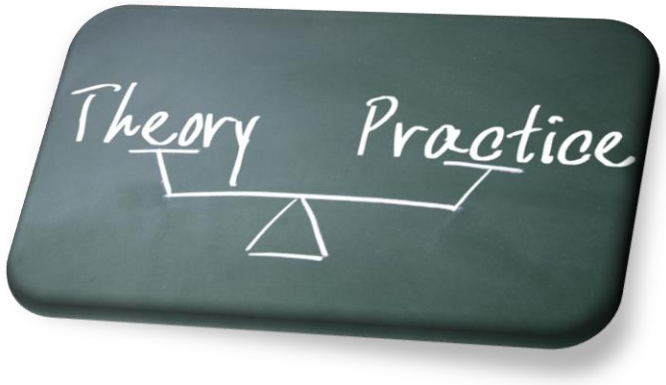
How would an academic explain software architectures to you?

Prepared by Michael Vodep

BearingPoint®

Why is the academic viewpoint interesting?

- Academics are very precise with definitions and delimitations
- … That helps to develop a common language which also improves the common understanding
- It also sharpens the mindset to think not only in terms of technologies but also in terms of **concepts**
  - E.g. Kafka as a technology versus “How can a broker architecture style can help me?”

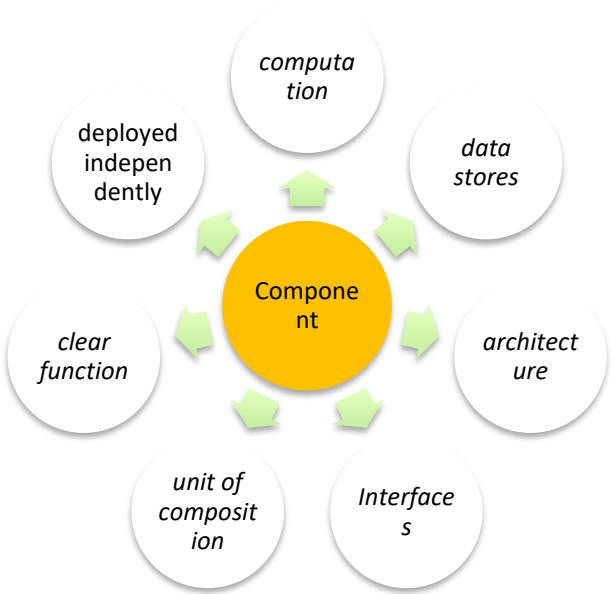


BearingPoint

What is a component?

Definitions from the literature

- "*the principal computation elements and data stores that execute in a system*" (Clements et al.)
- *"A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces."* (PhilippeKrutchen, RationalSoftware)
- *"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition."* (Clemens Szyperski, ComponentSoftware)
- …



BearingPoint

Example of clear function:

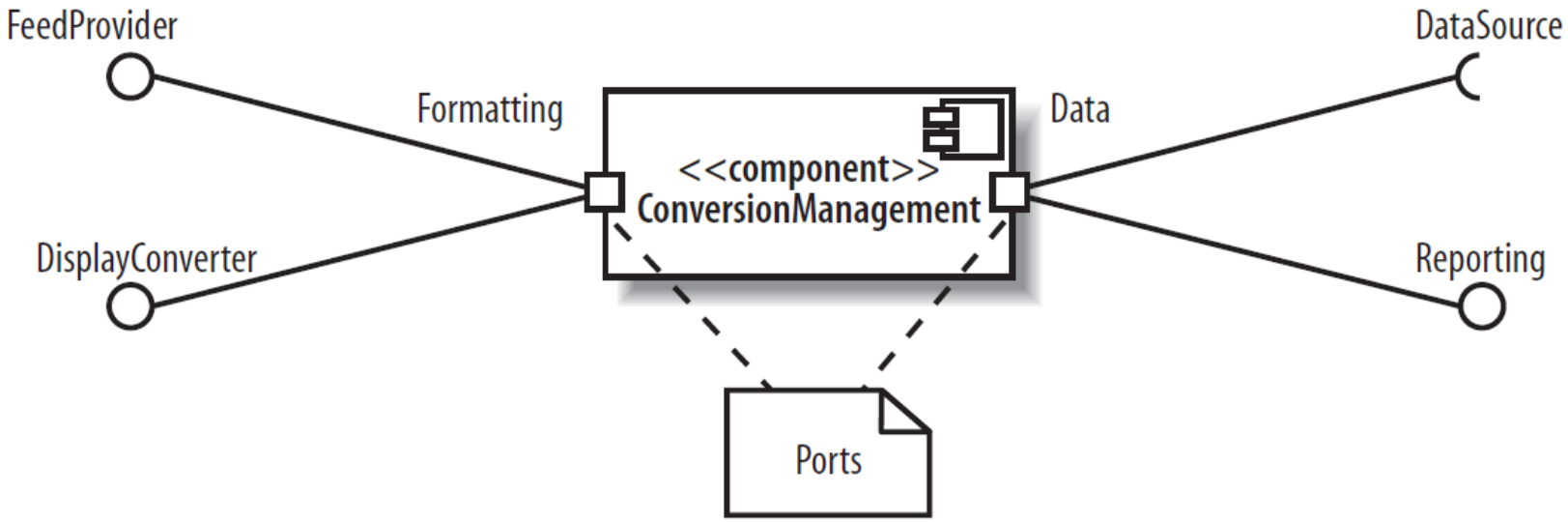
Modular system of an Airbus A-380

BearingPoint®

UML 2.0

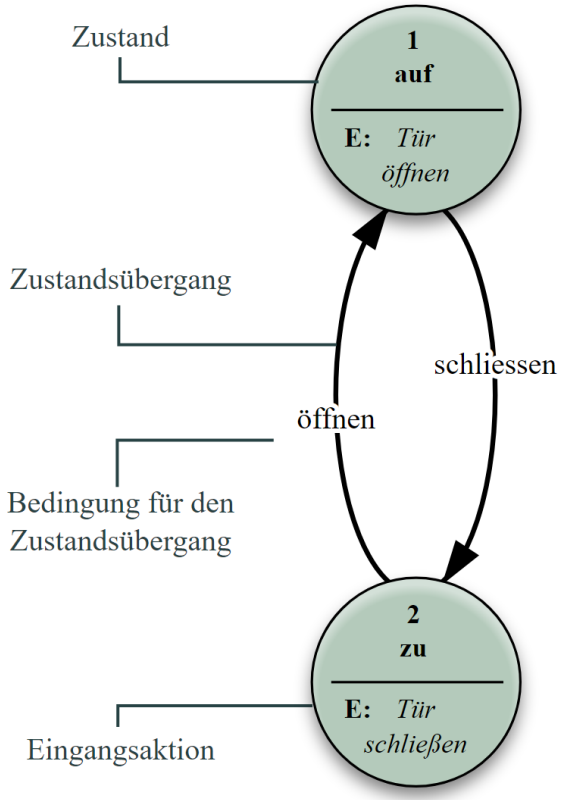
How components look like in UML 2.0

- **Interfaces**: Circle = provided, half circle = required



Port

- In UML 2.0 it groups interfaces
- In academic literature:
  - All public methods and events of a component are available via ports ("interface").
  - Addition: Call of methods only allowed in certain order (e.g. by Finite-State-Machine (FSM))

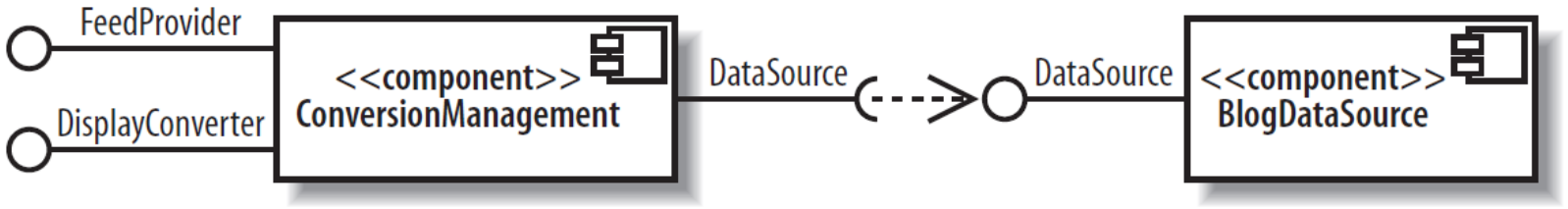


BearingPoint

To solve a problem ...

… divide it into subproblems (multiple components)

- The Ports / Interfaces of components are connected by **Connectors** (dashed line)
  - Connectors are the primary driver of the architecture style (local-method call, RPC, SQL, SOAP over HTTP, ...)
  - Can also perform other tasks: Encryption, Filtering, Transforming, Mediate between multiple sub-components ...
- **Non-Functional requirements** are very important for Connectors
  - impact on Interface (e.g. adapt to one call instead of several), deployment (e.g. remote vs. same physical host vs. same memory space), ...



BearingPoint

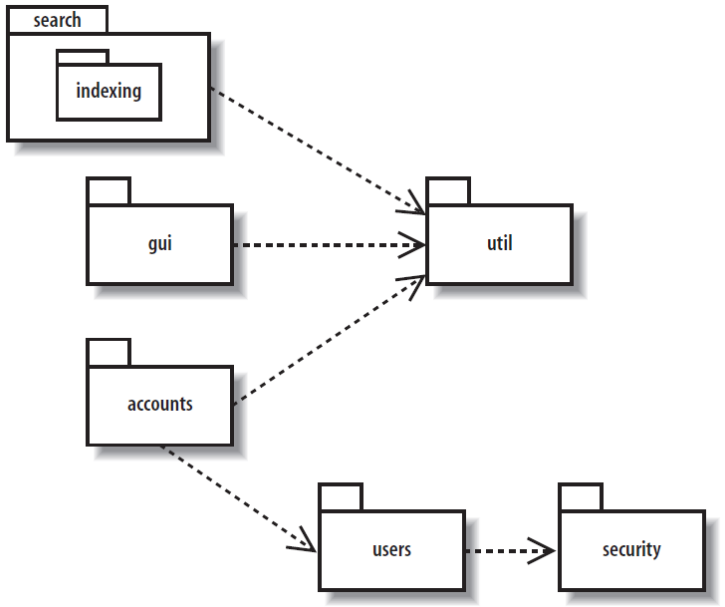
How can components communicate?

- Remote-Procedcure-Call / Local-Procedure-Call / IPC, …
- REST (it's an architecture style → think of the constraints):
  - It forces you for: client-server, stateless, think in terms of ressources
- Broker Style: decoupling
  - Technology examples: RabbitMQ, Kafka, …
- Brokerless: Messaging patterns without a broker
  - E.g. zeroMQ
- Shared Database: out of the box transaction support
- Shared Files / BLOBs: Easy to understand and handle, decoupling
- In-Memory-Grid
- …

What's the difference to modules?

Varies depending on the literature - we will therefore leave it at: Module - Design-Time vs. Component - Runtime

- modules: Artifacts (classes, interfaces) that are grouped together.
  - Exist only at design time
- Module structures exist in the file system (*.cs files in a class library)
- Example of UML 2.0 Package diagram



What is software architecture?

There is not “The” definition

- Lots of definitions out there …
- There are interesting quotes:
  - *“The software architecture of a system is the set of structures **needed to reason** about the system, which comprise software elements, relations among them, and properties of both.” (Software Architecture in Practise)*
  - *“The question, “Which U.S. cities can you travel to by boat?” **does not require a complete model** of the country”*
  - *“All models are wrong, but some are useful“ (George Box)*
- Summary:
  - A software architecture uses components, connectors as their primary tool
  - Software architecture can be described by models → which are always an abstraction → detailed enough, to see, if our non-functional requirements are met
  - Zoom-In and Zoom-Out → Depending on your question (for example https://c4model.com for a logical / deployment view)
  - Beside zooming you can also change the perspective: there are different views (“models”):
    - See https://en.wikipedia.org/wiki/4%2B1_architectural_view_model
    - See https://www.oreilly.com/library/view/documenting-software-architectures/9780132488617/

alpha

Content may be wrong

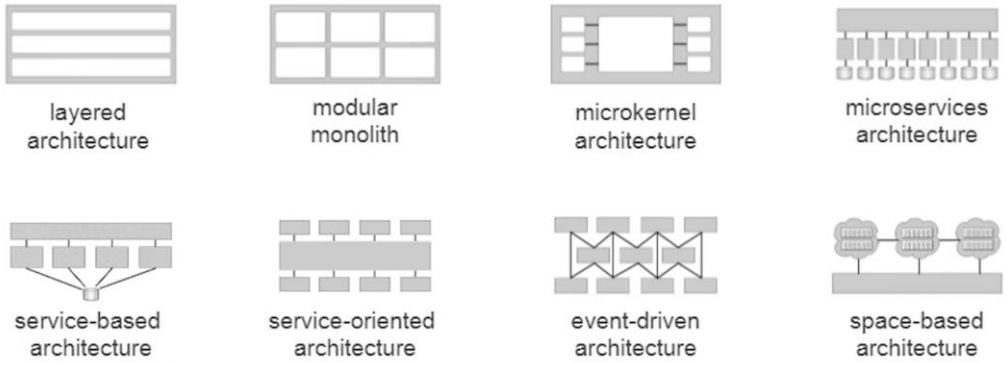
Freestyle: Some selected topics

BearingPoint®

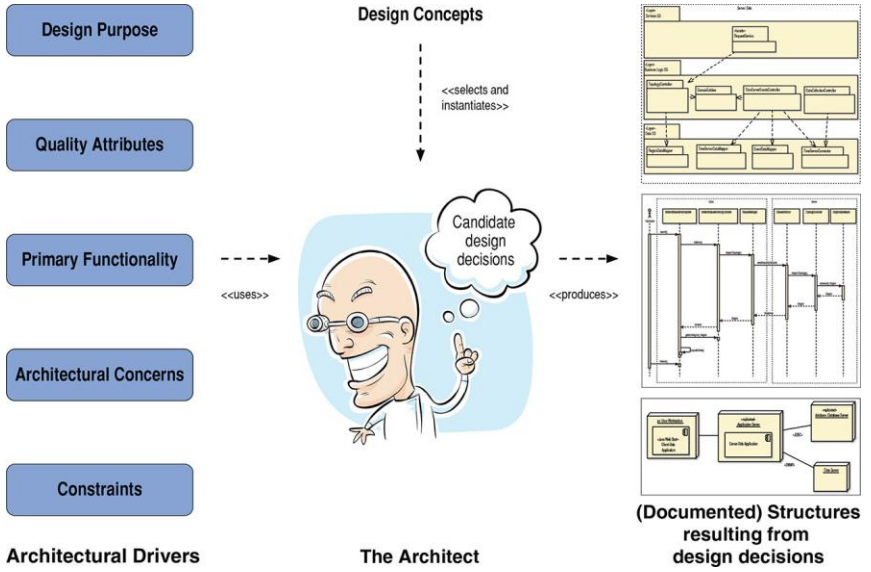
What's the difference to Microservices?

- None!
- Microservice also try to make problem decomposition
  - clear function for each microservice
  - Focus on Interfaces
- But there are lot of additional best practices: https://microservices.io/i/MicroservicePatternLanguage.pdf
  - Service per team
  - No Shared database
  - …
- Personal opinion (after read lots of discussions):
  - Components can also talk via Local-Method call or IPC
  - Microservices can consist of multiple components
  - That we face the challenges of distributed systems in many projects - unnecessarily - and blame microservices, I find wrong!

Microservices as I saw it some times ago



- I thought: “I have no Microservices – I have done something wrong on my architecture”
- Mostly: In some parts of the project there will be just one dominant architecture style → you can’t do client / server and peer-to-peer at once
  - A architecture style gives you guidance how to divide / organize your components
- Examples of styles: Layered, Modular Monolith, Microkernel, Even-drive, Microservices, Space-Based architecture, Service-Based Architecture, ...
  - But also: Peer-to-Peer, Client-Server, Pipes-And-Filters, …
- The thing is:
  - If google uses one of them – are all others bad?
  - If linkedIn uses a specific technology / communication style (e.g. Kafka) – are all other bad?
- There are lots of things which are influence an architecture



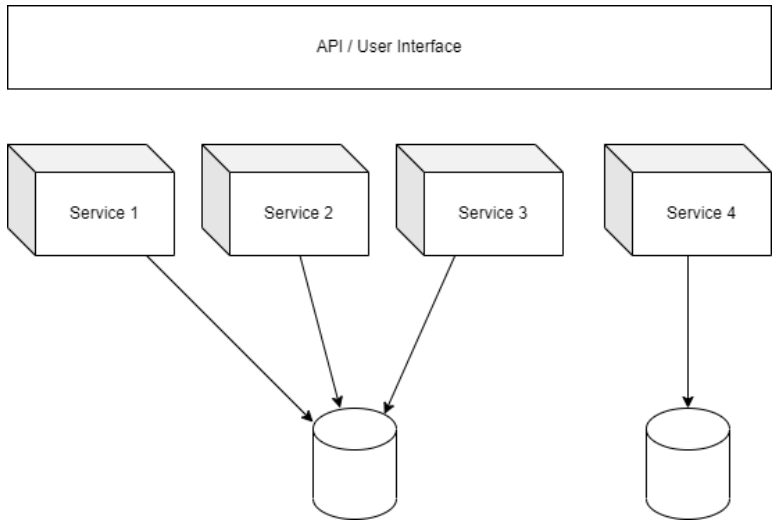
BearingPoint

Why microservices and why Independent deployability?

- Examples of Microservices: Payment Service, Notification Service (SMS, WhatsApp, …), AWS S3 API (as an API example)
- What do they have in common? Very, Very stable interfaces.
  - You don’t really need to re-run 10 times a day your integration tests, if a payment service does 10 times a day a deployment
    - Assuming: They do their service tests correctly
- Think twice if you use your Microservices for problem decomposition
  - Give each team its own repository, need tons if integrations tests between microservices to see “Service A 1.3, Service B 3.23, Service C 0.12” works as expected → not the idea of independent deployability → its just a distributed monolith

Hybrids

- not everything is black / white …
- Separation by domains – independent deployed
  - Services may also have separated database – but not all of them (advantage: transactions)
  - In other words: not all services need to be microservices

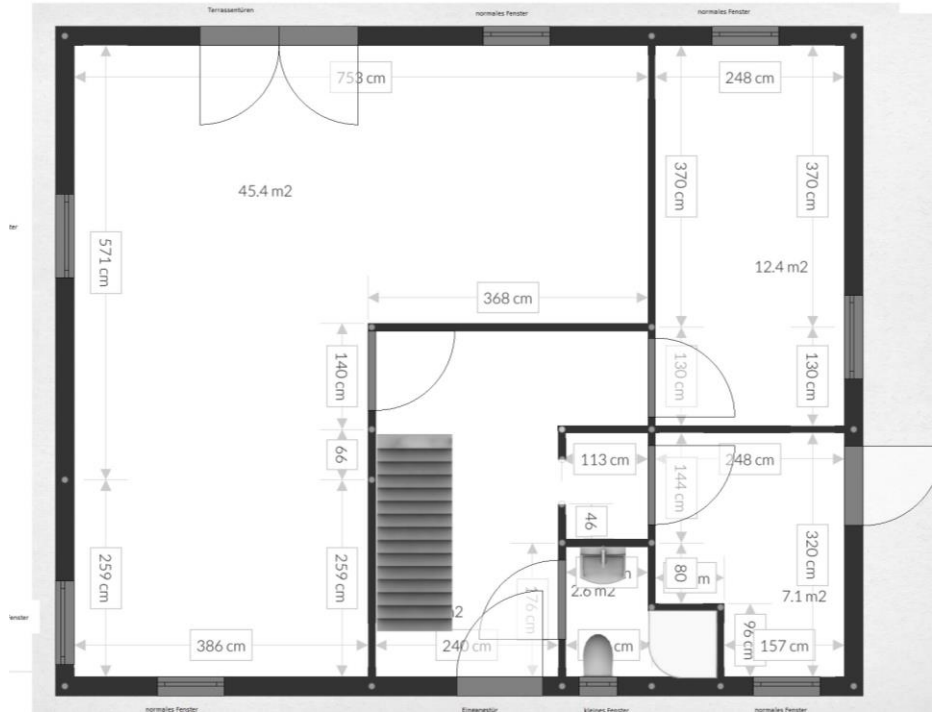


How much planning should be done?

Risk is your friend

- Would you let bricklayers (analog: software developers) start to build a residential building without planning the architecture (analog: programming)?
- Do you model every detail in a building plan? Where a mirror hangs or the wall color? No! Only those things that are necessary to analyze whether the drivers of the architecture have been achieved (remember: all models are abstractions)

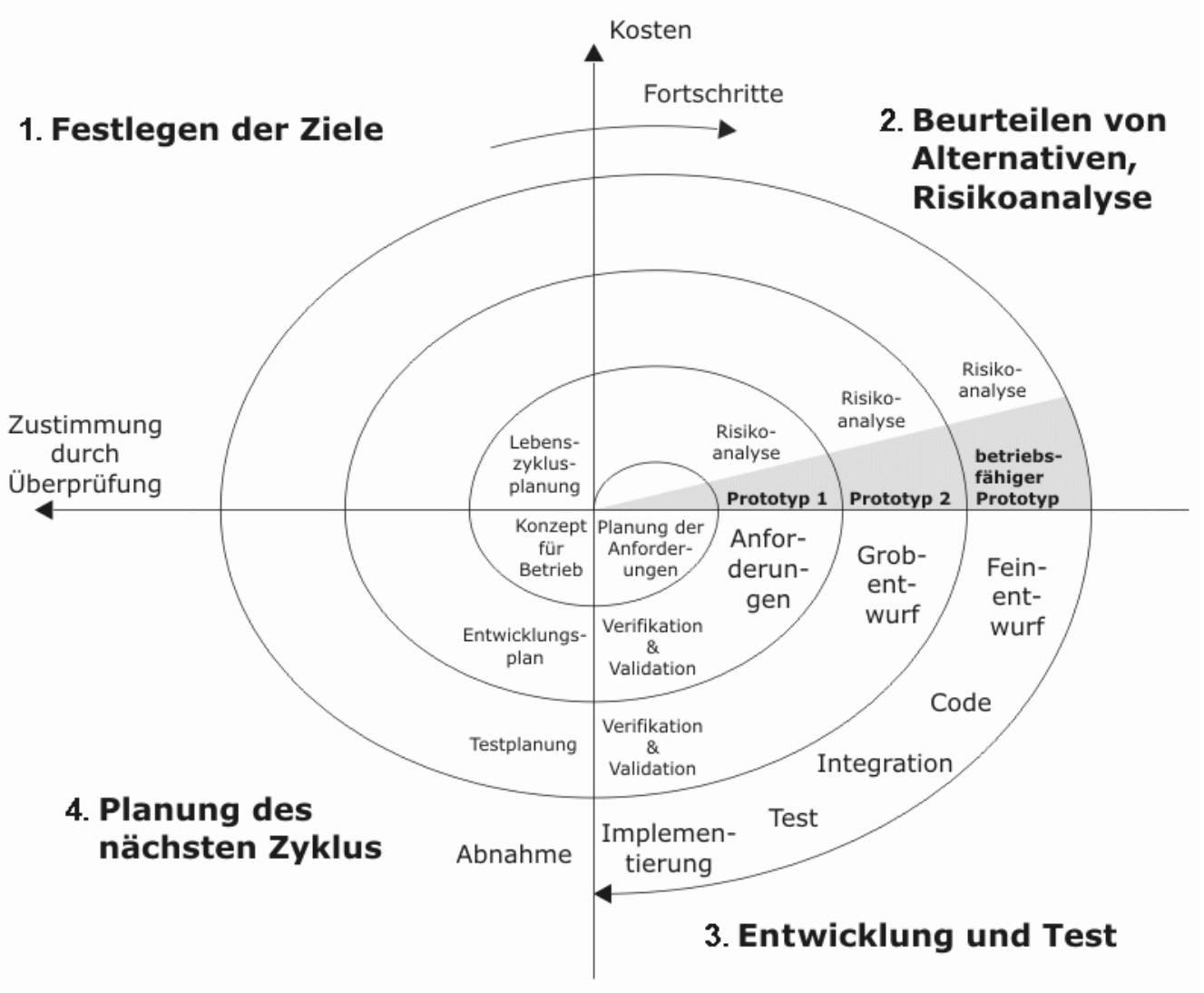
Example: 2 projects with different risk



BearingPoint


Front is part of the architecture

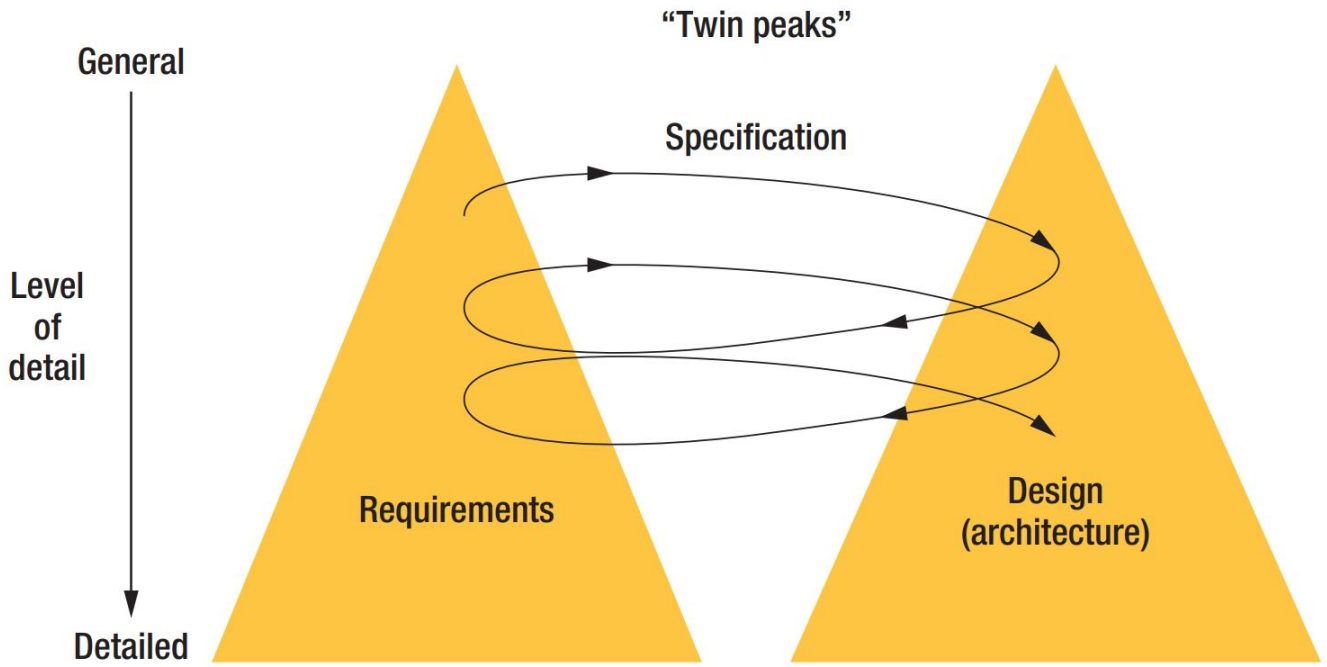
Böhms Spiral model



BearingPoint

Twin peaks

- If somebody comes which an architecture, without knowing the requirements → be sceptical



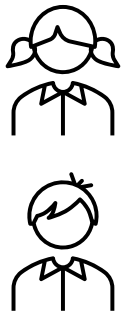
BearingPoint

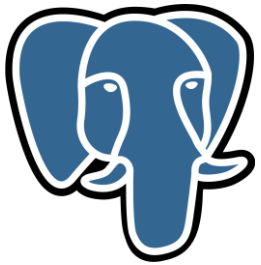
Architecture in the planning phase

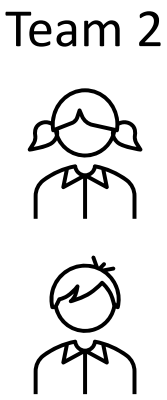
- Look for Presumptive architectures: „A presumptive architecture is a family of architectures that is dominant in a particular domain.”
  - Architecture for storing personal data, Architecture for IoT, Architecture for Trading, Architecture for self-driving cars, ...
- If not found: prototyping necessary?
- Technical Feasibility will be answered in the planning phase
- Try to get the amount of risk:
  - There are few alternative solutions
  - High risk of failure (e.g. medical, avionic, …)
  - Demanding quality attributes / Non-functional requirements
  - New domain for your team
  - …

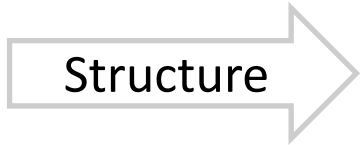
Conway's law

Can be found in most large companies



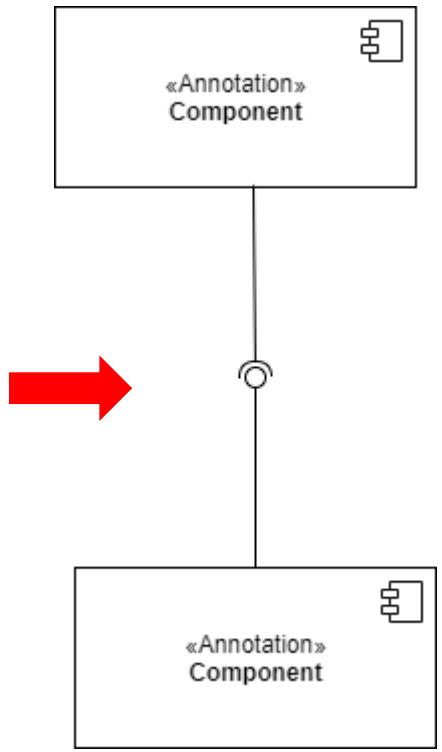






&

Quality and type of interface = quality and type of interpersonal communication

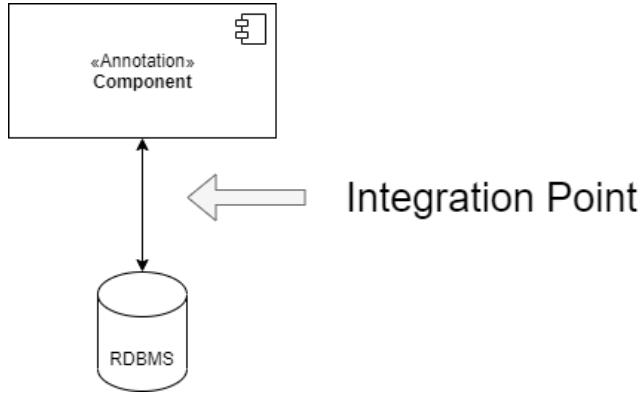


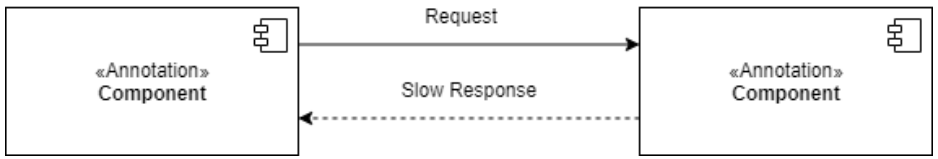
Anmerkung: Es muss nicht unbedingt eine Komponente sein – es kann auch ein ganzes System o.ä. sein

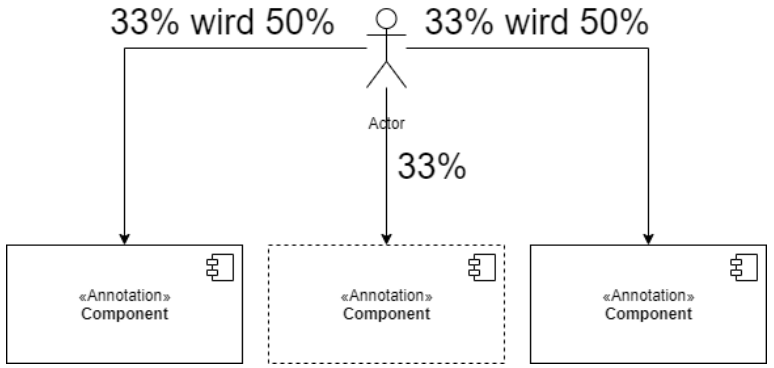
Components in daily life

Watch out for …

- Integration points: Errors from network error to semantic error
  - Patterns: Circuit Breaker, Timeouts, Decoupling Middleware, Handshaking
- Chain-Reactions: 3 components - 33% load each. 1 server fails - other components should take over work
- Slow-Responses
- From my kitchen: if we put everyhing together – how to design an health endpoint?
  - https://gist.github.com/mvodep/cd8b0c31676dea72504352f9ec3ad9f8#file-20220314-health-json







BearingPoint

BearingPoint®