (hier schon erste Anzeichen von Komponenten)

Exkurs: Problem Decomposition

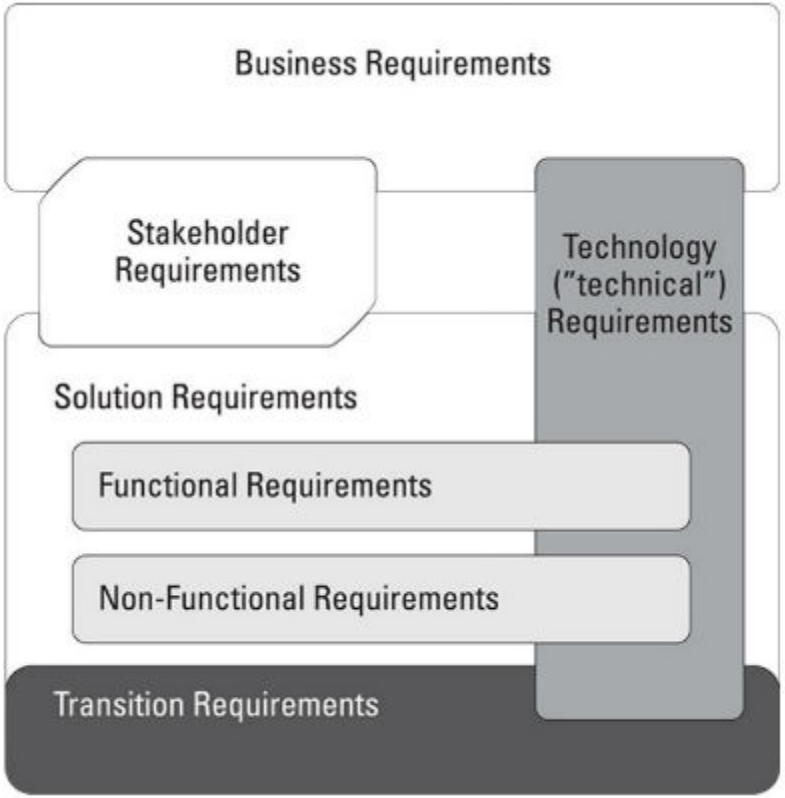
- Wichtig: Es ist eine Kunst, Stakeholder vor dem Verheddern in „Edge Cases“ zu bewahren → wichtig ist es aber, diese nicht zu ignorieren – können Einfluss auf Architektur haben



https://www.testandtrack.io/upload/decomposition1.png

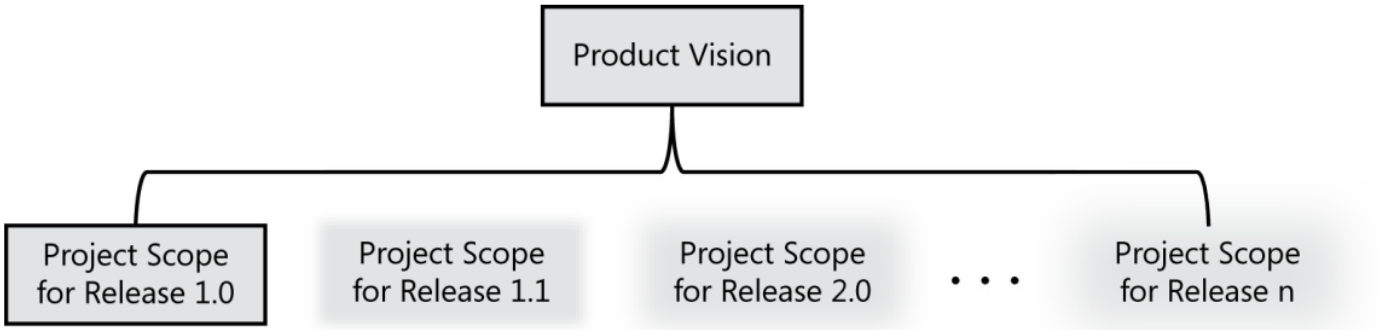
“Wo“ befinden sich die Anforderungen der Software Architektur?

- Funktionale Anforderungen
- Nicht-Funktionale Anforderungen



Business requirements

- Sind vom „Business Need“ abgeleitet
- Mit anderen Worten: Was wollen wir bauen, damit wir dem Unternehmen / Business etwas nützliches bieten können?
- Wichtig um die Vision und den Scope des Projekts auszudrücken
- Beispiel siehe https://www.site.uottawa.ca/~bochmann/SEG3101/Notes/COS_vision_and_scope.doc



Stakeholder requirements

- Beschreibungen Anforderungen aus Sicht der Stakeholder
- Achtung: (Technical-)Solution independent!
- Beispiele:

Airport check-in kiosk	Check in for a Flight Print Boarding Passes Change Seats Check Luggage Purchase an Upgrade
Online bookstore	Update Customer Profile Search for an Item Buy an Item Track a Shipped Package Cancel an Unshipped Order

Solution requirements

- Anforderungen rund um die Lösung, damit das gestellte Problem gelöst werden kann
- Wir reden hier aber auch noch immer nicht von technischen Lösungen!
- Details zu „Check in for a flight“ enthält noch **keine technischen Details**

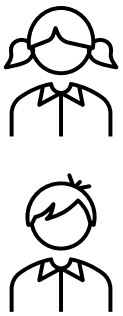
Technology requirements

- **Nachdem** Solution Requirements verstanden wurden, kann man sich Gedanken machen, wie man es am besten Umsetzen kann
- Mit den nicht-funktionalen Anforderungen können wir jetzt eine **Architektur entwerfen**
- Siehe auch: Gesetz von Conway
  - “Organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations.”

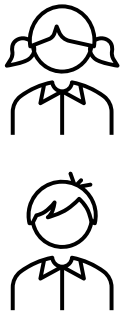
Tipp: Rechtzeitig mit dem Requirements-Management des Projekts auseinander setzen – jeder hat da seine eigene Technik. Hintergrund: Auf was muss ich achten?

Gesetz von Conway

Team 1



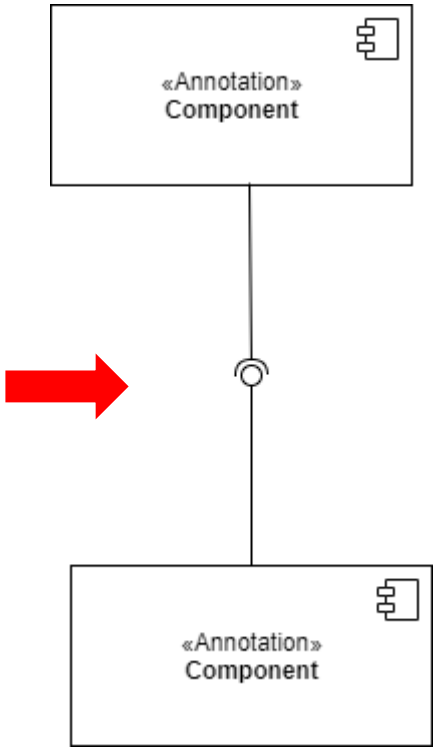
Team 2



Struktur

&

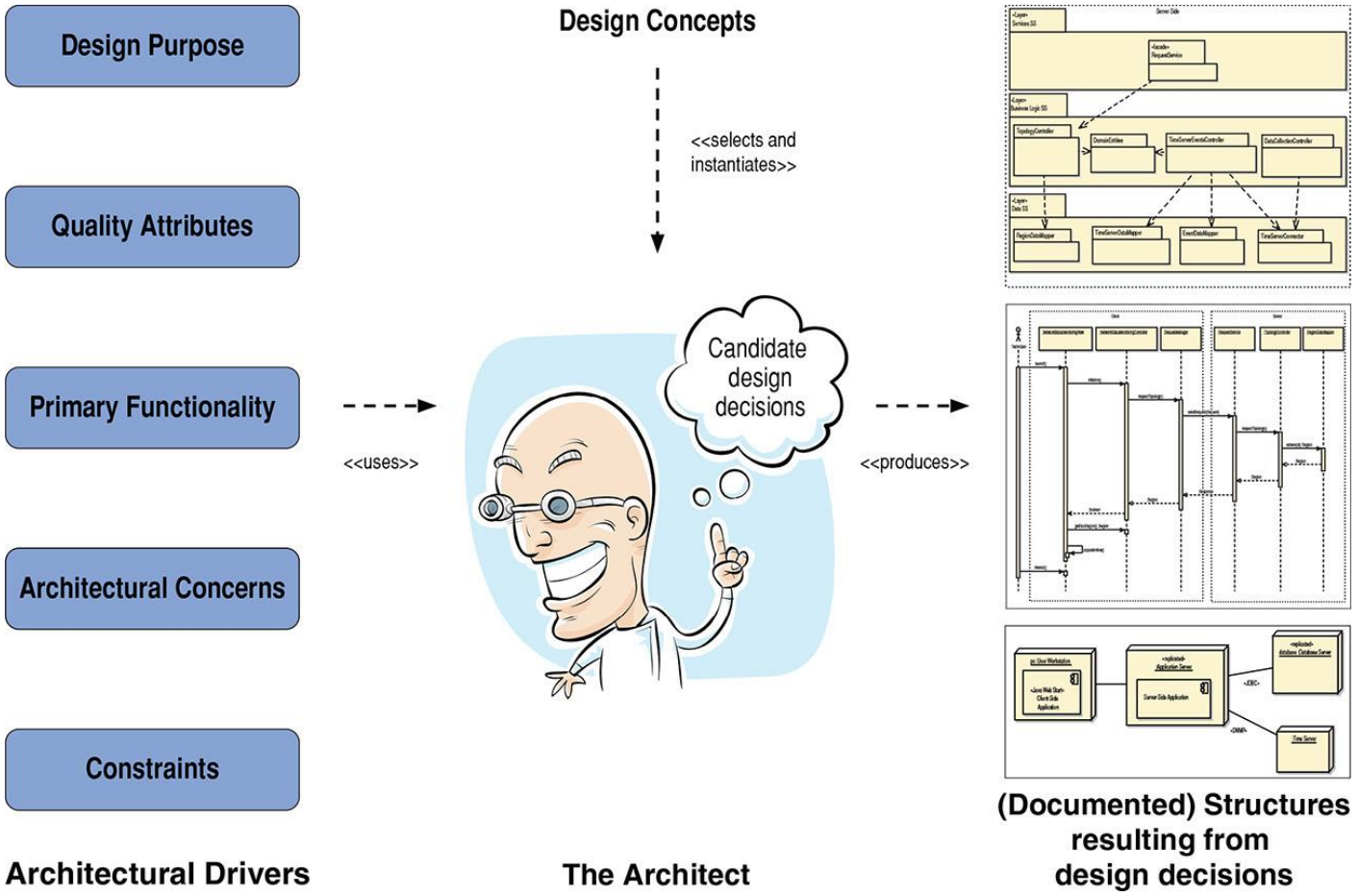
Qualität und Art der Schnittstelle = Qualität und Art der zwischenmenschlichen Kommunikation



Anmerkung: Es muss nicht unbedingt eine Komponente sein – es kann auch ein ganzes System o.ä. sein

Neben Nicht-Funktionalen Anforderungen (auch genannt Non-Functional Requirements oder Quality Attributes) – welche Treiber gibt es noch?

Treiber der Softwarearchitektur



Design Purpose

- Warum und wann macht man das Architektur-Design?
- Architektur Design kann viele Gründe haben:
  - Für einen Projektantrag
  - Für die Entwicklung eines Prototypen
  - Für die Umsetzung / Entwicklung eines Projekts
- Was sind die Business-Goals, über die sich das Unternehmen am meisten sorgt?
- Halten Sie den Grund der Planung in der Dokumentation fest
  - Vermeidet falsche Erwartungen des Lesers (e.g. Prototyp Architektur wird blind umgesetzt)

Quality Attributes

- Sind mess- und testbare Eigenschaften des Systems
- Oft auch als nichtfunktionale Anforderungen bekannt
- Sind jene Driver mit dem größten Einfluss auf die Architektur
- Beispiele:
  - Performance
  - Modifiability
  - Availability
  - Security
  - uvm.

Primary Functionality

- Man muss die Funktionalität des Systems kennen, um diese Elementen (Software-Modulen → Komponenten) der Architektur zuordnen zu können (Modifiability / Reusability)
- Unabhängig von Quality-Attributes
  - Ob ich alles in eine Komponente stecke oder verteilt in viele Komponenten stecke, macht für die Funktionalität keinen Unterschied
- Zum Tragen kommt die Zuordnung, wenn man etwas ändern will (**Refactoring ***):
  - Ist die Funktionalität nur in 1 Komponenten lokalisiert → einfach

* Refactoring: Funktionalität bleibt gleich, Quality-Attributes ändern sich im Regelfall

Architectural Concerns

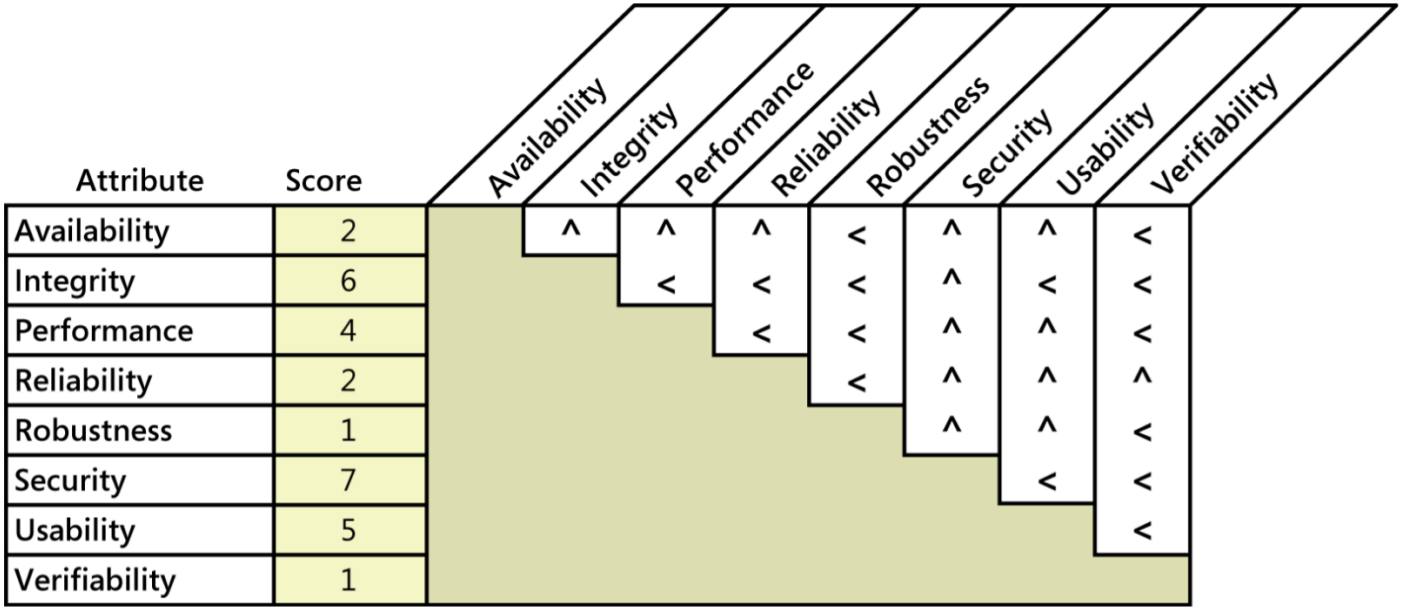
- Sachen, die beim Design berücksichtigt werden sollen, aber nicht als klassische Requirements ausgedrückt werden können
- Beispiele:
  - Dass das initiale Entwickler-Server-System rechtzeitig zur Verfügung steht
  - Das Wissen im Team über Entity-Framework und SQL Server ist nicht vorhanden
  - Das Entwickler-Team ist über 3 Kontinente verteilt

Constraints

- Beispiele:
  - Gesetze / Normen die einzuhalten sind
  - Rückwärtskompatibilität (e.g. weil der Kunde seine Landschaft zeitnah nicht updaten kann)
  - Verfügbare Technologien (e.g. vom Kunden vorgegeben, da schon Lizenzen oder Know-How verfügbar)
- Zum Vergleich:
  - Ein Quality-Attribute (Non-Functional-Requirement) ist immer eine Quantifizierung eines Functional-Requirement (Wie lange darf es dauern, wenn ich auf Speichern klicke?)
  - Ein Constraint ist ein in Stein gemeißelte, bereits getroffene Entscheidung welche „projektweit“ gilt

Nicht-funktionale Anforderungen finden

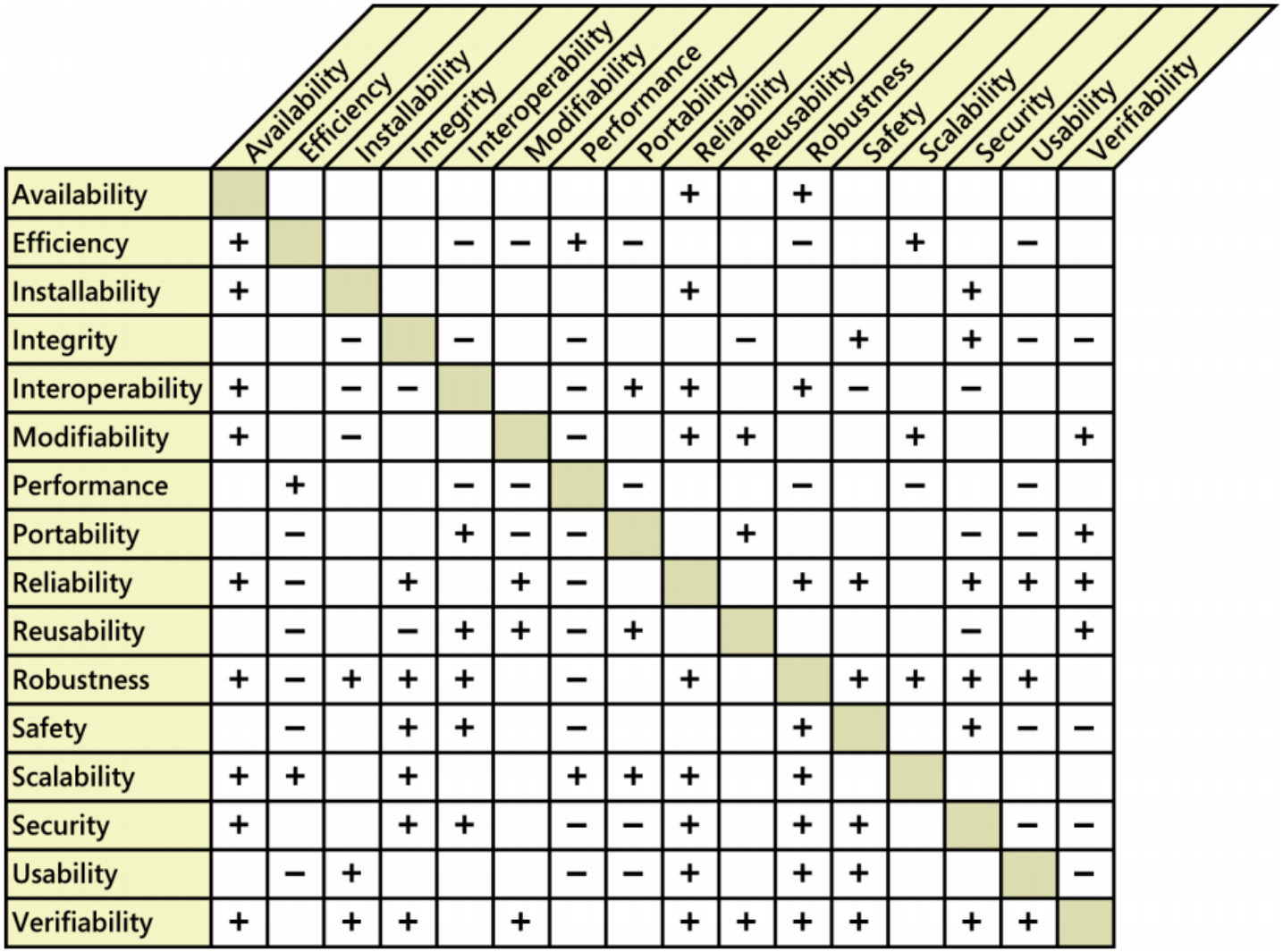
- Funktionalität ist nicht alles …
- Es geht auch um „wie gut“ ist etwas
- Template hilft zu priorisieren:



Attribute	Score	Availability	Integrity	Performance	Reliability	Robustness	Security	Usability	Verifiability
Availability	2	^	^	^	<	^	^	<	
Integrity	6		<	<	<	^	<	<	
Performance	4			<	<	^	^	<	
Reliability	2				<	^	^	^	
Robustness	1					^	^	<	
Security	7						<	<	
Usability	5							<	
Verifiability	1								

Wiegers, K., & Beatty, J. (2013). Software requirements. Pearson Education.

Wechselwirkungen beachten (Wiegers)



	Availability	Efficiency	Installability	Integrity	Interoperability	Modifiability	Performance	Portability	Reliability	Reusability	Robustness	Safety	Scalability	Security	Usability	Verifiability
Availability									+		+					
Efficiency	+				−	−	+	−			−		+		−	
Installability	+								+					+		
Integrity			−		−		−			−		+		+	−	−
Interoperability	+		−	−			−	+	+		+	−		−		
Modifiability	+		−				−		+	+			+			+
Performance		+			−	−		−			−		−		−	
Portability		−			+	−	−			+				−	−	+
Reliability	+	−		+		+	−				+	+		+	+	+
Reusability		−		−	+	+	−	+						−		+
Robustness	+	−	+	+	+		−		+			+	+	+	+	
Safety		−		+	+		−				+			+	−	−
Scalability	+	+		+			+	+	+		+					
Security	+			+	+		−	−	+		+	+			−	−
Usability		−	+				−	−	+		+	+				−
Verifiability	+		+	+		+			+	+	+	+		+	+	

-: Zeile hat negative Auswirkung auf Spalte, +: Zeile hat negative Auswirkung auf Spalte

Beispiel Performance

- Stakeholder können die Frage nach Performance Requirements schwer beantworten
- Daher konkrete Fragen:
  - Welche Zeit für eine Suchanfrage ist akzeptabel?
  - Gibt es Tage / Monate an denen der Shop öfters besucht wird?
  - Wie viele Benutzer sind im Durchschnitt gleichzeitig im Shop?
  - …

Was sollten Sie aus dem Kapitel mitnehmen?

- Eine jede Software hat **immer** eine Architektur – so wie jedes Haus eine Architektur hat. Sie kann gut oder schlecht sein
- Es gibt 5 essentielle Treiber / Driver, welche die Architektur beeinflussen
- Der Architekt muss diese erkennen und analysieren und anschließend aus (im Regelfall) mehreren Kandidat-Architekturen, eine Lösung auswählen
  - Nicht Fokus der Vorlesung → daher keine gesonderte Übung dazu

Tool Support: Jama

- Nicht Fokus der Vorlesung – sollte man aber mal gesehen haben