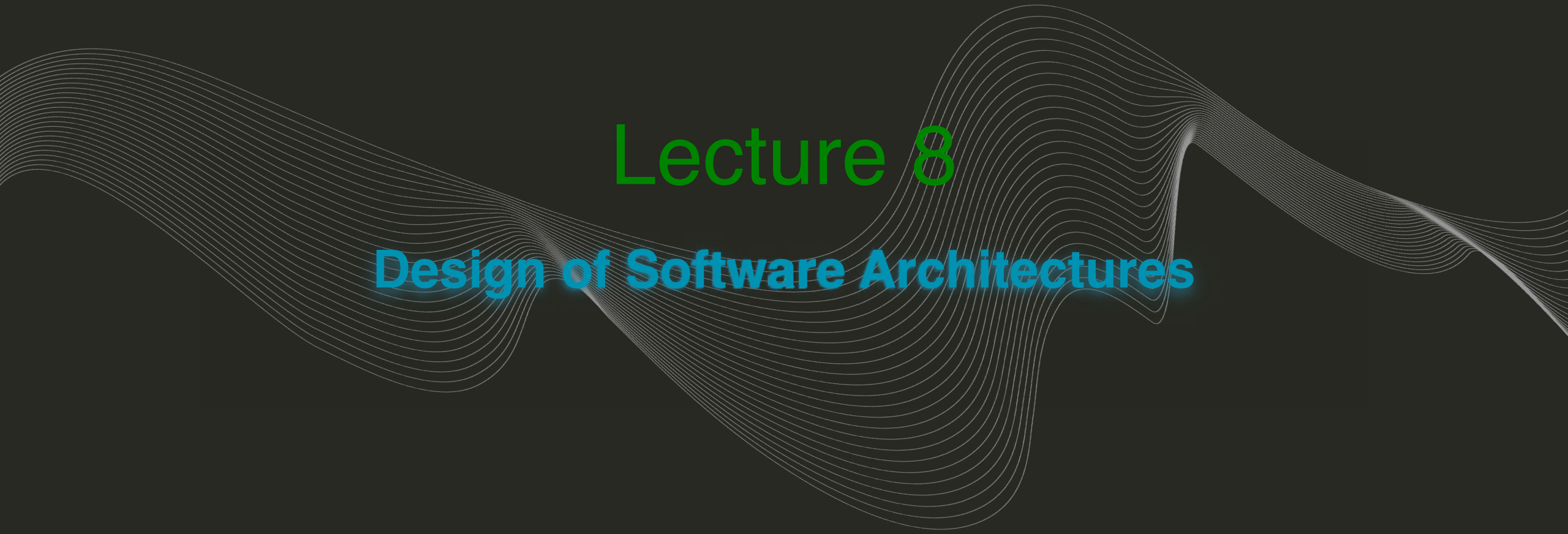


Lecture 8

Design of Software Architectures

Dr. Georgiana Caltais, Dr. Vadim Zaytsev

UNIVERSITY OF TWENTE.

Lecture 8: Overview

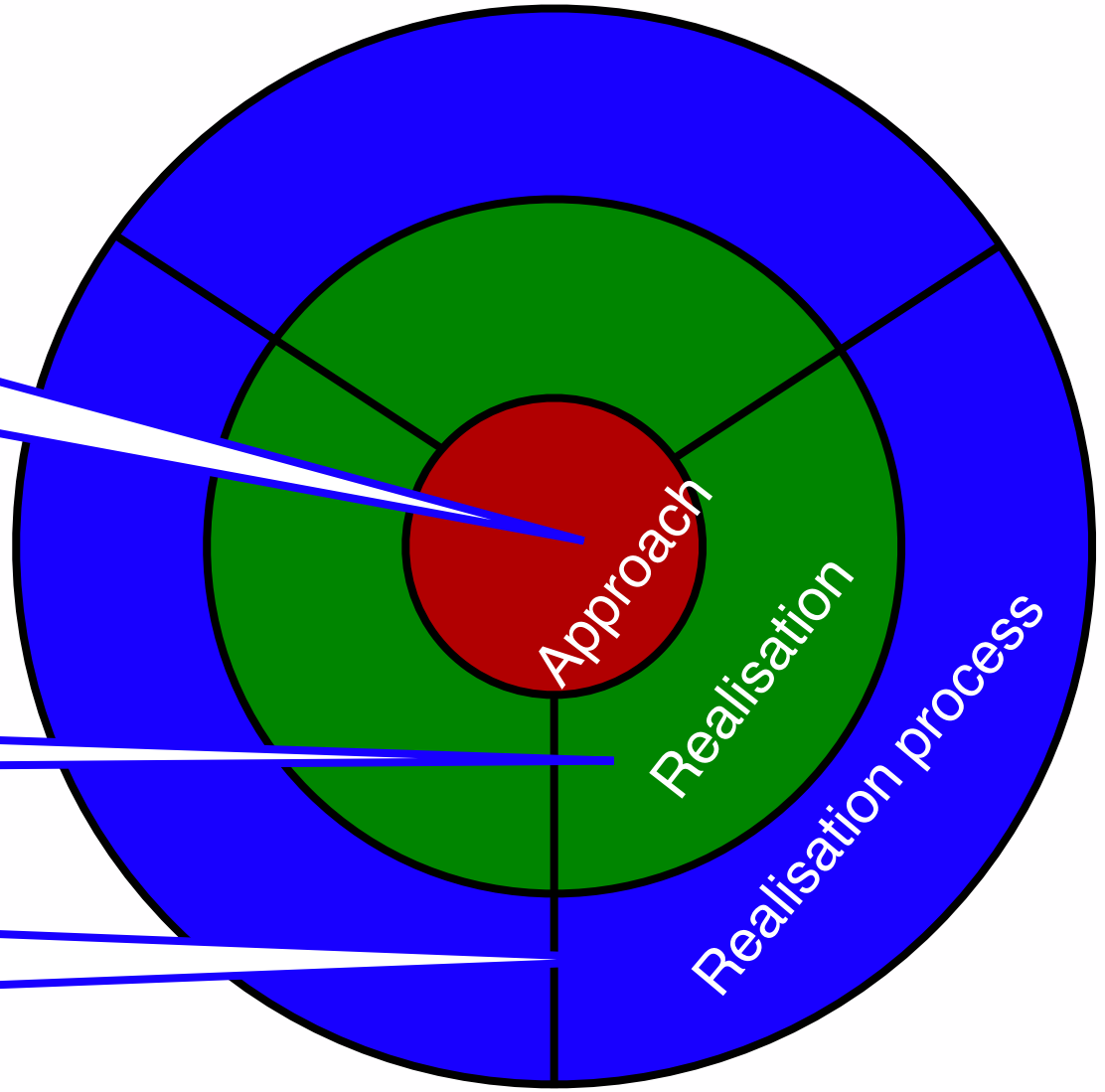
- Recap
- Family Architectures
- Quality Attributes
- QA Scenarios

Decisions - Realisation domain

By which activities & what behaviour?
- main tasks
- phases
- milestones

Who does what when?
- planning, roadmap,
- responsibilities,
- work packages

Which organisation/people?
- team structure,
- knowledge/skills,
- education/training



by Robert Deckers, based on ISO 42010 & DYA

Dominant decomposition

- Choose one style — dominant
  - explain how it fits your domain!
- Choose another style — secondary
  - how is it related?
- How does it map
  - onto the structure of the dev org?
  - onto the tasks of team members?
- Which major stakeholder concerns:
  - will be served by your decomposition? (and why)
  - will be difficult to address well? (and why)

Patterns - general solution to general problem

- **Design**
  - solutions to specific problems encountered during software development
  - guidelines for object relationships, responsibilities, and behaviours
  - Singleton, Factory, Observer, Decorator, Strategy, etc.
- **Architectural**
  - abstract solutions to problems related to the overall structure of a software system
  - address concerns such as system scalability, performance, maintainability, and security
  - Layered, Event-Driven, Microservices, Event-Bus, Pipe-Filter, Blackboard, MVC
  - Relation to: arch styles / decomposition?

Lecture 8: Overview

- Recap
- Family Architectures
- Quality Attributes
- QA Scenarios

What is a system family?

- Collection of systems
  - with common properties

- Collection of “genes”
  - to create systems
  - with the same properties

(give examples)

System family - genes?

https://www.brouwerijeanske.nl/onze-bieren/

E-commerce Software Family - genes?

- Shopping Cart Gene
- User Authentication Gene
- Product Catalog Gene
- Payment Processing Gene
- Order Management Gene

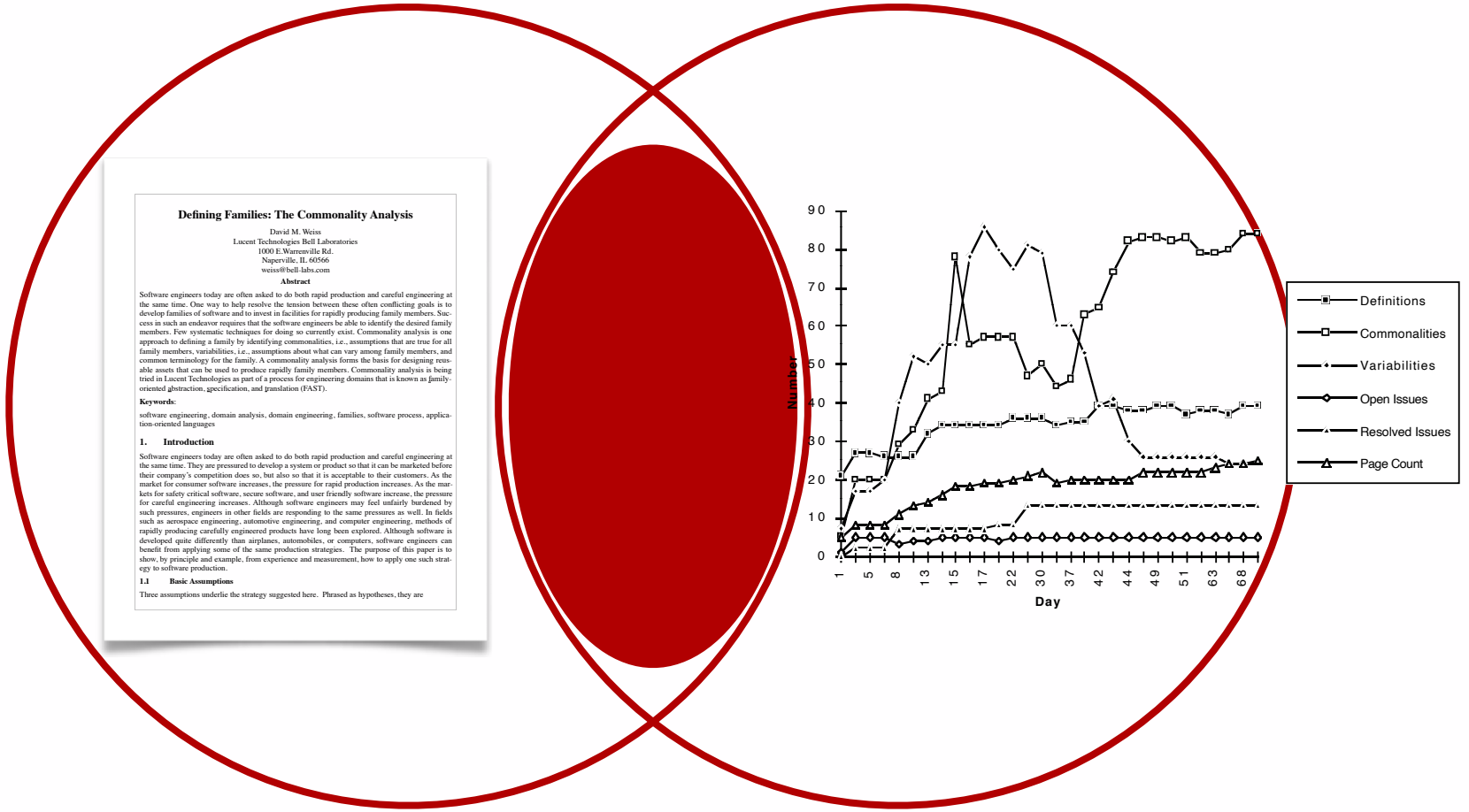
Advantages of a family approach

- One solution per development problem
  - lower development costs per system
  - lower development time per system
- Homogeneous systems
  - lower management/maintenance costs
  - predictability for quality/behaviour/…
- Solution validation in multiple systems
  - faster insight if solution works
  - bugs detected earlier

Set up a family

- Scope/goal: identify possible members
- Establish common architecture
- Describe realisation of reusable elements
  - common database, ESB/DAL, GUI, DPs, libraries, API,…
- Plan realisation of a new member
- Ensure family reqs for all members
  - security, performance, availability
- In practice: add reference implementations

Commonalities & variabilities

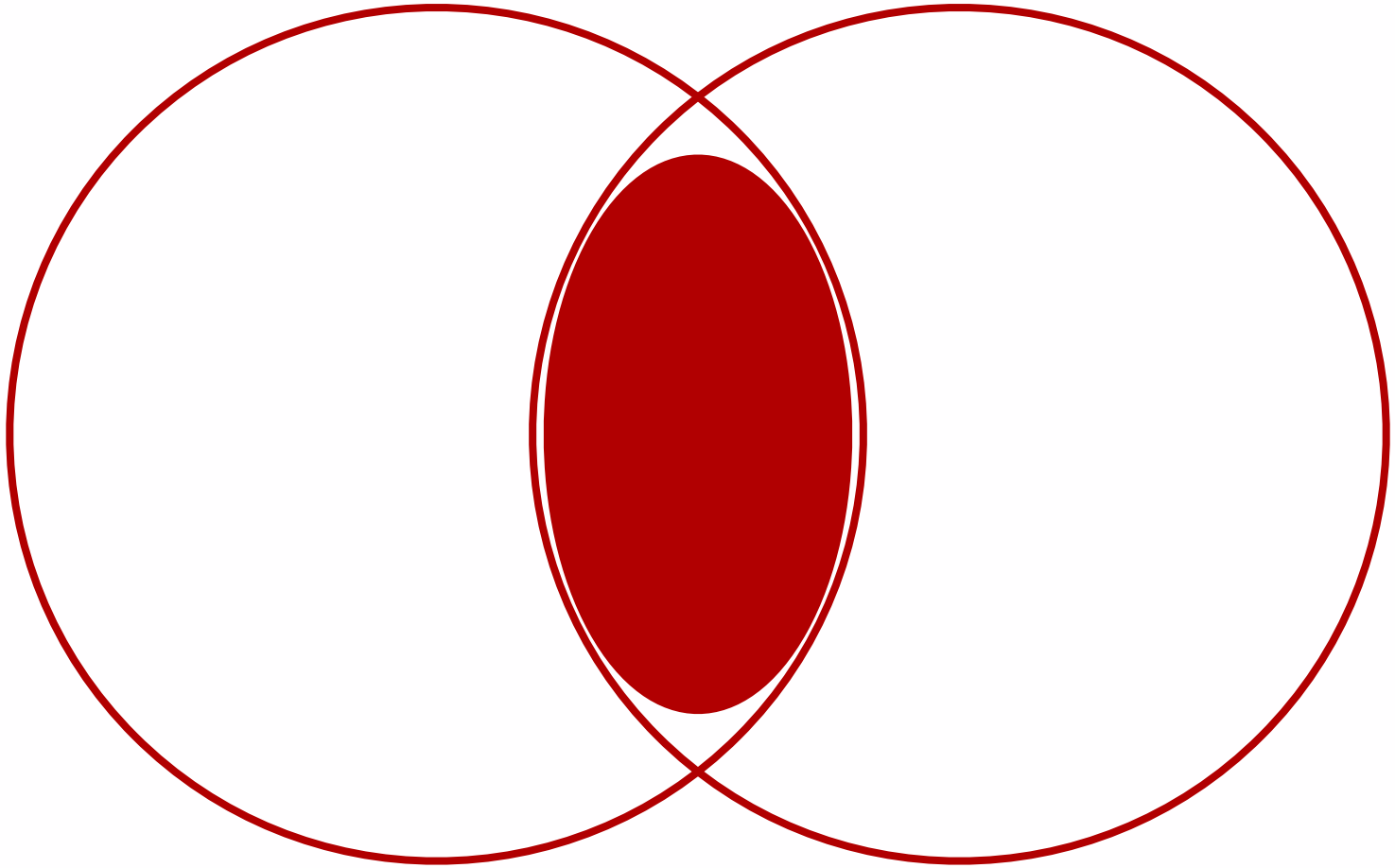


David M. Weiss, *Defining Families: The Commonality Analysis*, 1997?

Achieve commonality

- Fix functional reqs
  - later: in house, outsource, buy, …
- Prescribe design elements:
  - design patterns
  - software components
  - platform
- Standardise dev't process/organisation
  - test approach, coding standards, guidelines, …
  - deliverables and templates with consistency rules
  - roles and knowledge of people

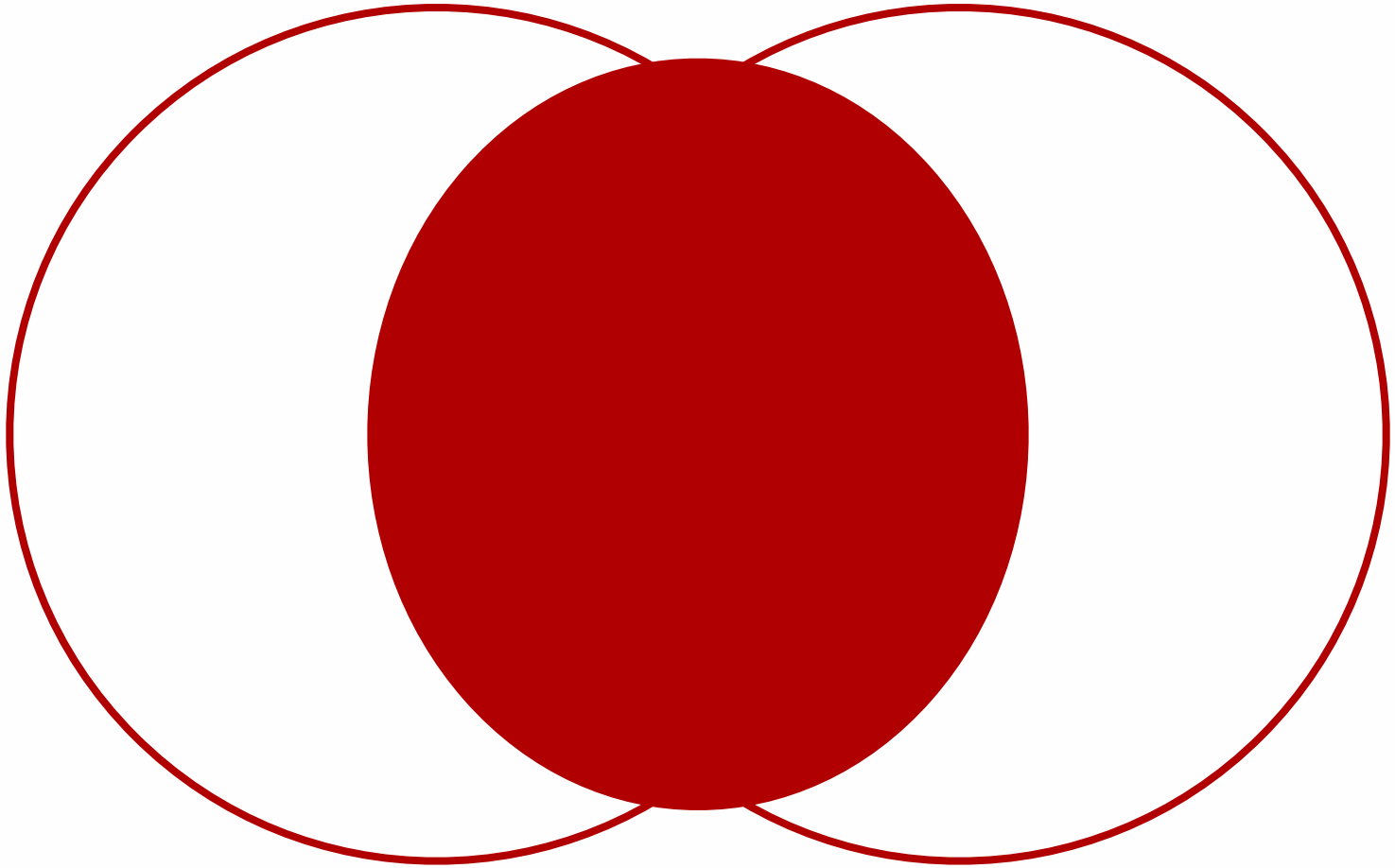
Too little in common



Achieve variability

- Configurability
- Software components
- Plugin infrastructure
- Instantiation
- Inheritance / specialisation
- . . .

Too much in common

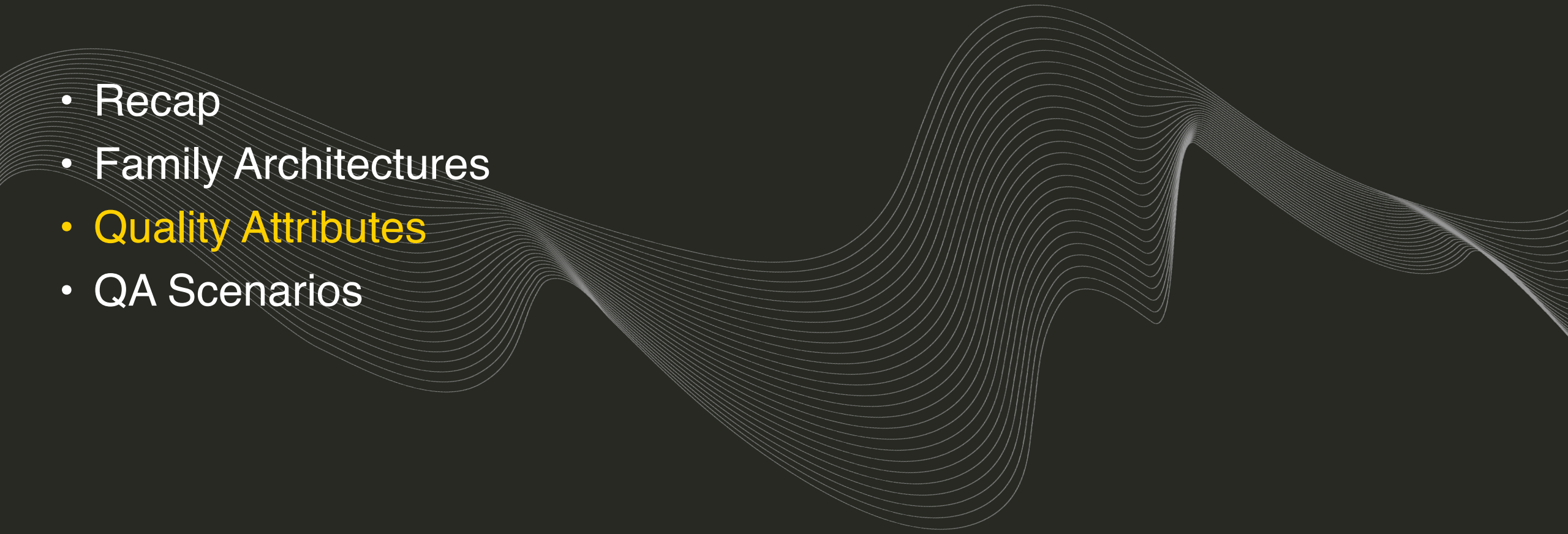


H

Apply the family approach

- Identify families in the context of your system
  - could be the system or components
- Think about:
  - commonalities
  - variabilities
- Describe:
  - family name/definition
  - what's common/variable
  - mechanisms to achieve c/v
  - show an example

Lecture 8: Overview



- Recap
- Family Architectures
- Quality Attributes
- QA Scenarios

stakeholder statement

vs

system requirement

vs

quality attribute

What could go wrong?

- The product must be user friendly and easy to use
- Response time should be less than 1 second
- The system should be always available
- It shall work just like the previous system, but on a new platform
- The system must never fail, and it if does, should produce a report
- The product should be accessible
- The communication protocol cannot change
- The system has to be robust
- You must support many parallel users
- The product must fit all users' individual needs

ReQtest, *10 of the Worst Requirements I have Ever Seen*, 2016

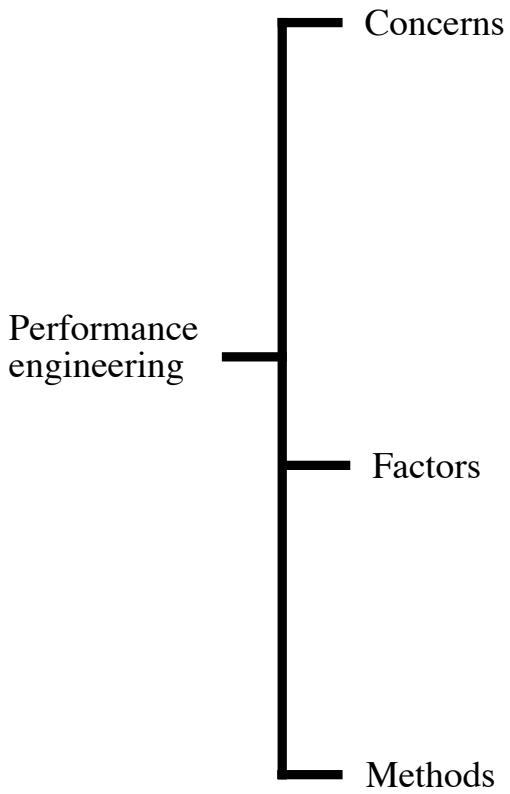
Everyone wants quality

Quality is subjective

Quality attributes

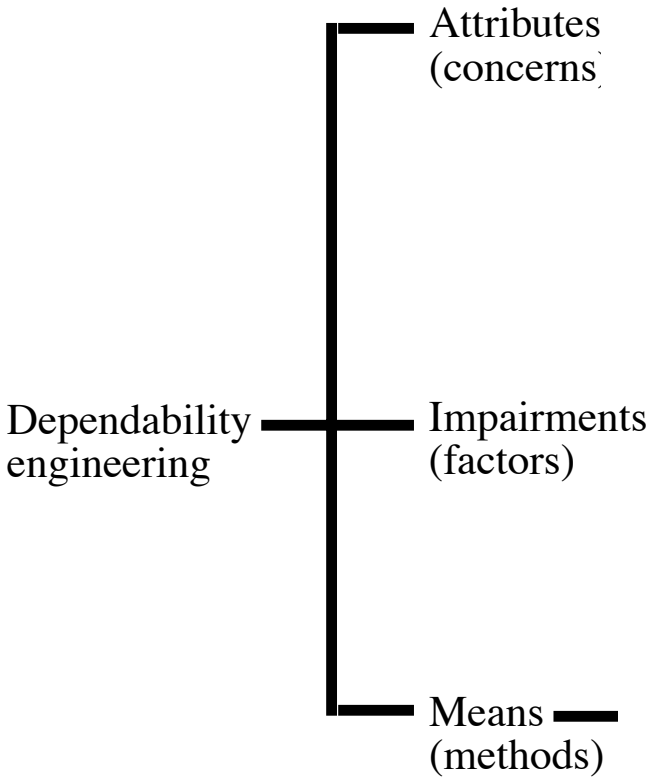
- **Red flags**
  - speed
  - ease
  - performance
  - bug-free
  - usability
  - . . .
- **Green flags**
  - configure
  - recover
  - analyse
  - replace
  - maintain
  - . . .

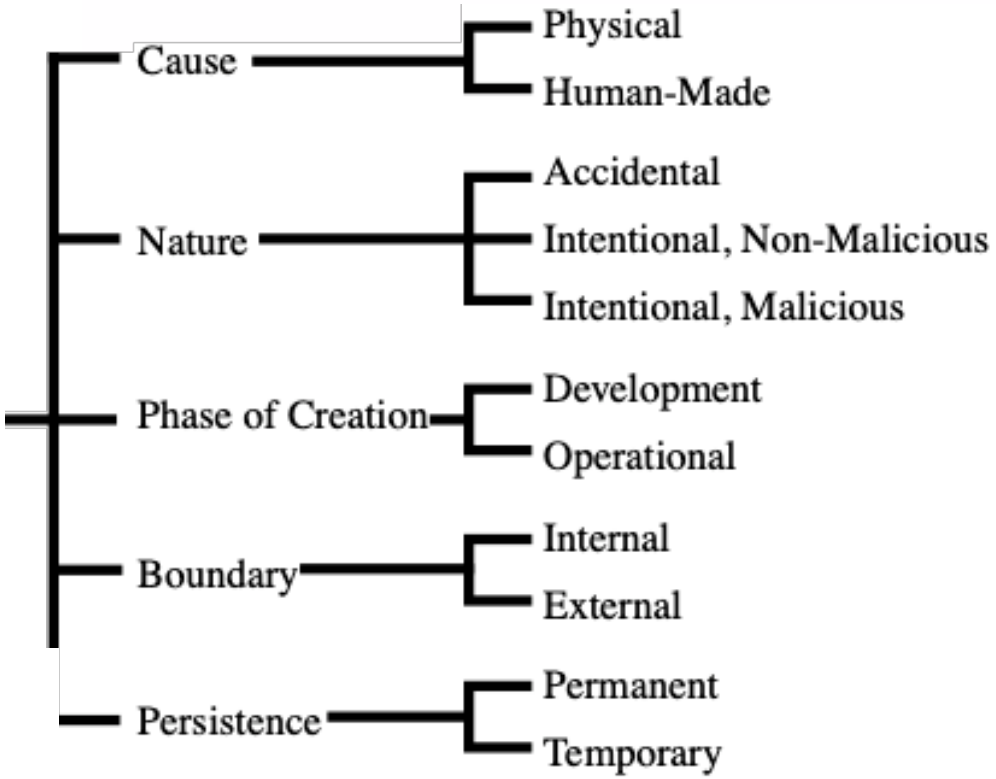
Performance



Barbacci, Klein, Longstaff, Weinstock, *Quality Attributes*, 1995

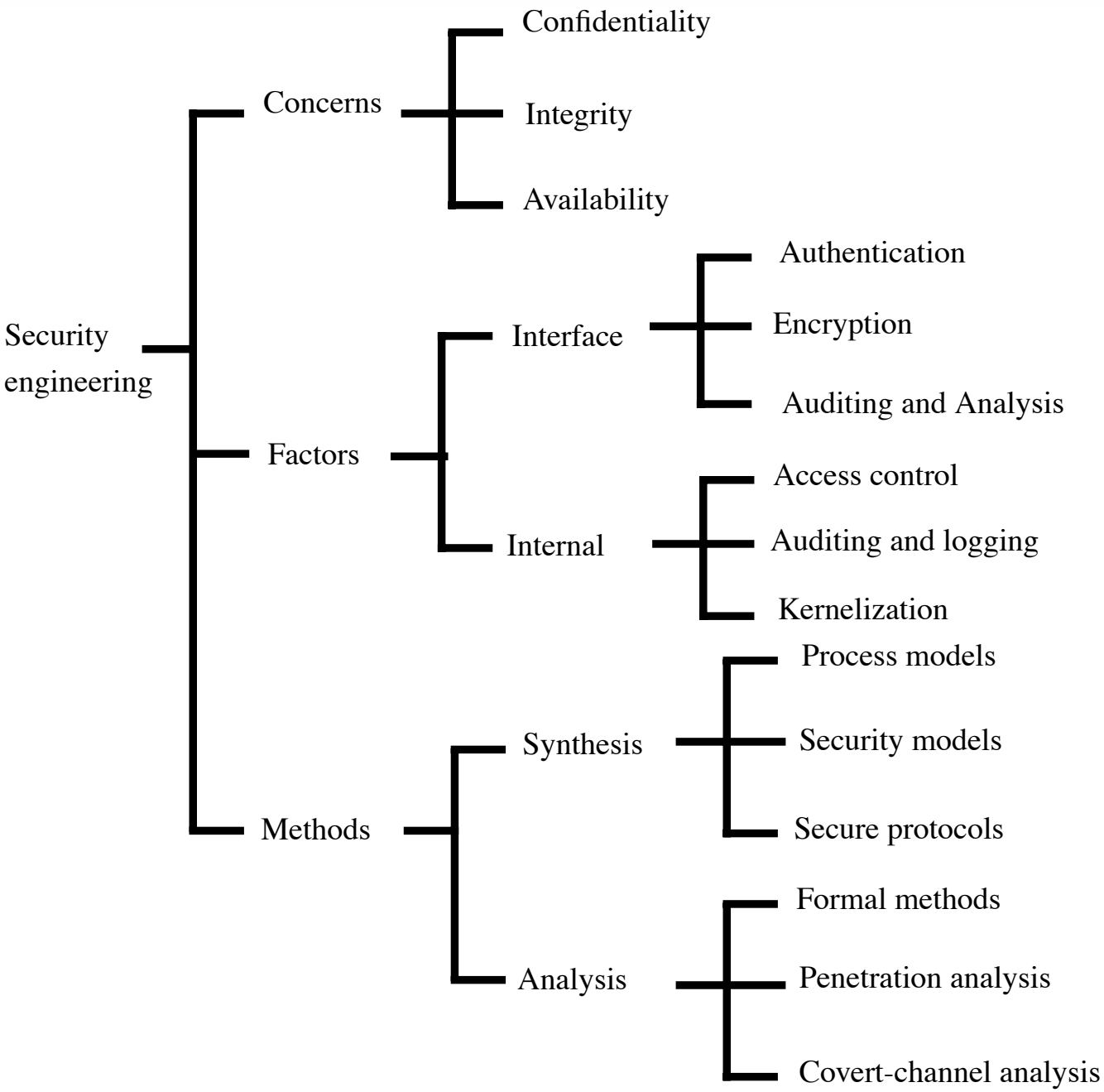
Dependability





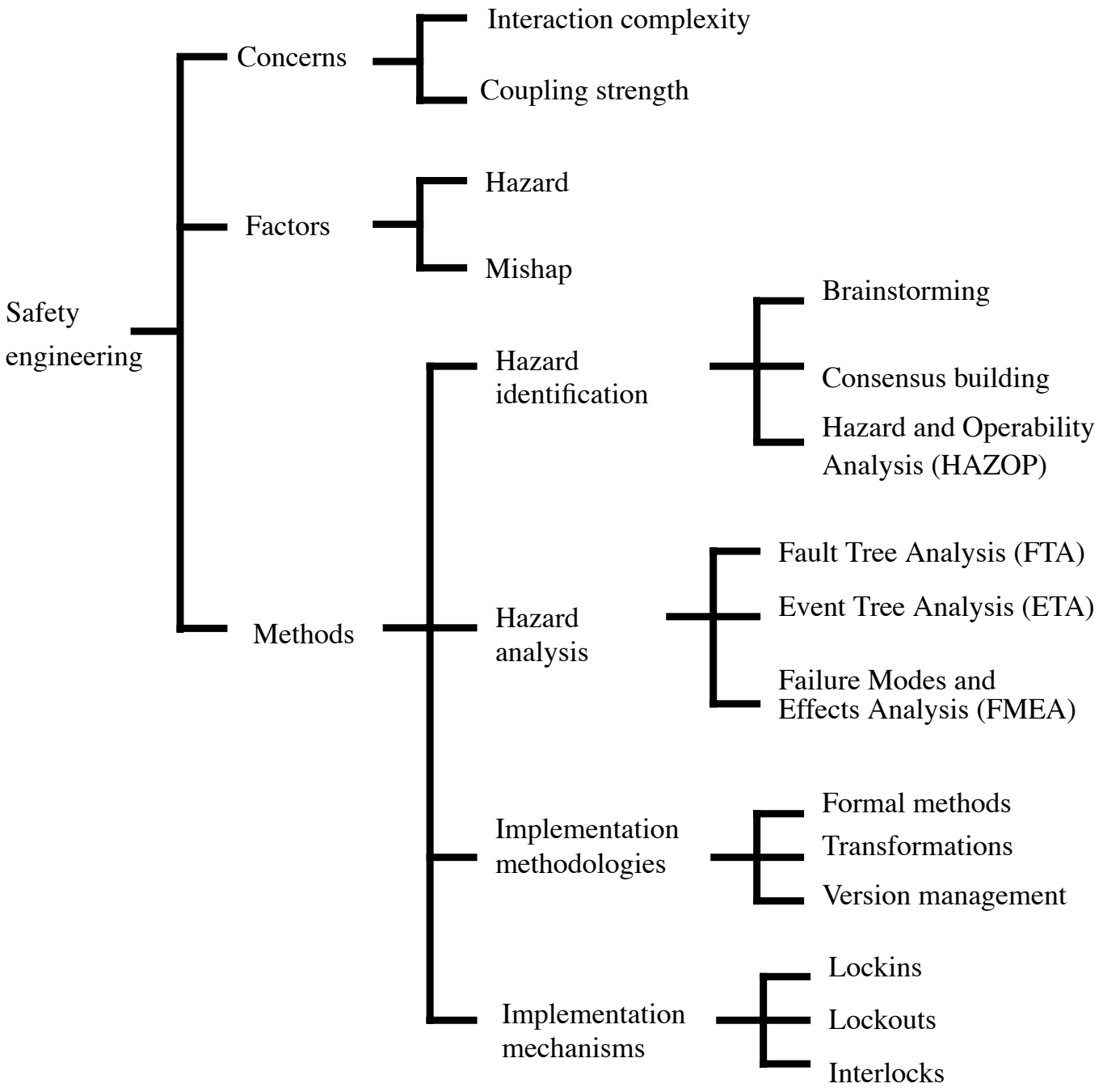
Barbacci, Klein, Longstaff, Weinstock, *Quality Attributes*, 1995

Security



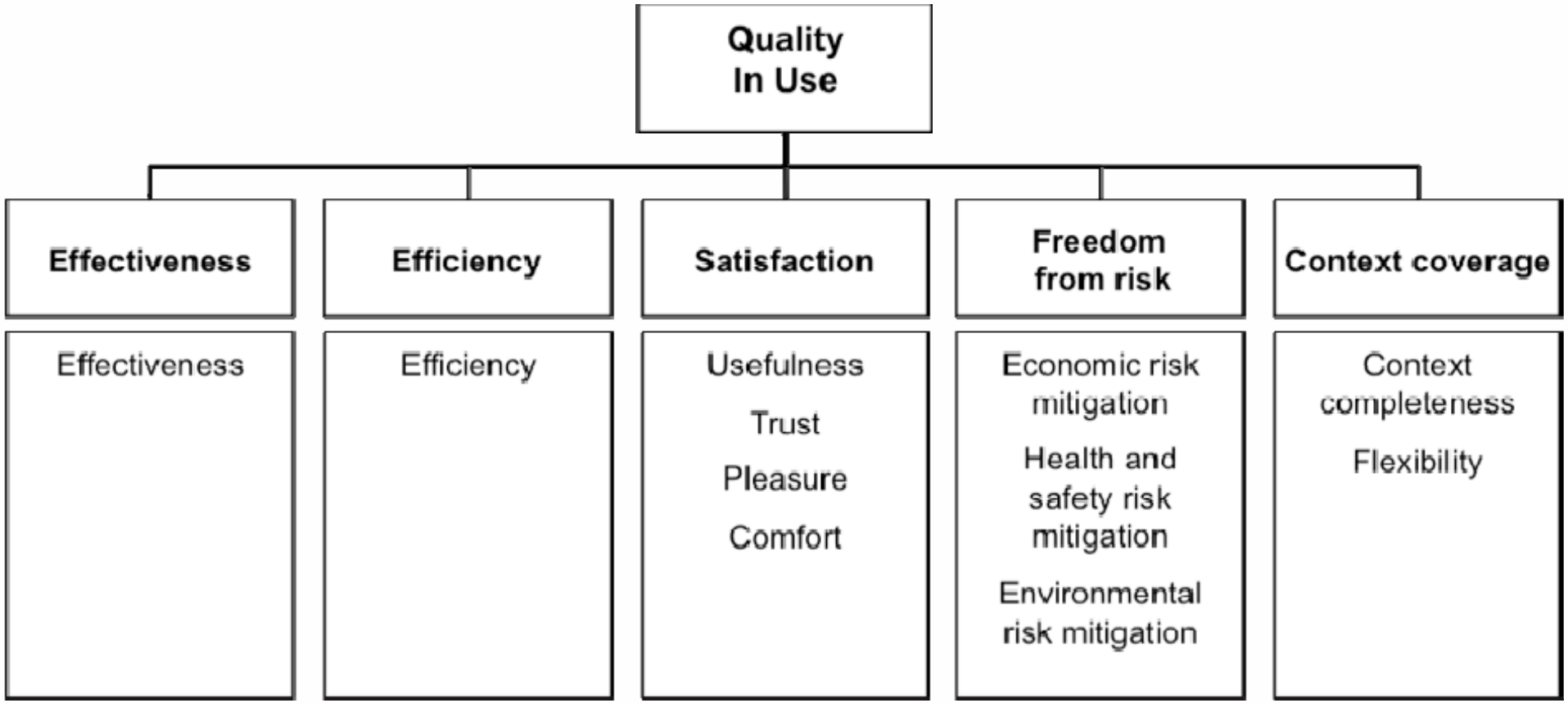
Barbacci, Klein, Longstaff, Weinstock, *Quality Attributes*, 1995

Safety



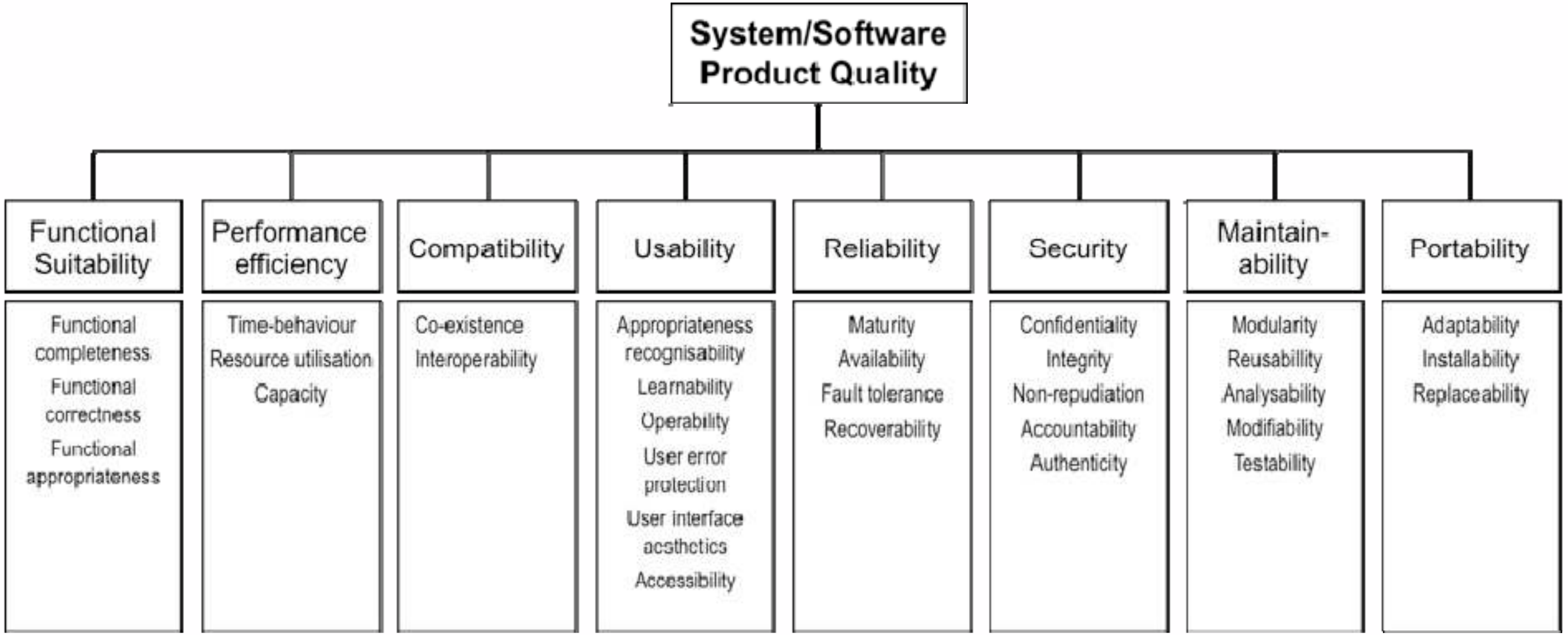
Barbacci, Klein, Longstaff, Weinstock, *Quality Attributes*, 1995

Quality in use



ISO/IEC 25010:2011(E)

Product quality



ISO/IEC 25010:2011(E)

ISO/IEC 25010:2011(E)

- **Helps**
  - great starting point
  - QAs are well defined
  - metrics are helpful
  - reusable design
- **Does not suffice**
  - SE bias
  - domain may need extension
  - too blind/rigid on metrics
  - does not guarantee completeness
  - does not ensure consistency

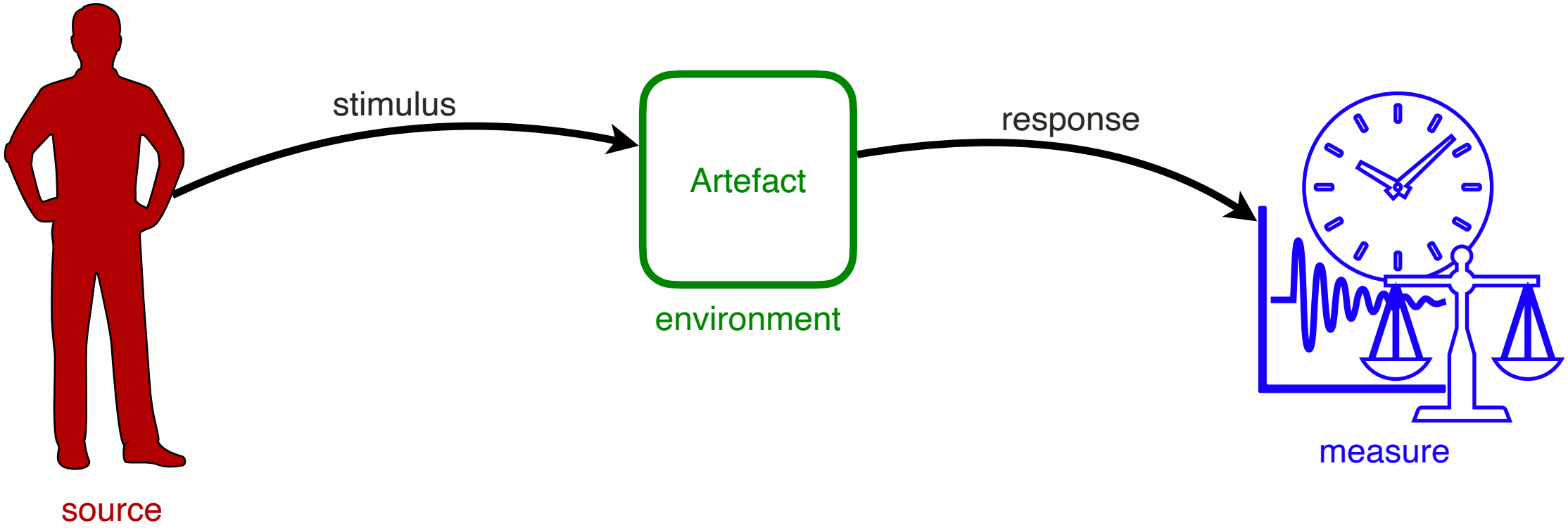
Quality attributes

- Find QAs from stakeholder interaction with the system
  - name the interaction
- Read ISO 25010:
  - To save time: start with §2; Read §§3.1-3.6
  - Scan through §4
  - Select applicable quality attributes and metrics
  - Write requirements for your case

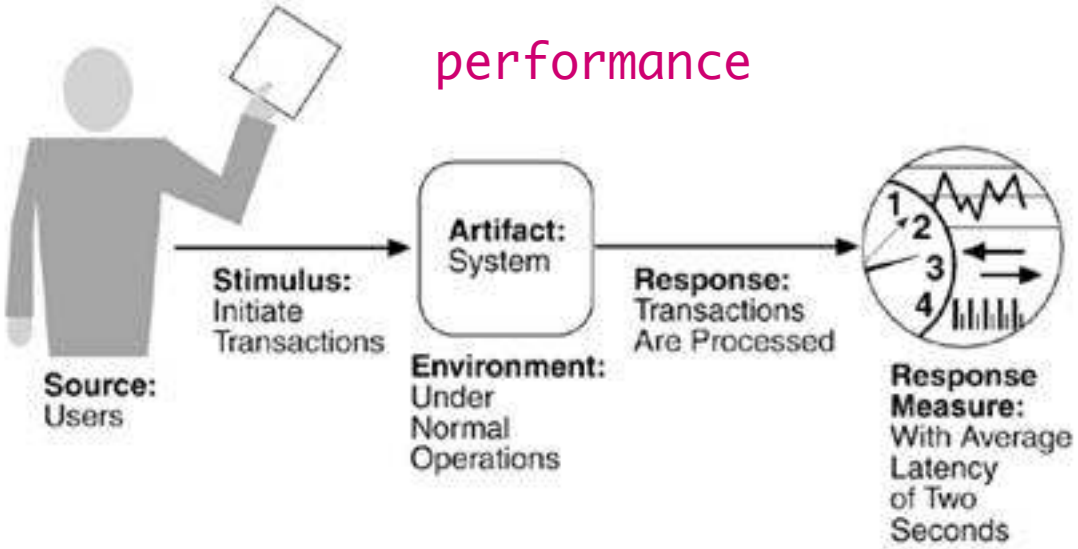
Lecture 8: Overview

- Recap
- Family Architectures
- Quality Attributes
- QA Scenarios

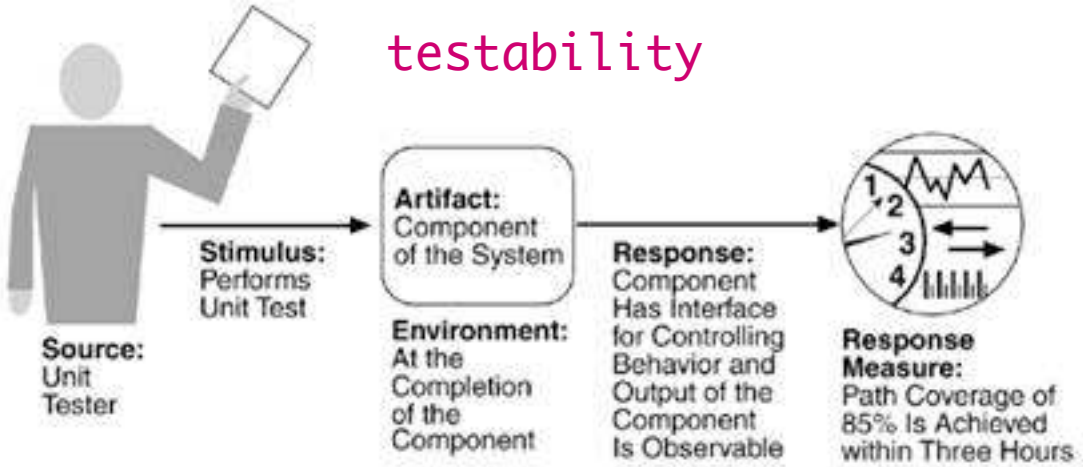
QA scenarios



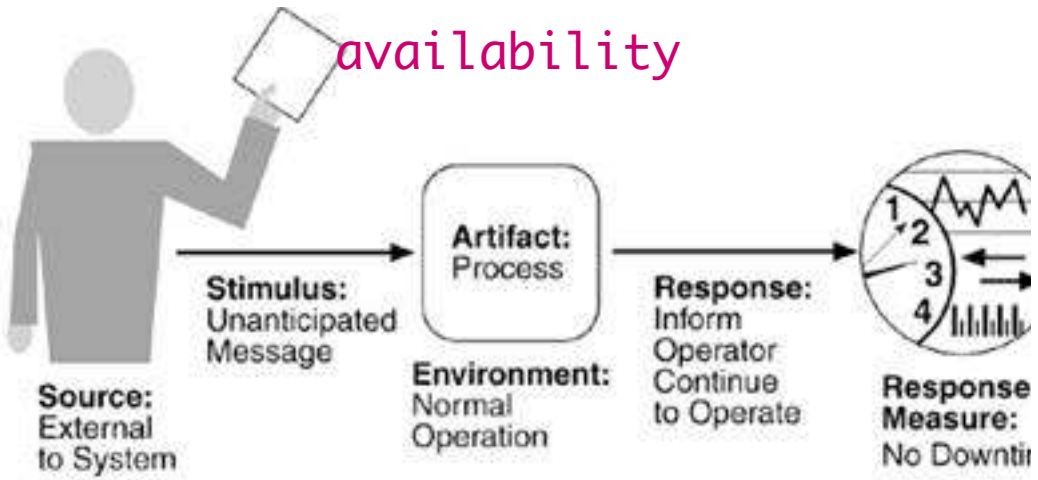
performance



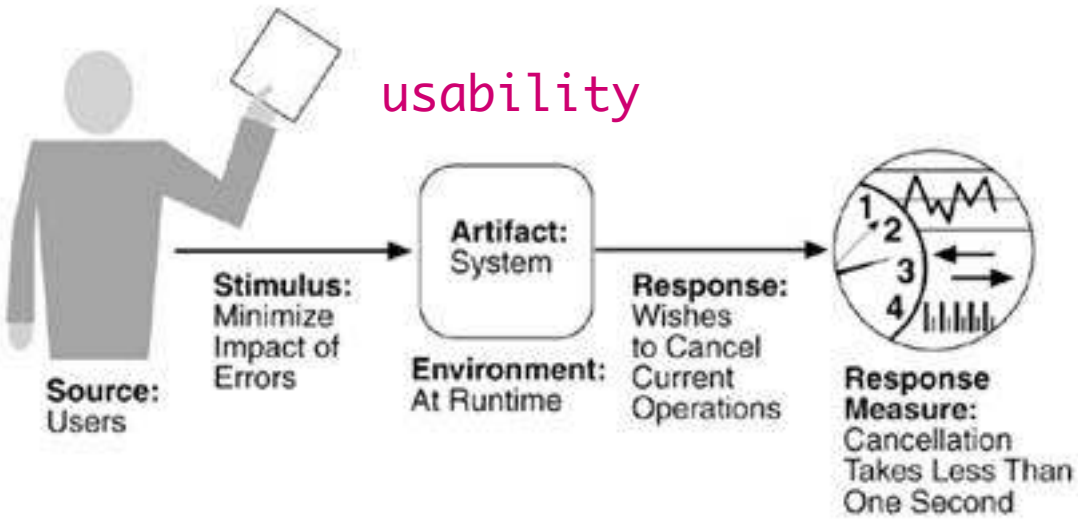
testability



availability



usability



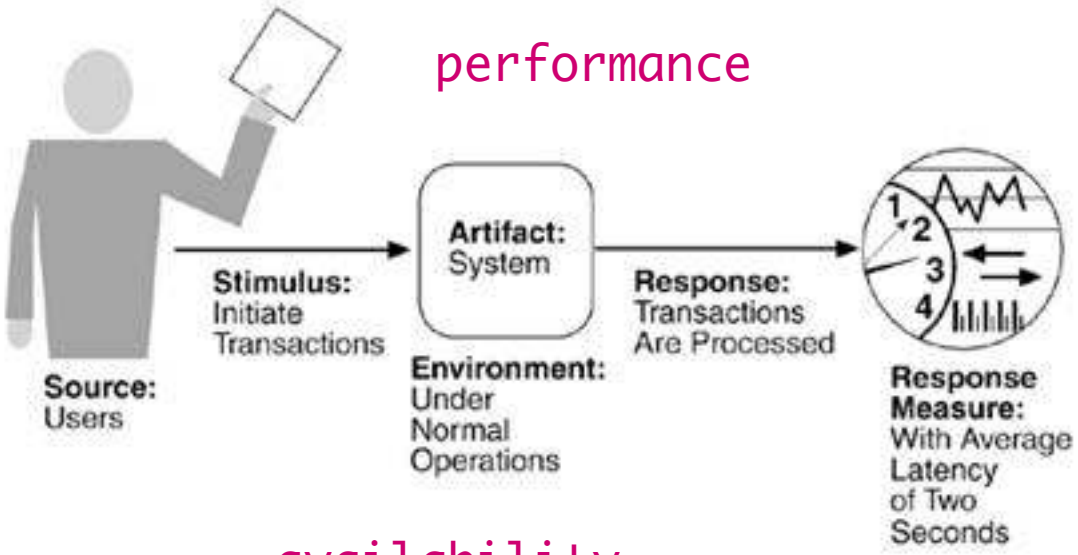
Let's fix it:

- The product must be user friendly and easy to use
- Response time should be less than 1 second
- The system should be always available
- It shall work just like the previous system, but on a new platform
- The system must never fail, and it if does, should produce a report
- The product should be accessible
- The communication protocol cannot change
- The system has to be robust
- You must support many parallel users
- The product must fit all users' individual needs

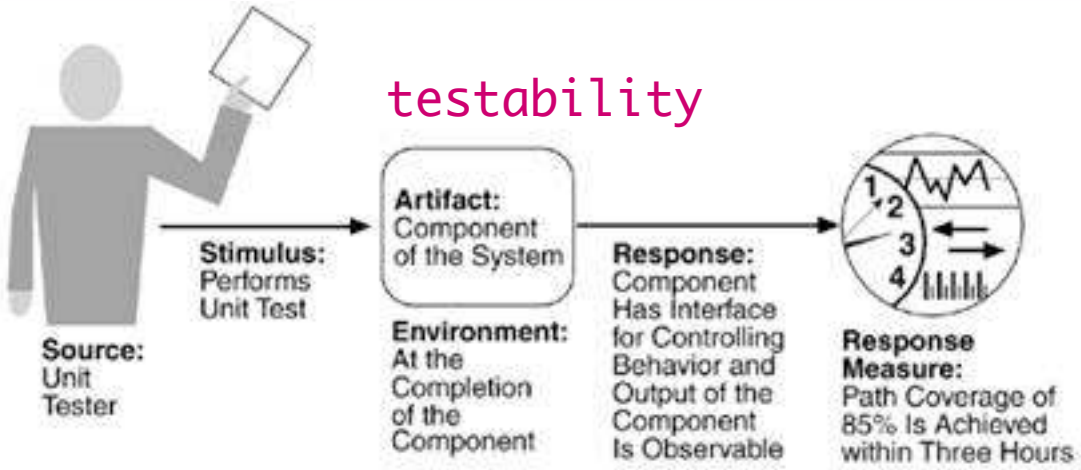
ReQtest, *10 of the Worst Requirements I have Ever Seen*, 2016

- The product must be user friendly and easy to use

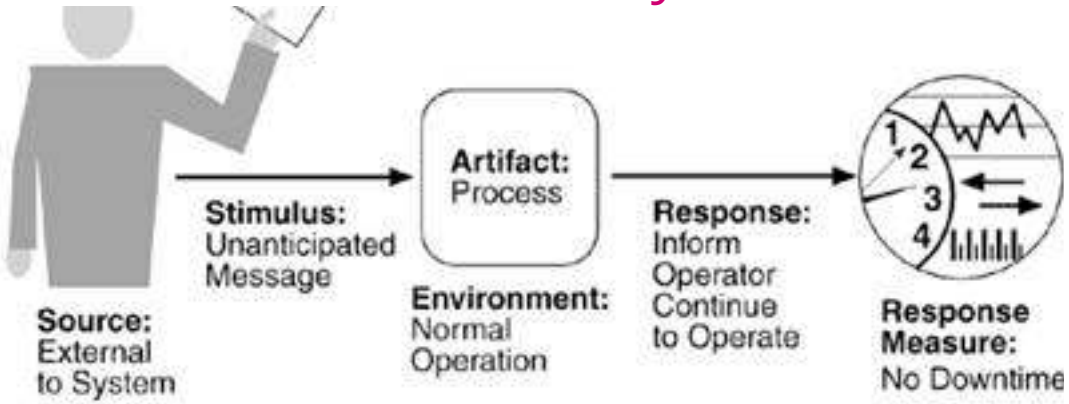
performance



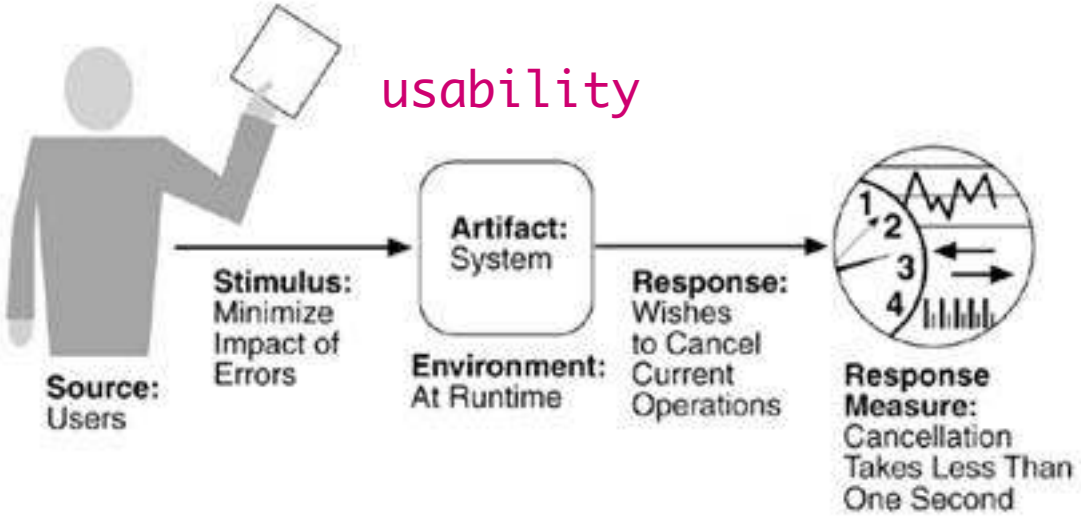
testability



availability



usability



QA scenarios

- Write/draw 3+ scenarios
  - most important quality attributes
- Be explicit about the conditions & other limits
- Explain how your design realises it
  - what happens between stimulus and response?
- Be specific about measures