UNIVERSITÄT DES SAARLANDES

30 YEARS DFKI Human Centric AI.

System Qualities & Scenarios (Non-functional Requirements)

Architectural Thinking for Intelligent Systems

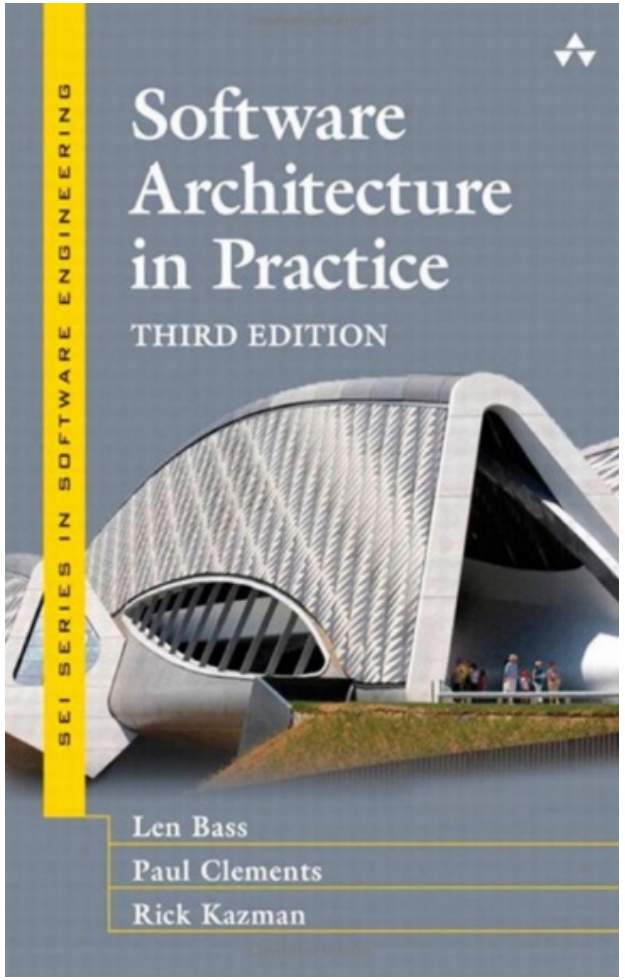
Winter 2019/2020

Prof. Dr. habil.Jana Koehler

© DFKI - JK

Agenda

- Importance of non-functional requirements
- Making qualities measurable with scenarios

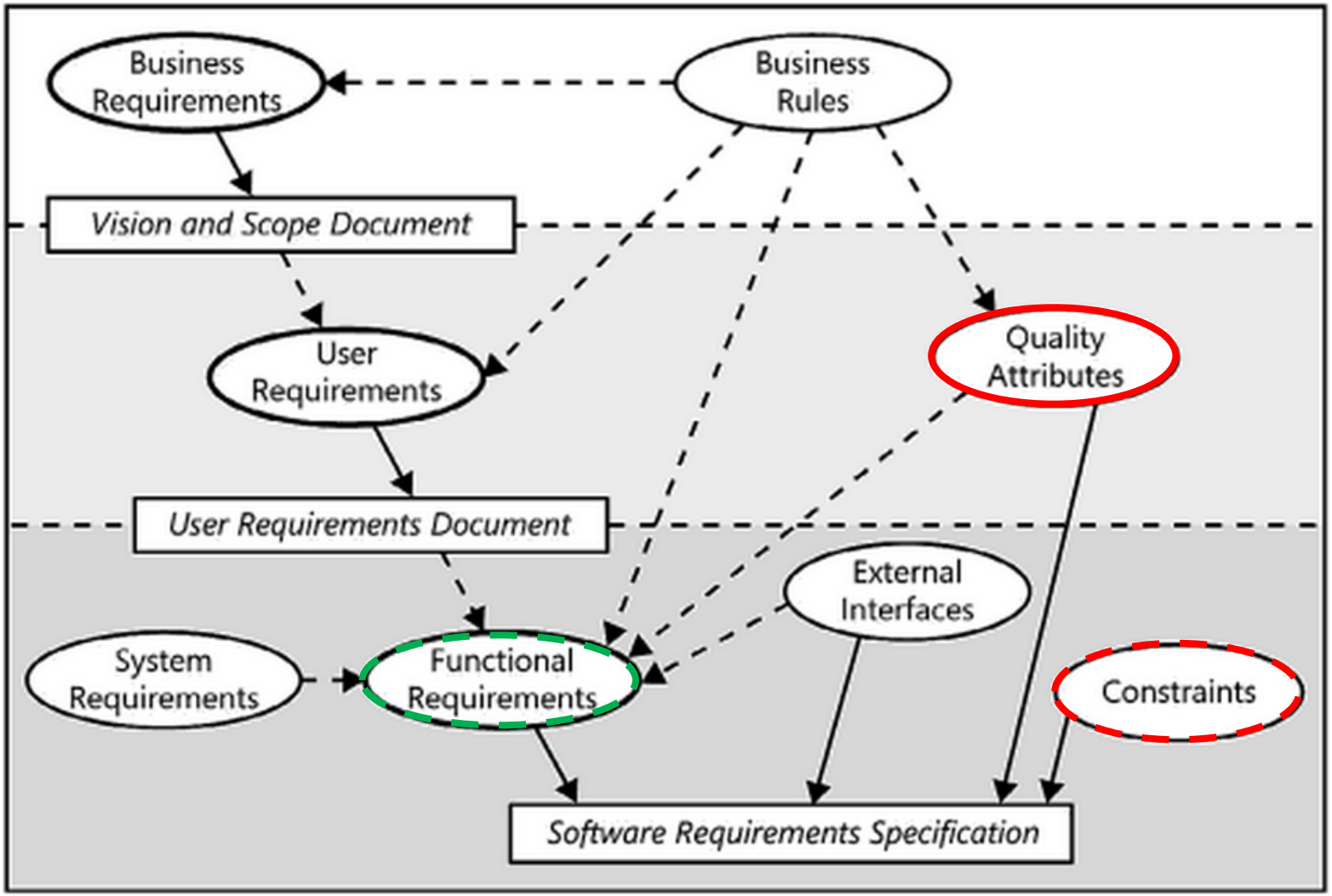


© DFKI - JK

Tutorial Assignment 5:

- We thoroughly vivisect the non-functional requirements for our system,
- write scenarios to make them measurable and
- decide which non-functional requirements we put into the focus of attention.

© DFKI - JK



Wiegers/Beatty: Software Requirements, 3rd edition 2013

© DFKI - JK

System Qualities and Non-functional Requirements

- Functional requirements
  - System functionality = system capabilities, services, responsibilities & behavior
  - *«What does it offer to me, what can I do with it?»*

- Non-functional requirements
  - How system functionalities are fulfilled
  - *«How well does the system meet my needs?»*
  - *«How well does it offer ist services to me»*
  - What decisions have already been taken (constraints)

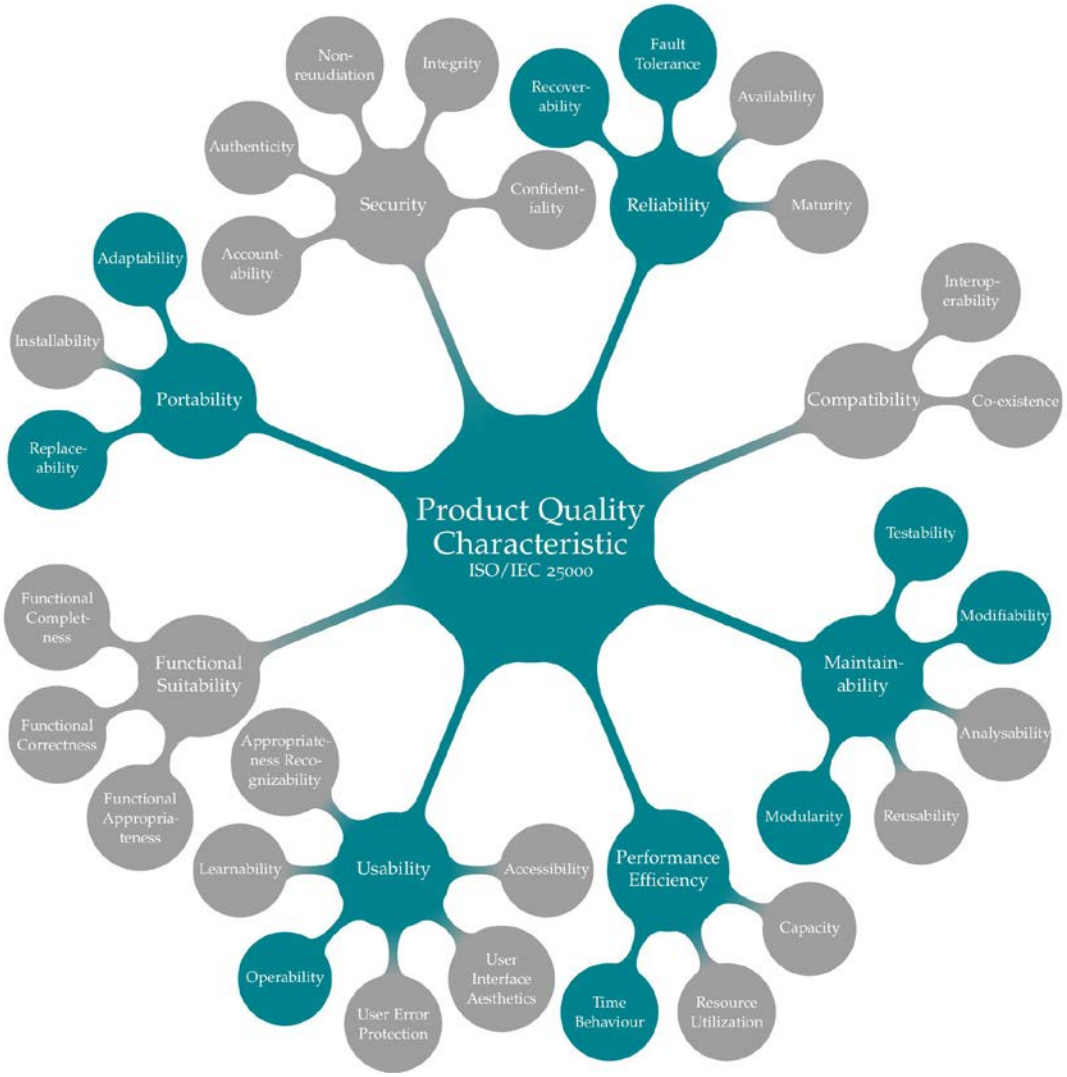
© DFKI - JK

Systems undergo change because maintenance or portability is difficult, or because performance, scalability, security, usability ... are insufficient.

Functional enhancements come in 2nd place.

© DFKI - JK

System Qualities determine System Architecture



© DFKI - JK

Quality Attributes

- Measurable & testable!!!

- Quality attributes are fulfilled by an architecture by anchoring specific structures with specific interactions and a certain behavior in the architecture

- Functional requirements are met by transferring responsibilities to specidic elements in the architecture

  ➢ **Architecture is not determined by functionality, but only by system qualities!**

© DFKI - JK

2 Relevant Groups of Quality Attributes

- Qualities of the system at runtime = observable behavior
  - Availability
  - Performance
  - Usability
  - Interoperability
  - Security
- Qualities of the system at development time
  - Modifiability
  - Testability
  - Reusability

© DFKI - JK

Quality Attributes are not Independent of each Other

- Quality can only be measured indirectly, not absolutely
- Quality is relative and different for different stakeholders
- Quality of architecture <> Quality of software
- A complete implementation of the functional requirements does not allow statements to be made about the quality achieved

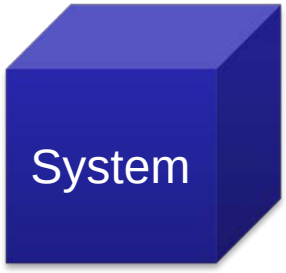
- Quality attributes are at the heart of architectural thinking
  - Need to make compromises in decisions
  - "Trade-offs"

© DFKI - JK

How can we describe Quality Attribute Requirements?

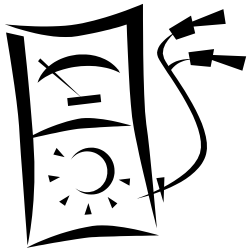
Stimulus Source

Stimulus



Environment Artifact

Response

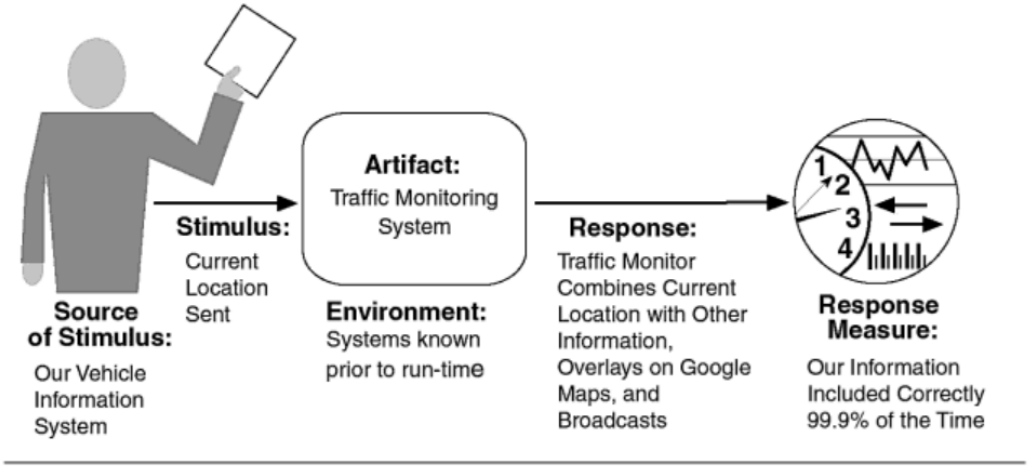


Response Measure

- **"Scenario"**
  - **What happens when a stimulus affects a system in a certain situation?**

© DFKI - JK

Example

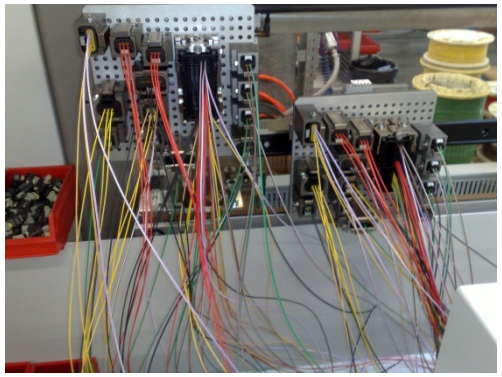


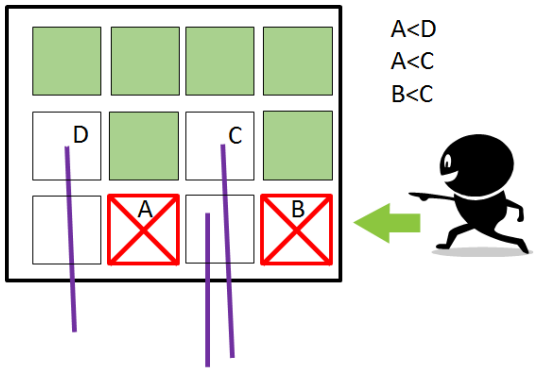
Sample concrete interoperability scenario

Part of the Scenario	Description
Stimulus source	Car information system
Stimulus	Message regarding the current position
Environment	Traffic Monitoring system (+ located vehicles)
Artifact	Message receiver (sensor) of the traffic monitoring system
Response	Current position transmitted from sensor to traffic monitoring server
Response measure	99.99% of all captured positions reach the server

© DFKI - JK

Example





Part of the scenario	Description
Stimulus source	Komax technician
Stimulus	Desire for visualization of constraints between 2 housings
Environment	Topwin Software
Artifact	AI System GUI
Response	Display of all constraints
Response metrics	With only 2 clicks in 2 scrollable windows

© DFKI - JK

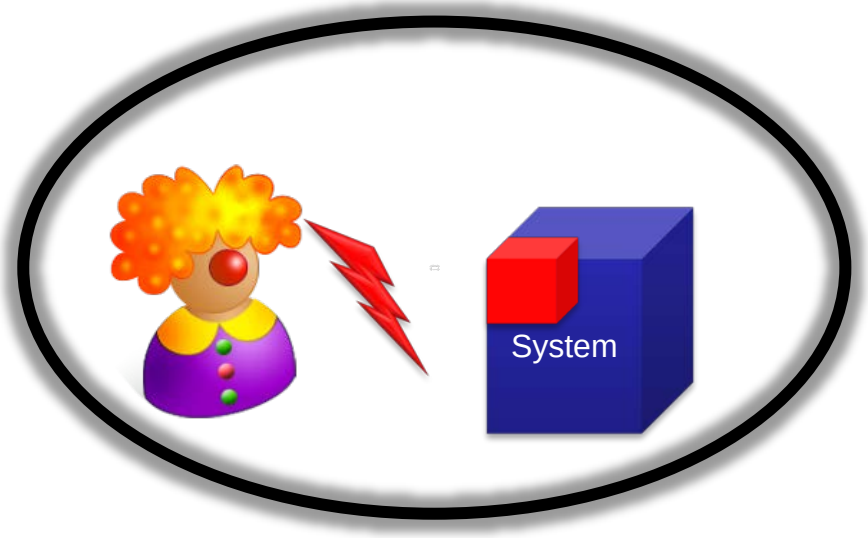
How can we Analyze Quality Attributes?

1. Consult the sources of the request and further break down the attribute – «what exactly is meant?»
   - Collect concrete scenarios for the attribute
   - Analyzing the collected stimuli, responses, measures – what is the essential core of the attribute?

2. Find tactics and patterns that support the quality attribute
   - How can the required responses be achieved?
   - What can cause a system fail to achieve the attribute and which system structures can prevent the failure?

© DFKI - JK

Environment & Artifact



- **Environment**: In which context, under which circumstances and boundary conditions does a stimulus occur?
  - e.g. modification request before or after “code freeze”
- **Artifact**: Part of the system that must fulfill the attribute
  - GUI, database, but often the whole system as well

© DFKI - JK

Specification of Scenarios

1. Stimulus source
2. Stimulus
3. Environment
4. Artifact
5. Response
6. Response measure

- Scenarios make non-functional requirements (the quality attributes) SMART
- It often requires more than one scenario per non-functional requirement

© DFKI - JK

Scenario Pattern for Usability

Part of the Scenario	Description
Stimulus source	Stakeholder (user, architect, developer,...)
Stimulus	Stimulus source tries to: - learn how to use the system efficiently and correctly - understand the operating or usage concepts of the system - minimize the effects of operating errors - adapt system to own needs (configure) - understand system and software architecture
Environment	Normal operation, runtime, installation or configuration time
Artifact	Complete system (including user interface, control flow, architecture documentation)
Response	- System supports by examples or explanations - Incorrect operation has local/overlapping effects - System (part) is not configurable - System and software architecture with underlying concepts is not documented
Response measure	- Time spent, number of errors, number of user goals/tasks achieved, user satisfaction, increase in knowledge, ratio of successful uses to total uses, proportion of errors occurring - Extent of damaged data, loss of time, abortion of interactions

The most important System-Independent Quality Attributes

© DFKI - JK

Availability (availability/reliability)

"Ninety percent of life is just showing up." Woody Allen

- Functionalities are available and usable when needed

Stability
Reliability

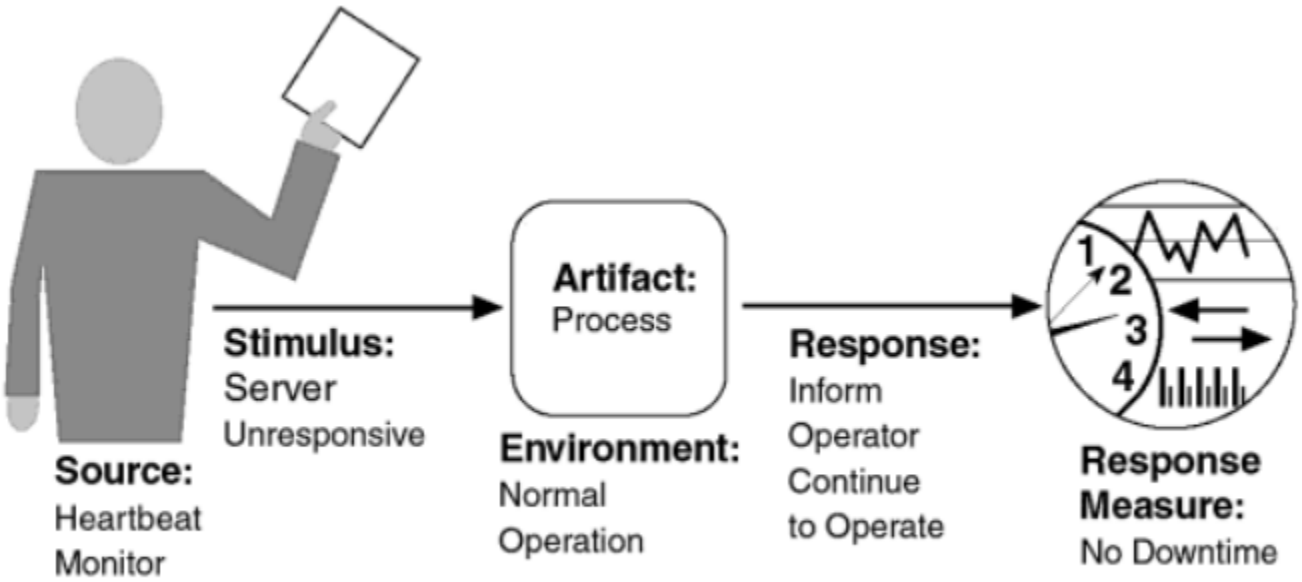
+

Ability to avoid failure or to restore functionality after failure within a specified period of time

- Which errors occur (internal, external)? How often? What response do they trigger? How are they prevented? How are they noticed, communicated, and corrected? How long is the system or artifact "down"?

© DFKI - JK

Example



Sample concrete availability scenario

© DFKI - JK

Scenario Pattern for Availability

Part of the scenario	Value
Source	Internal/external: people, hardware, software, physical infrastructure, physical environment
Stimulus	Fault: omission, crash, incorrect timing, incorrect response
Environment	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation
Artifact	Processors, communication channels, persistent storage, processes
Response	- Prevent the fault from becoming a failure (breakdown of system parts) - Detect the fault: logging, notifications (people, system) - Recover from the fault: disable source of events causing the fault, be temporarily unavailable while repair is being effected, fix or mask the fault/failure or contain the damage it causes, operate in a degraded mode while repair is being effected
Response measure	- Time or time interval when system must be available, availability percentage - Time to detect/repair the fault, time in degraded mode … - Number or percentage of faults the system can detect, prevent, handle without failing

© DFKI - JK

Types of Errors (possible Stimuli)

- Omission
  - no response to a stimulus
- Crash
  - Omissions that occur repeatedly
- Timing
  - Response occurs at the wrong time (too early, too late)
- Response
  - Incorrect response of the system

© DFKI - JK

Responses

- Prevention
  - Add redundancy, safety functions, load balancing to system architecture
- Detection & Isolation
  - Logging of the error
- Recovery
  - Notify users and other systems
  - Start actions to limit potential damage
  - Limit availability or functionality of the affected system (parts)

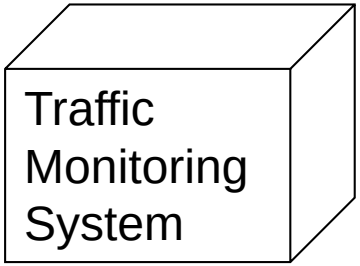
© DFKI - JK

Interoperability

- Degree to which two or more systems can exchange information in a meaningful way
  - syntactic data exchange (interfaces)
  - semantic data interpretation
- With whom? With what? Under what conditions?

My car

Entrance to city & time



Env: system known

99.99% of all invoices are correct

© DFKI - JK

Modifiability / Extensibility

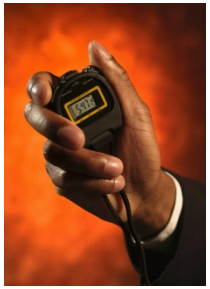
- Degree to which a stakeholder is able to make changes in the system
  - What can change?
  - How likely is a change?
  - When does a change happen and by whom?
  - What will it cost? What is the risk?

User

GUI change

" tile surface"

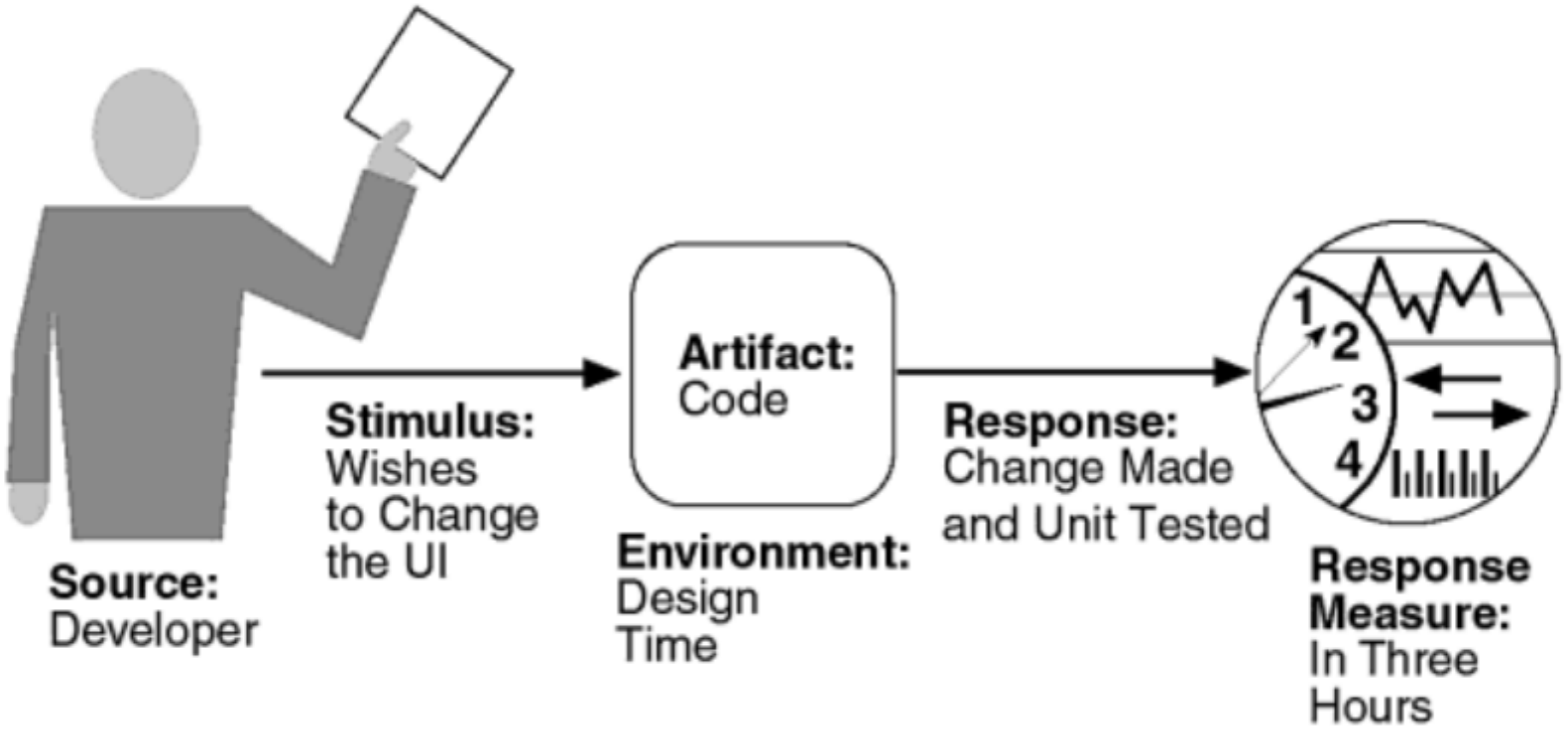
Delete tile or add new one



< 1 second

© DFKI - JK

Example



Sample concrete modifiability scenario

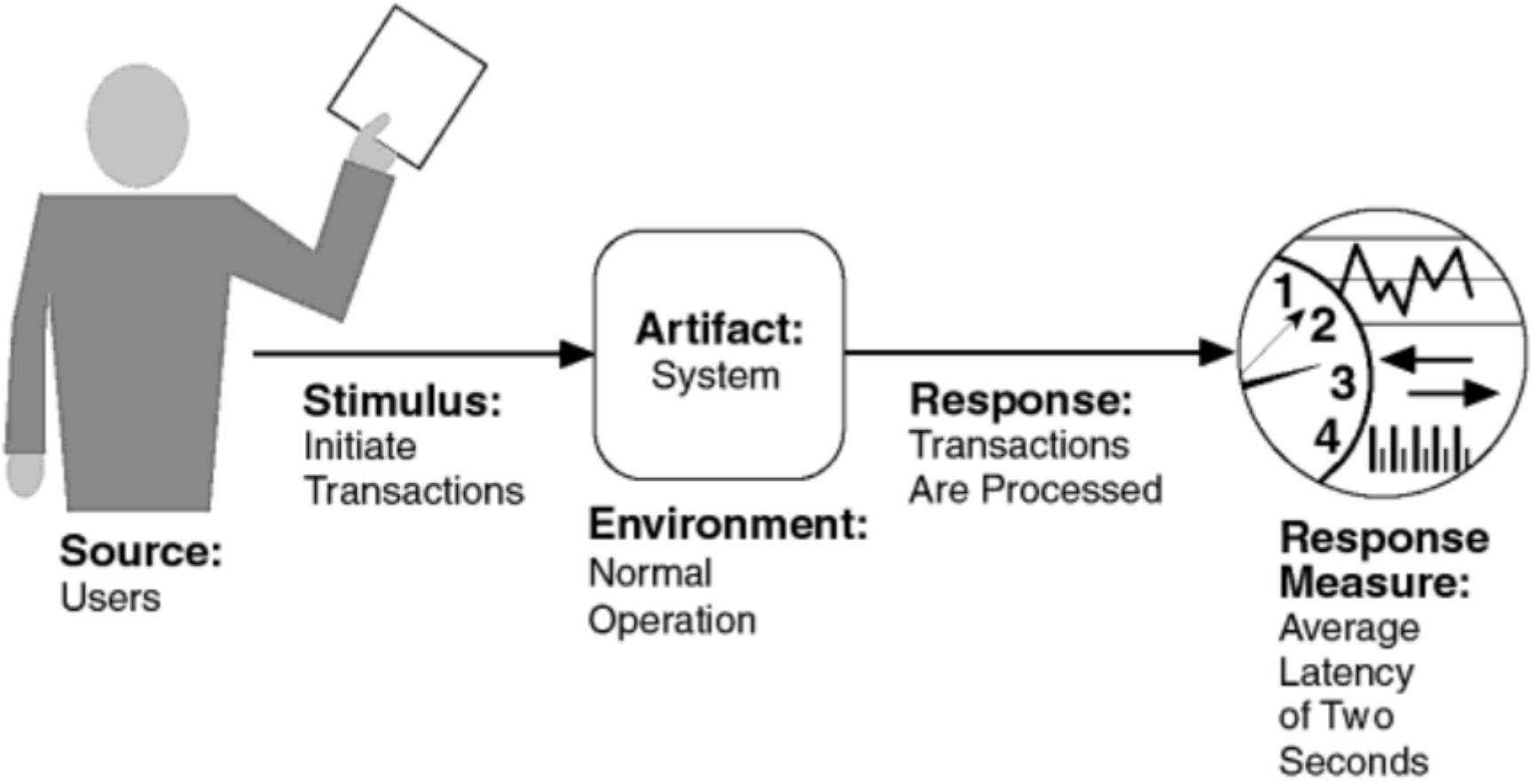
© DFKI - JK

Performance

- System reaction in a certain time to a certain event
  - often associated with scalability requirements
- Events occur periodically, stochastically, sporadically
- Measurement of Response
  - latency: time between event occurrence and system reaction
  - deadlines, throughput (transactions/s), jitter (latency variance, number of unprocessed events)

© DFKI - JK

Example



Sample performance scenario

© DFKI - JK

Scenario Pattern for Performance

Part of the scenario	Description
Stimulus source	internal or external
Stimulus	Occurrence of the stimulus (periodic, stochastic, random)
Environment	Normal operation, high load, overload, emergency operation
Artifact	Complete system (or certain parts: servers, databases)
Response	- Processing the event (execution behavior) - Change in execution behavior (complete vs. partial processing) - Restricted use of functions, data, etc. - Change in runtime behavior/resource usage by the system
Response measure	- Latency, response time, throughput - Error rates, amount of lost data or no longer available functions - Fluctuations in loads and response times - Use specific quantities and durations/time points!

© DFKI - JK

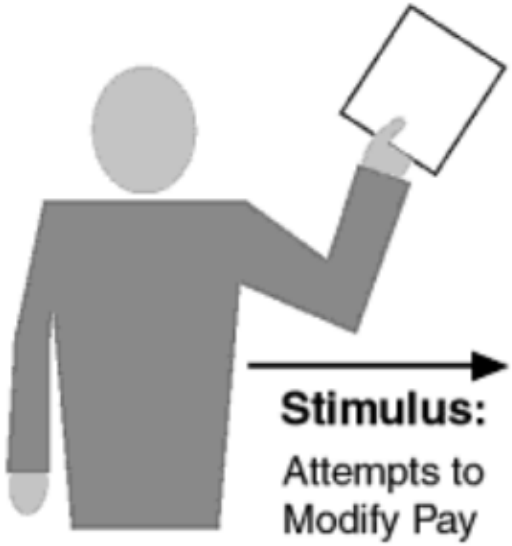
Security

- Ability of a system
  - to protect data and information from unauthorized access, but allow authorized access by users (and other systems)
  - to ensure information integrity (protect against unauthorized manipulation)
  - to be available for legitimate use
  - to recognize, resist, respond to and recover from attacks

- Example: "Denial of service" attack does not prevent that books can be ordered in an online shop

© DFKI - JK

Example



Source:
Disgruntled Employee from Remote Location

Stimulus:
Attempts to Modify Pay Rate

Artifact:
Data within the System

Environment:
Normal Operations

Response:
System Maintains Audit Trail

Response Measure:
Correct Data Is Restored within a Day and Source of Tampering Identified

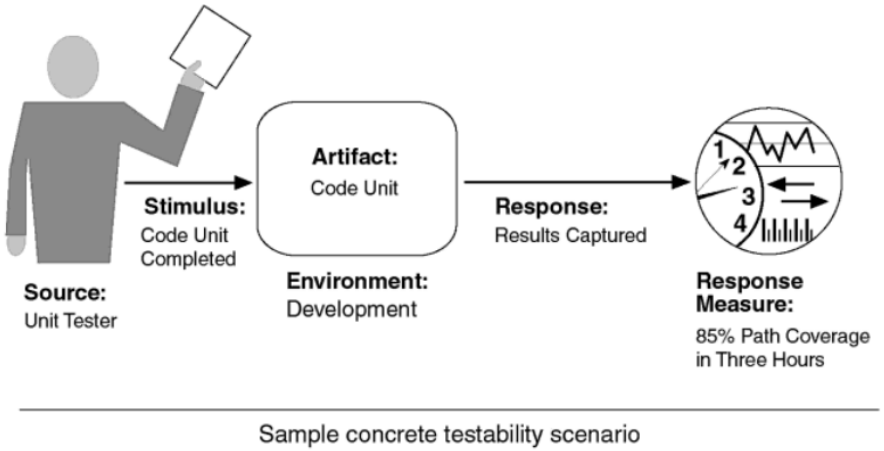
Sample concrete security scenario

© DFKI - JK

Testability

- How easy (or difficult) is it to detect (and locate) errors in the system?
- «*A system is testable, if it gives up its faults easily*»

- Source: unit tester
- Stimulus: code unit completed
- Artifact: code unit
- Environment: development
- response: written & executed tests with results
- measure: 85% path coverage in 3 hours



Sample concrete testability scenario

© DFKI - JK

Usability

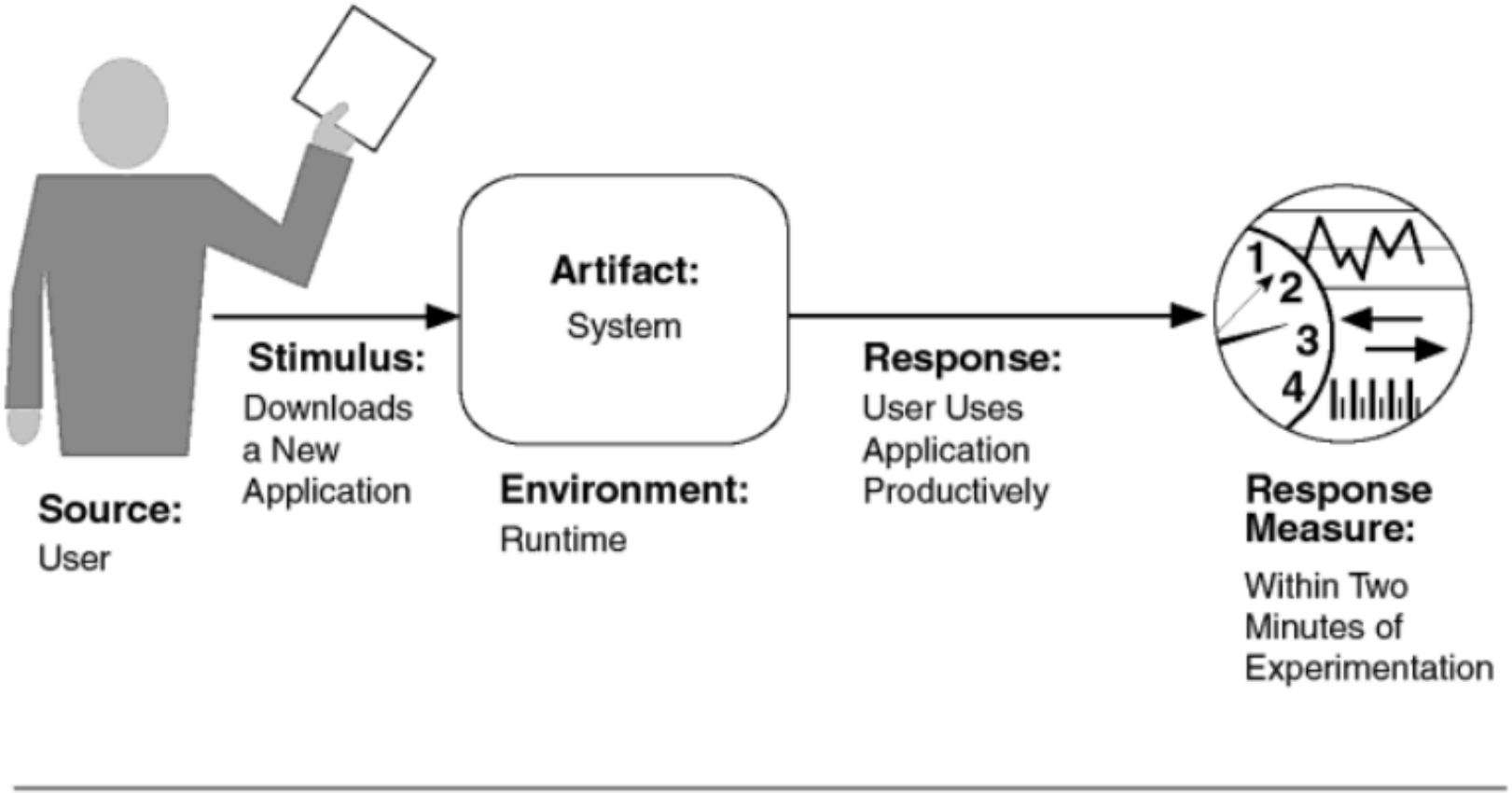
- How easy is it for the user to use the system?
- What support does the user receive?

- Learning to use a new system, «self-explaining»
- Adequate information visualization
- Effects of erroneous user behavior
  - Adaptability of the system to the user
  - Confidence and satisfaction of the user

- «App x can be used to <x> within 1 minute after download»

© DFKI - JK

Example



Sample concrete usability scenario

© DFKI - JK

More Quality Attributes

- Portability
- Scalability
- Mobility
- Controllability
- Safety (Can input from one user become a danger to others?)

- Conceptual integrity of the architecture
- Marketability

© DFKI - JK

Example

- Quality attributes in destination control
  - Usability: what happens when transportation capacity is exceeded?
  - Portability: how to adapt to different buildings and users?
  - Performance: what to optimize?

- Quality attributes in cable wiring control software
  - Usability: ease of applying the software
  - Scalability: how many cables can the solver handle?

© DFKI - JK

Quality Characteristics in ISO/IEC 25010:2011 (Reviewed and confirmed in 2017)

Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models

© DFKI - JK

Summary

- Non-functional requirements (system qualities) define the architecture!
- Put the 2 most important quality attributes in the focus of architectural thinking
- Use scenarios to make qualities measurable
- Writing good scenarios takes time and requires a detailed analysis and understanding of stimulus source, stimulus, environment, artifact, response and measure
- Scenario patterns provide helpful guidance
- Write more than one (many!) scenarios for each quality attribute

© DFKI - JK

Working Questions

1. What role do non-functional requirements and quality attributes play in architectural thinking?
2. Give examples of quality attributes.
3. How do you describe a quality attribute using scenarios?
4. What is the relationship between a quality attribute and a use case? How would you add information about a quality attribute in a use case description?

© DFKI - JK