Quality attributes of software architectures

Paolo Ciancarini

Agenda

- Exercise: Architectural qualities
- *ISO25010: System and software quality models*
- A model for expressing and testing architectural requirements

Quality attributes

- What is quality?
- When a new system (eg. a software) is built, customers have wishes and want “quality”
- Some wishes are functional requirements
- Quality wishes are non functional requirements

examples

- I wish a software to play chess (functional wish) so that I can win versus the world champion (quality wish)
- I wish an app to call people (functional wish) so that I save money (quality wish)
- I wish a system to drive my car (functional wish) so that I have no accidents (quality wish)

Requirements (definition)

- *A Requirement is a condition or capability that must be met by a system or component to satisfy a contract, standard, specification, or other form of request*
- *Non-Functional Requirements are requirements specifying general properties of a system, not its specific functional behaviour*

Non functional requirements

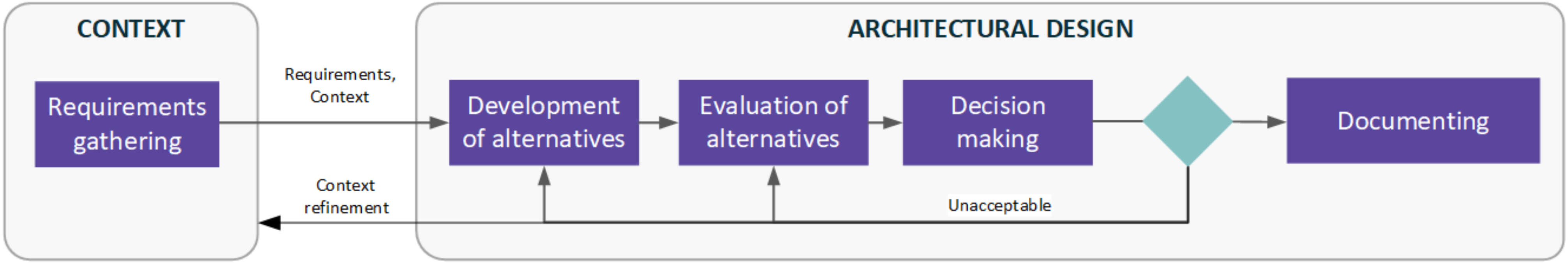
- Requirements are sometimes described as being part of the FURPS spectrum, meaning **F**unctional, **U**sability (UX), **R**eliability, **P**erformance and **S**upportability
- The latter 4 types – URPS - are collectively called *non-functional requirements*.
- The spectrum is usually expanded to FURPS+ since there are several more important categories of non-functionals

Non functional vs functional

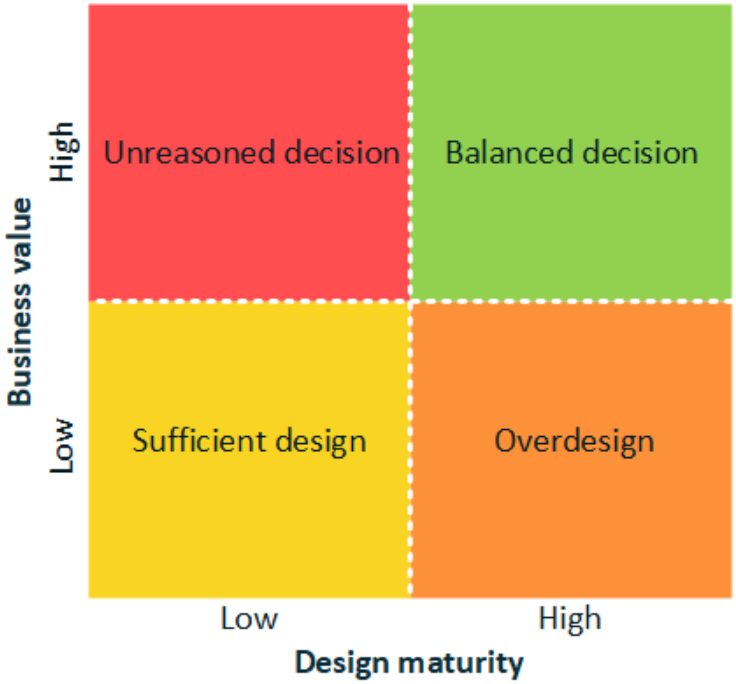
- Non-Functional Requirements (NFRs) are considered to be the most important from an architectural perspective
- They are "architectural requirements"

- REMEMBER: a Product must meet both its functional and non-functional requirements to provide business value.

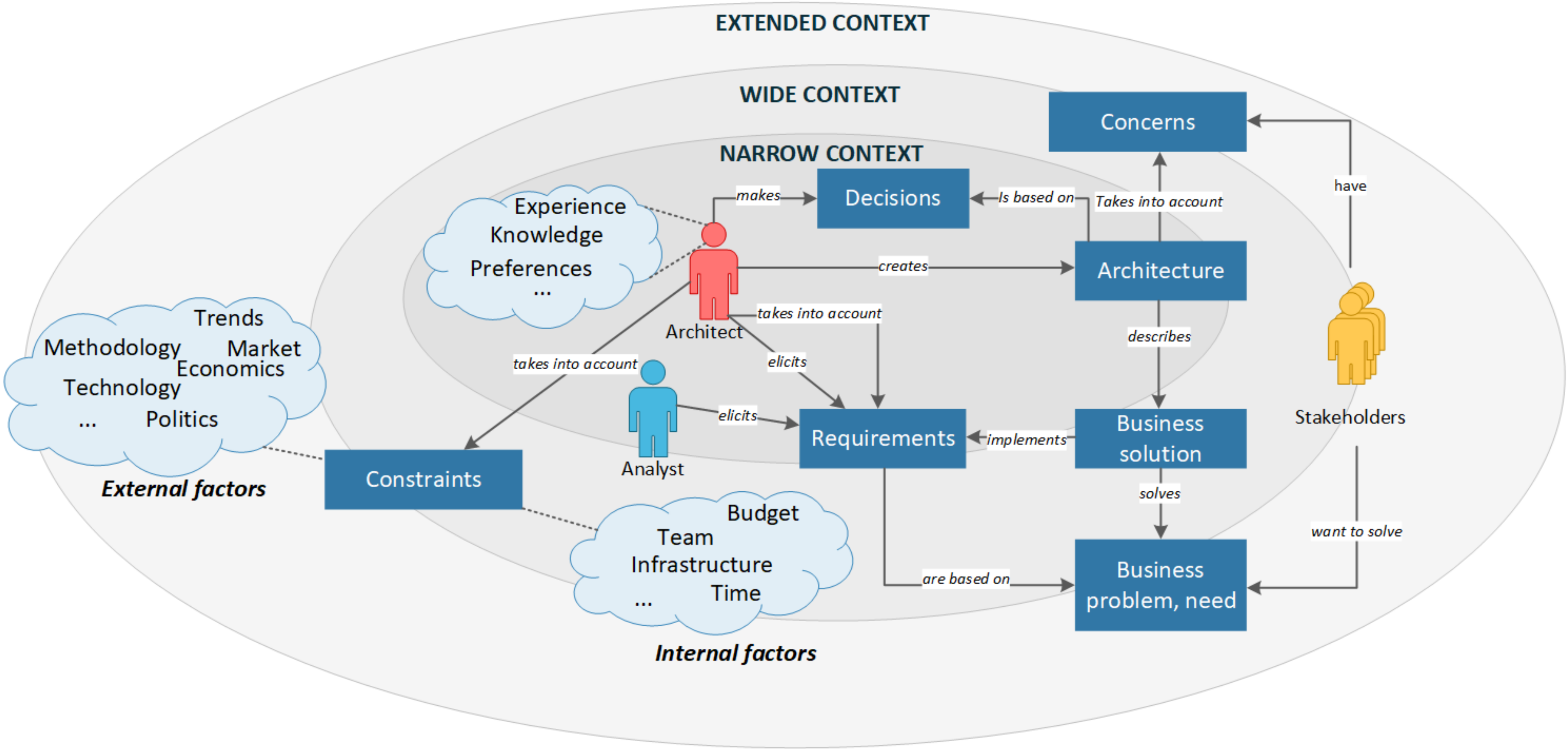
Context and design



https://www.iasaglobal.org/on-making-architectural-decisions/



Context and architecture



How to start designing an architecture: main questions

- The sw architecture of any system is strongly influenced by non functional requirements

- Architectural design is a set of decisions to satisfy all requirements, both functional and non-functional

I. How to express the qualities we want our architecture to provide? Eg., what does it mean to say that a system is performant, or modifiable, or reliable, or secure?

II. How do we test or measure these non functional requirements?

Some important qualities

- Performance
- Efficiency
- Usability
- Modifiability
- Security
- Testability
- Availability

- Time to market
- Cost and benefit
- Projected system lifetime
- Targeted market
- Rollout schedule
- Integration / Legacy

Some system qualities are “architectural”

- Qualities of the system, eg. performance or modifiability
- Business qualities (such as time to market) that are affected by the architecture.
- Qualities, such as *conceptual integrity*, that are about the architecture itself although they indirectly affect other qualities, such as modifiability.

  - Conceptual Integrity the architecture is coherent
  - Correctness the design is correct wrt to the requirements
  - Completeness the design covers all the requirements
  - Flexibility the architecture supports future changes to its requirements
  - Reusability the architecture (re)uses existing assets
  - Buildability the architecture is realistic and suitable for its context

Brooks on “conceptual integrity”

I will contend that conceptual integrity is the most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas.

Fred Brooks, *The Mythical Man-Month*

Example: usability

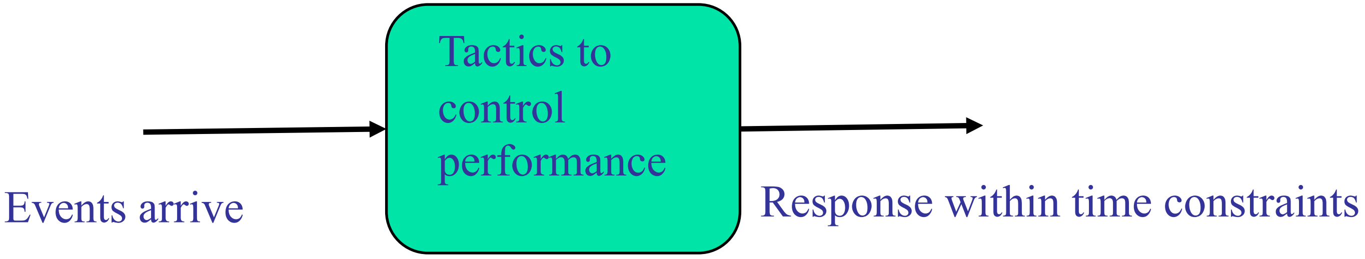
- Usability involves both architectural and nonarchitectural aspects.
- The nonarchitectural aspects include making the user interface clear and easy to use.
- Whether a system provides the user with the ability to cancel operations, to undo operations, or to re-use data previously entered is architectural, however. These requirements involve the cooperation of multiple elements.

Example: modifiability

- Modifiability is determined by how functionality is divided (architectural) and by coding techniques within a module (nonarchitectural
- a system is modifiable if changes impact the fewest possible number of distinct elements..

Example: performance

- Performance involves both architectural and nonarchitectural dependencies.
- Performance depends partially on
  - how much communication is necessary among components (architectural),
  - what functionality has been allocated to each component (architectural),
  - how shared resources are allocated (architectural),
  - the choice of algorithms to implement selected functionality (nonarchitectural),
  - how these algorithms are coded (nonarchitectural).


Events arrive
Tactics to control performance
Response within time constraints

Example: resilience

- Resilience is the property that ensures that a system well behaves under stress:
- it is the ability of a system to recover and, in some cases, transform itself from adversity
- Resilience testing ensures that applications perform well in real-life conditions.
- It is part of the non-functional sector of software testing that also includes testing compliance, endurance, load, recovery, etc

ISO 25010: software qualities

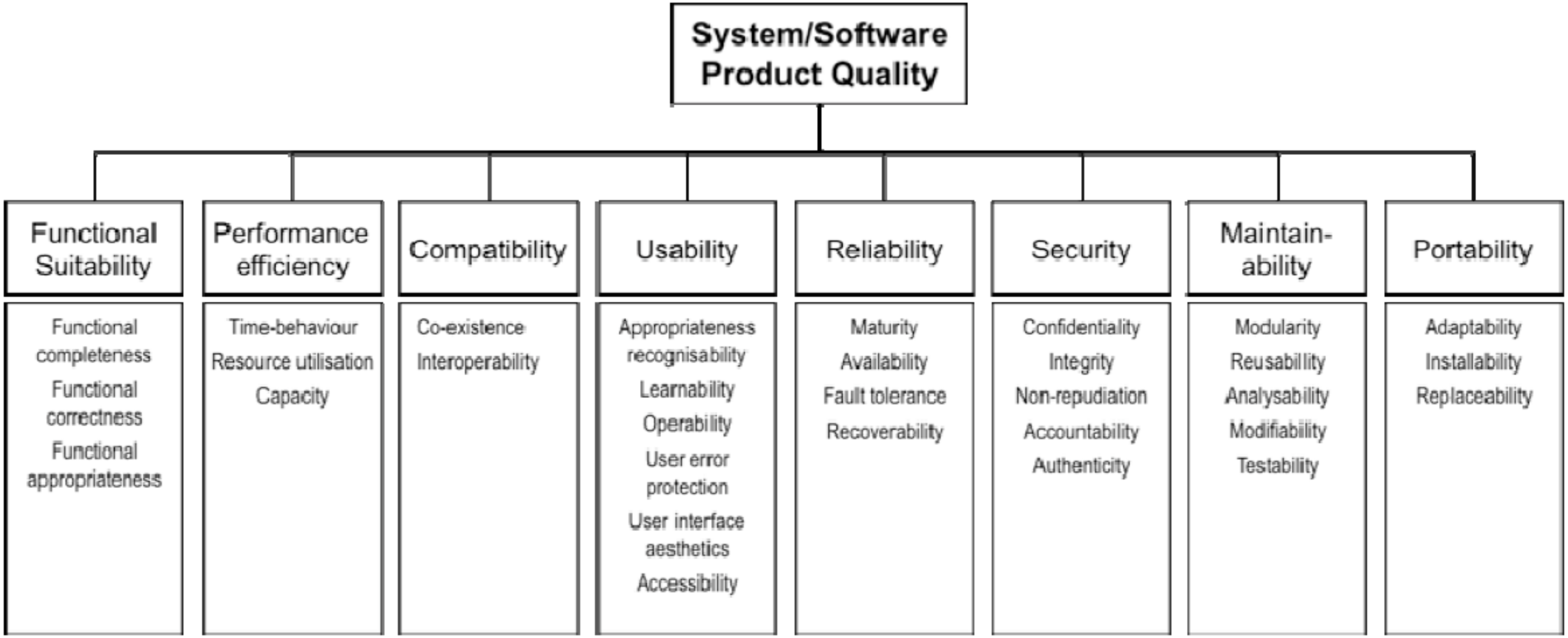
Quality in use

- Effectiveness
- Efficiency
- Satisfaction
- Safety
- Usability

Product quality

- Functional suitability
- Reliability
- Performance efficiency
- Operability
- Security
- Compatibility
- Maintainability
- Trasferability

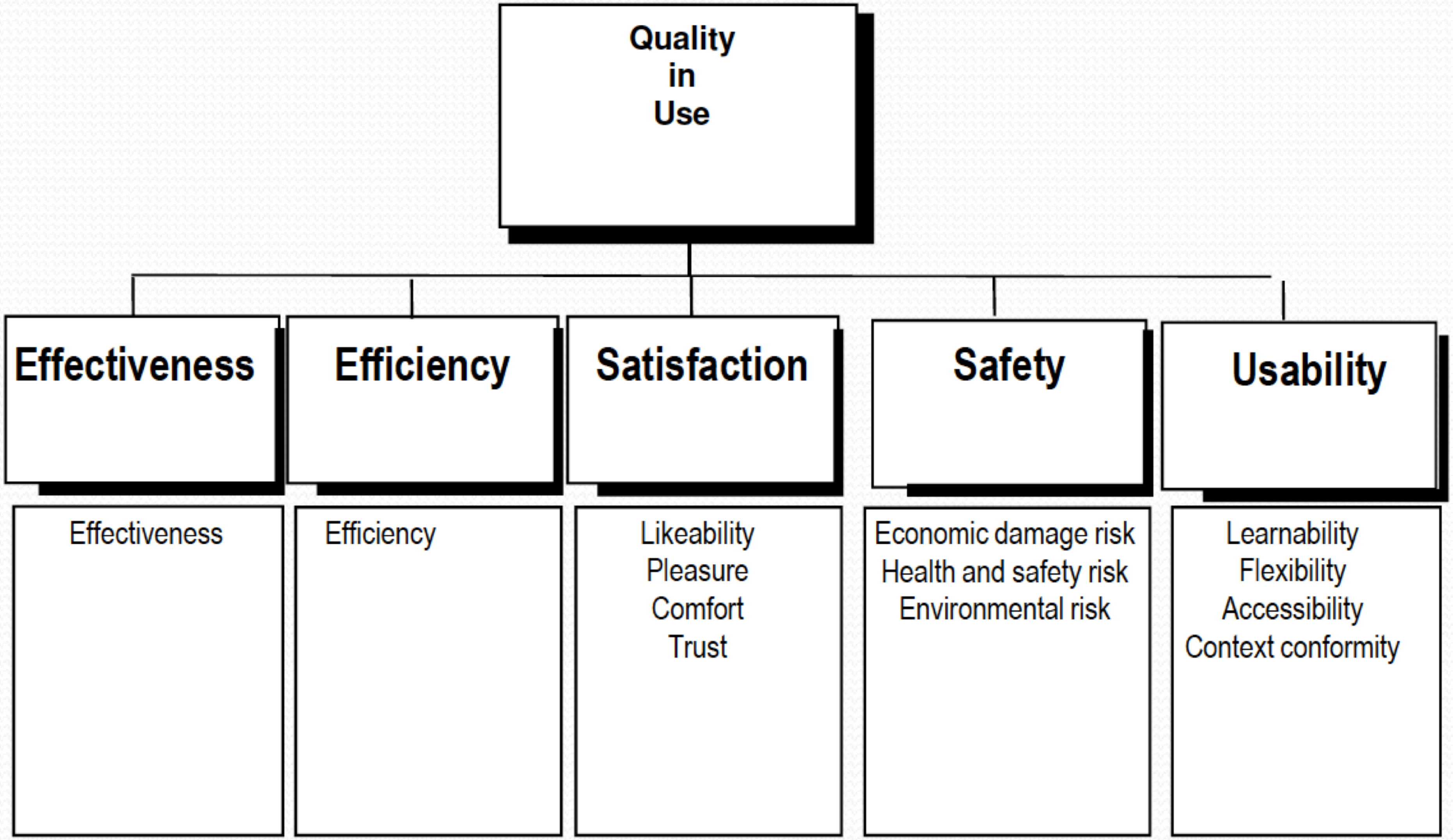
ISO25010: sw product qualities



System/Software Product Quality

- Functional Suitability
  - Functional completeness
  - Functional correctness
  - Functional appropriateness
- Performance efficiency
  - Time-behaviour
  - Resource utilisation
  - Capacity
- Compatibility
  - Co-existence
  - Interoperability
- Usability
  - Appropriateness recognisability
  - Learnability
  - Operability
  - User error protection
  - User interface aesthetics
  - Accessibility
- Reliability
  - Maturity
  - Availability
  - Fault tolerance
  - Recoverability
- Security
  - Confidentiality
  - Integrity
  - Non-repudiation
  - Accountability
  - Authenticity
- Maintainability
  - Modularity
  - Reusabillity
  - Analysability
  - Modifiability
  - Testability
- Portability
  - Adaptability
  - Installability
  - Replaceability

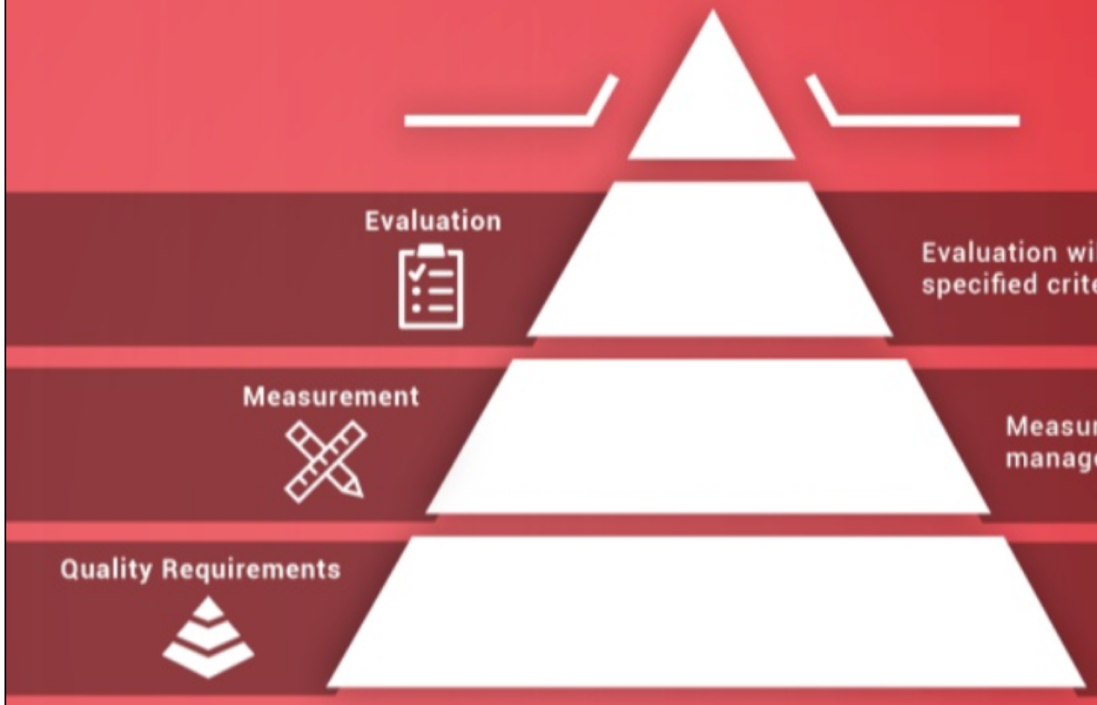
ISO25010: Quality in use



Quality in Use

Effectiveness	Efficiency	Satisfaction	Safety	Usability
Effectiveness	Efficiency	Likeability Pleasure Comfort Trust	Economic damage risk Health and safety risk Environmental risk	Learnability Flexibility Accessibility Context conformity

ISO/IEC 25000 Software Quality Requirements and Evaluation (SQuaRE)



Evaluation will help you determine to what extent has your product met its specified criteria.

Measurement will satisfy the needs of developers, acquirers, managers and other stakeholders.

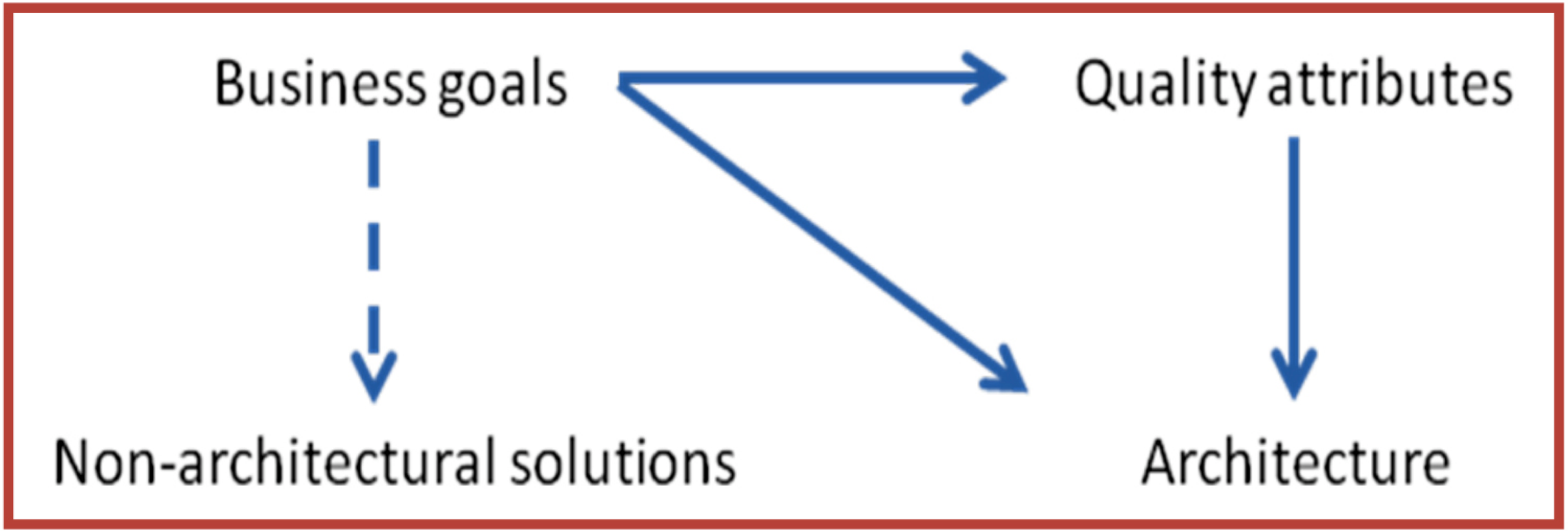
By defining Quality Requirements, Planning and Development you will increase the success of your project.

Activities during product development that can benefit from the use of the SQuaRE include:

- Identifying software and system requirements;
- Validating the comprehensiveness of a requirements definition;
- Identifying software and system design objectives;
- Identifying software and system testing objectives;
- Identifying quality control criteria as part of quality assurance;
- Identifying acceptance criteria for a software product and/or software-intensive computer system;
- Establishing measures of quality characteristics in support of these activities

Qualities and trade-offs

- The qualities are all good
- The value of a quality is project specific
- The qualities are not independent



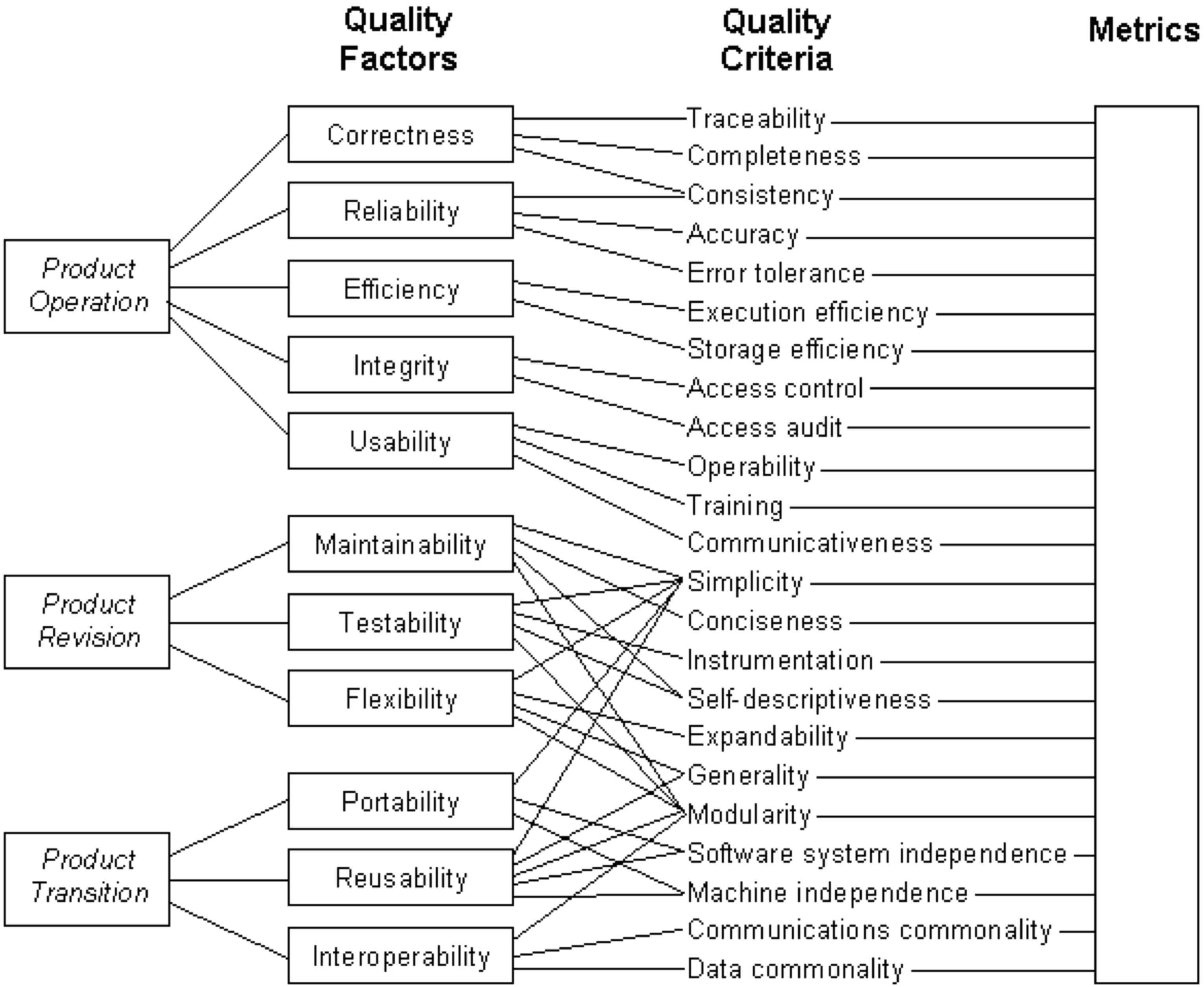
Quality attributes: in use

- **Safety** freedom from risk: absence of catastrophic consequences on the users or the environment
- **Usability** is how easy it is for the user to accomplish tasks and what support the system provides for the user to accomplish this. Dimensions:
  - Learning system features
  - Using the system efficiently
  - Minimizing the impact of errors
  - Adapting the system to the user’s needs
  - Increasing confidence and satisfaction

Quality attributes: product

- **Modifiability** is about the cost of change, both in time and money. Performance is about timeliness. Events occur and the system must respond in a timely fashion.
- **Testability** refers to the ease with which the software can be made to demonstrate its faults or lack thereof. To be testable the system must control inputs and be able to observe outputs
- **Maintainability** is the ease with which a product can be maintained in order to isolate and correct defects, prevent unexpected breakdowns, meet new requirements
- **Availability** is concerned with system failure and duration of system failures. System failure means unreadiness for correct service, when the system does not provide the service for which it was intended
- **Reliability**: the ability of a system or component to function under stated conditions for a specified period of time (=continuity of correct service)
- **Dependability**: availability + reliability + maintainability

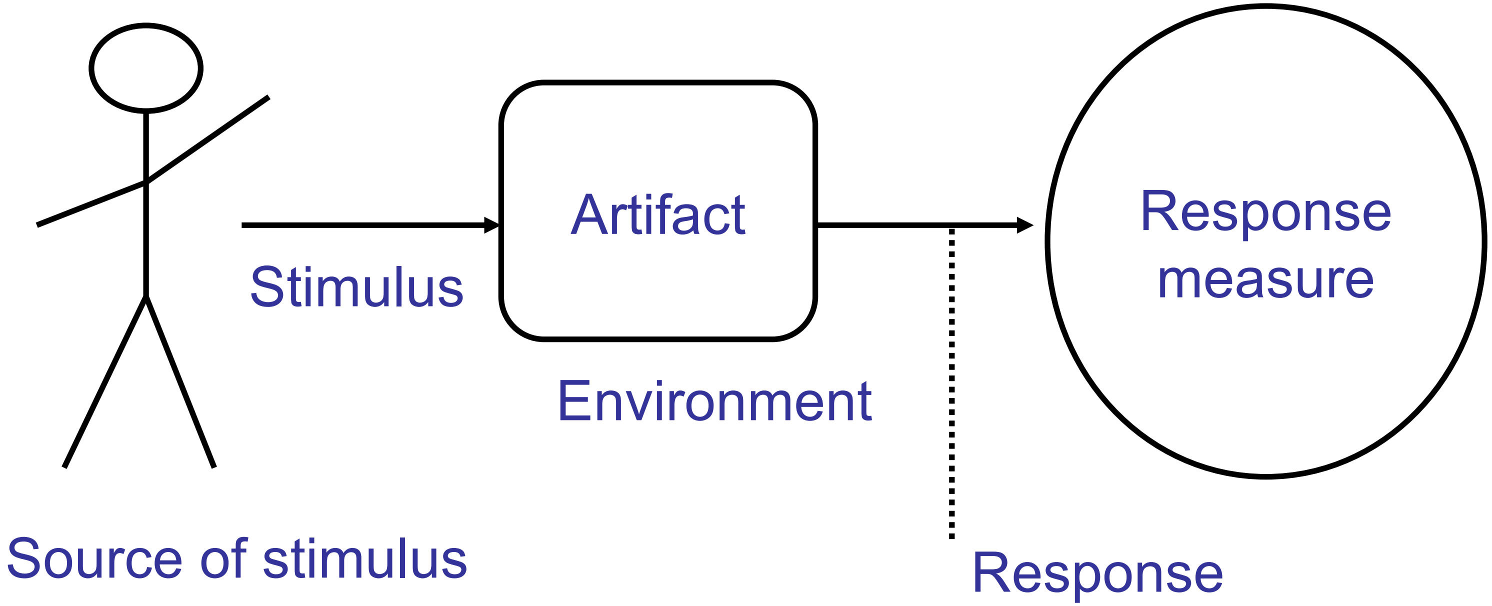
Quality and metrics



msritse2012.wordpress.com/2013/01/27/quality-models-in-software-engineering/

Quality attributes shape the architecture

- The critical choices made during architectural design determine the ways the system meets the driving quality attribute goals
- A good way to discuss and prioritize quality attribute requirements is a set of *scenarios*



Structuring a quality attribute scenario

i. Source of stimulus
ii. Stimulus
iii. Environment
iv. Artifact
v. Response
vi. Response measure

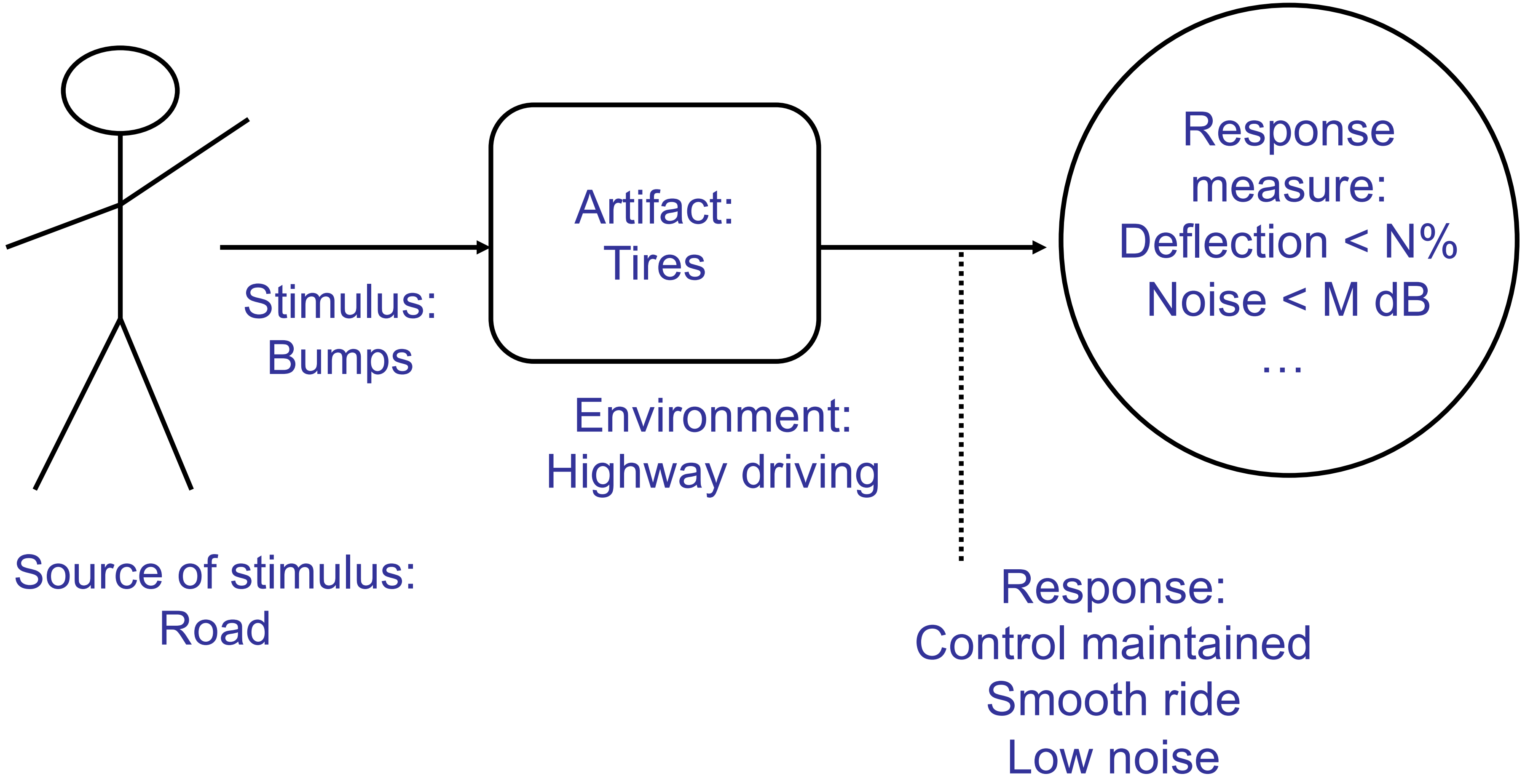
In the *environment*, the *source* throws the *stimulus* and hits the system in the *artifact; we get a response than can be measured*

How to generate a scenario

- A scenario is a system independent specification of a requirement with a measurable quality attribute
- This table is a framework to structure scenarios

Elements	Short description
Stimulus	A condition to be considered when it arrives at a system
Response	The activity undertaken at the arrival of the stimulus
Source of stimulus	An entity that generates the stimulus (human, external system, sensor, etc.)
Environment	A system's condition when a stimulus occurs
Stimulated artifact	Some artifact that is stimulated; may be the whole system or part of it
Response measure	The response to the stimulus should be measurable someway so that the quality requirement can be tested

Example from cars



Source of stimulus:
Road

Stimulus:
Bumps

Artifact:
Tires

Environment:
Highway driving

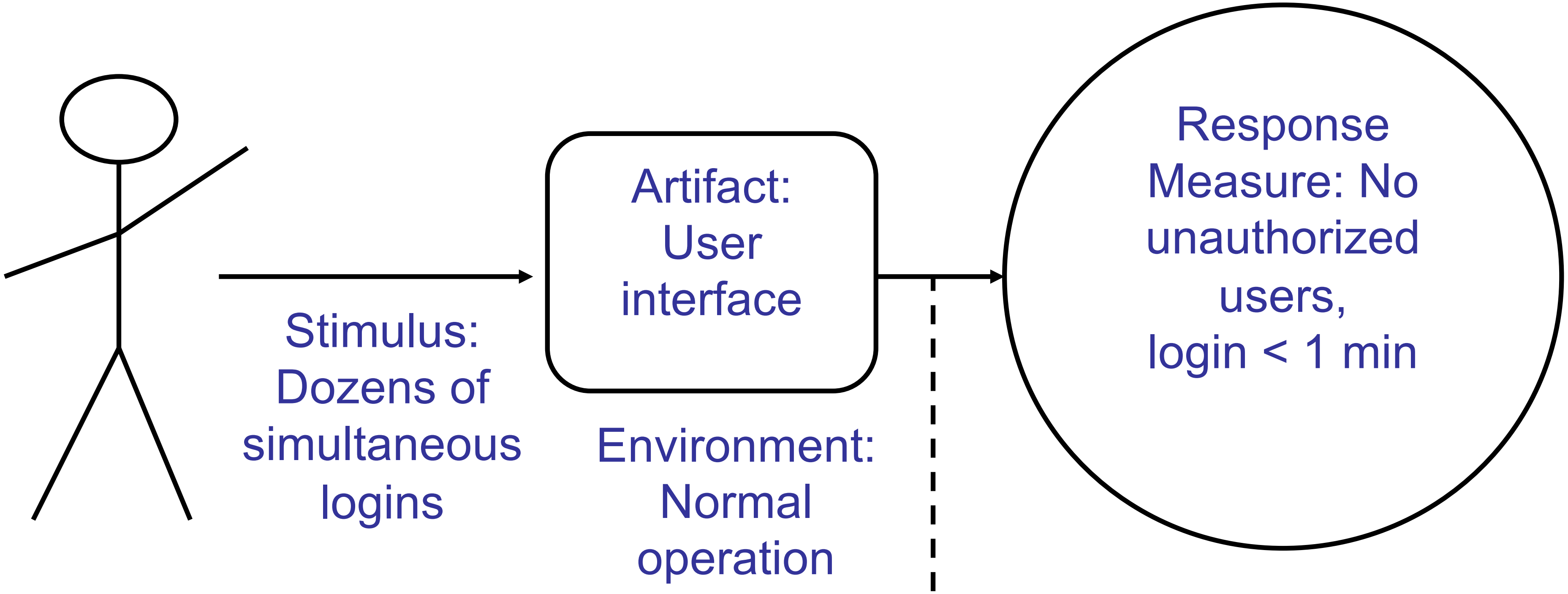
Response:
Control maintained
Smooth ride
Low noise

Response measure:
Deflection < N%
Noise < M dB
…

Suggestions

- One stimulus per scenario
- One environment per scenario
- One artifact per scenario
- Multiple response measures are OK

Example from software: security



Source of stimulus: Users

Stimulus: Dozens of simultaneous logins

Artifact: User interface

Environment: Normal operation

Response: Security maintained Acceptable delays

Response Measure: No unauthorized users, login < 1 min

Example from software: modifiability

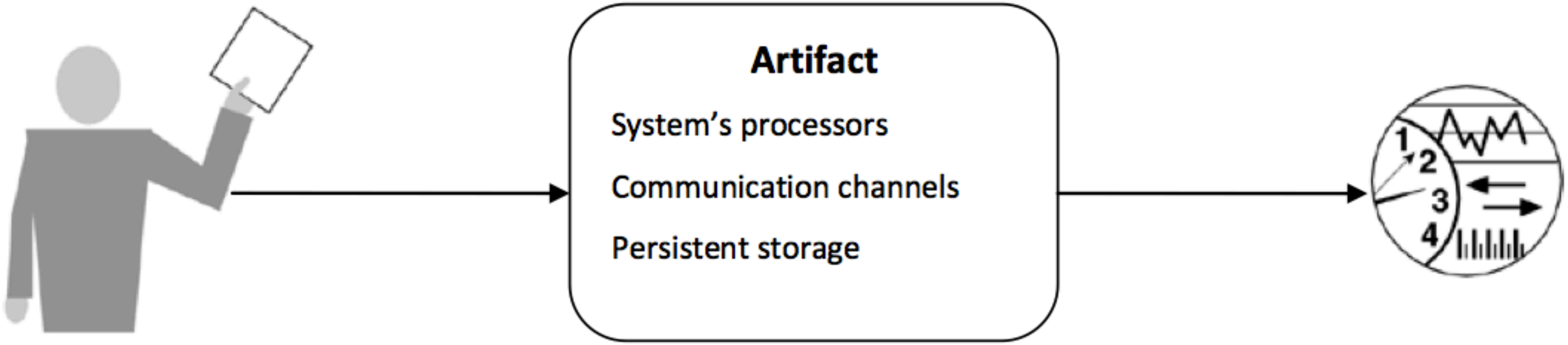
Scenario Part	**Possible Values**
Source	End user, developer, system admin
Stimulus	Add/delete/modify/vary functionality, quality attribute or capacity
Artifact	User interface, platform, environment, other system
Environment	Runtime, compile time, build time, design time, setup, configuration
Response	Places to be modified without other effect, test change, deployment
Response Measure	Cost of change in number of elements, effort, duration, money. Extent the change affects other functionality or quality attributes

Qualities must be testable

- To be effective, quality attribute scenarios must be *testable* (just like any other requirement)
- Therefore, the
  - Stimulus
  - Artifact
  - Environment
  - Response measure(s)

  must be clear and specific

General availability scenario



Artifact

System’s processors

Communication channels

Persistent storage

Source

Internal to system

External to system

Stimulus

Crash

Omission

Timing

No response

Incorrect response

Environment

Normal operation

Startup

Shutdown

Repair mode

Degraded (failsafe) mode

Overloaded operation

Response

Prevent the failure

Log the failure

Notify users / operators

Disable source of failure

Temporarily unavailable

Continue (normal / degraded)

Measure

Time interval available

Availability %

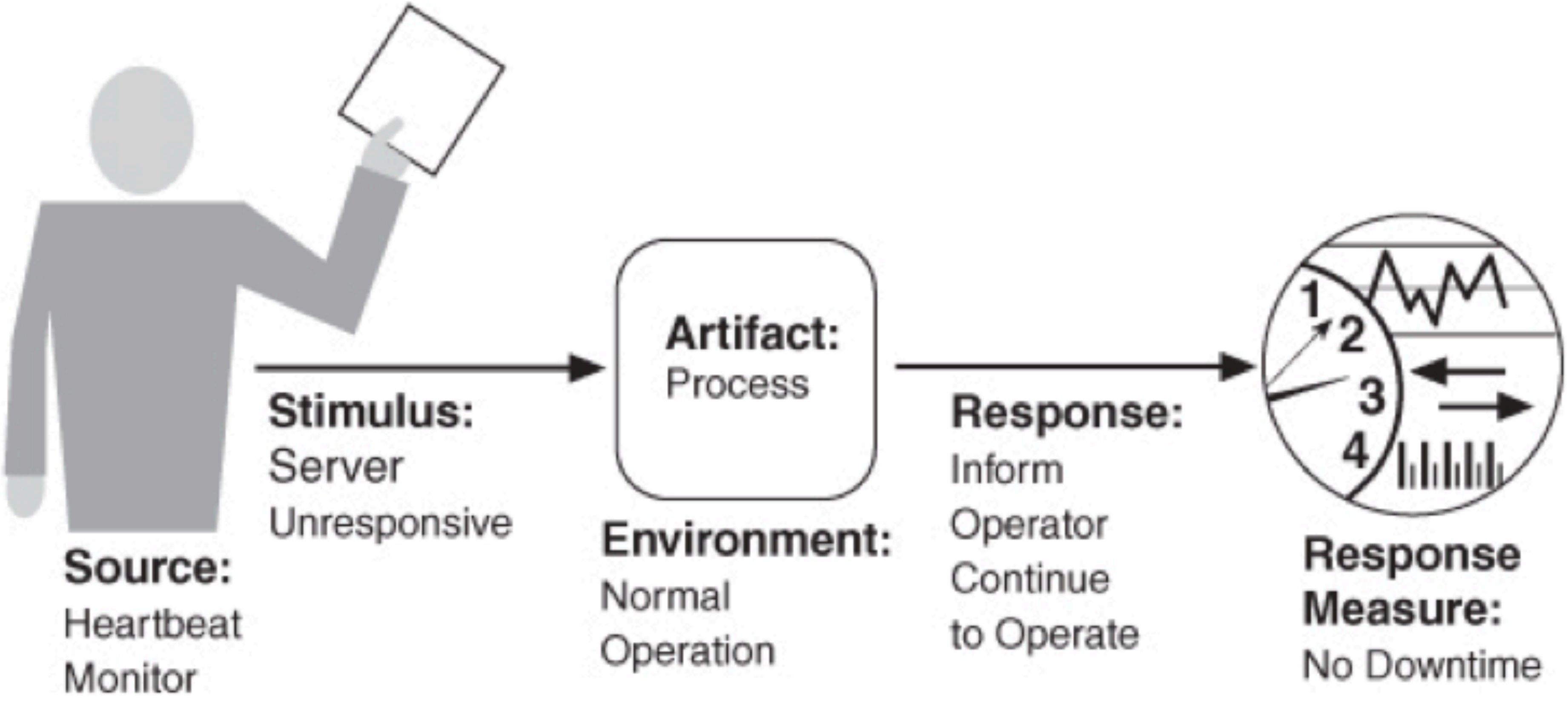
Detection time

Repair time

Degraded mode time interval

Unavailability time interval

Sample availability scenario



Source: Heartbeat Monitor

Stimulus: Server Unresponsive

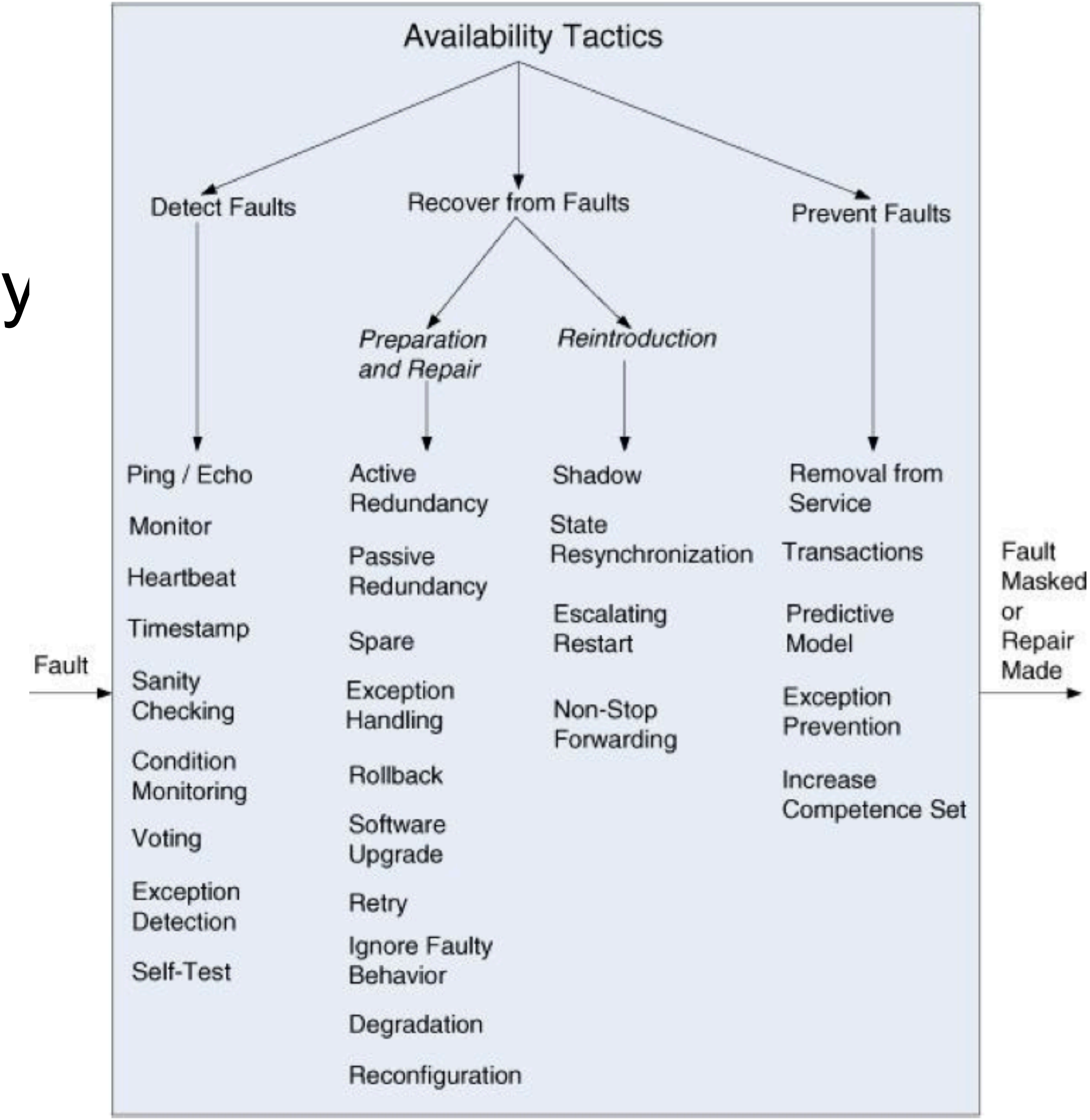
Artifact: Process

Environment: Normal Operation

Response: Inform Operator Continue to Operate

Response Measure: No Downtime

Availability tactics



Availability Tactics

Detect Faults

- Ping / Echo
- Monitor
- Heartbeat
- Timestamp
- Sanity Checking
- Condition Monitoring
- Voting
- Exception Detection
- Self-Test

Recover from Faults

Preparation and Repair

- Active Redundancy
- Passive Redundancy
- Spare
- Exception Handling
- Rollback
- Software Upgrade
- Retry
- Ignore Faulty Behavior
- Degradation
- Reconfiguration

Reintroduction

- Shadow
- State Resynchronization
- Escalating Restart
- Non-Stop Forwarding

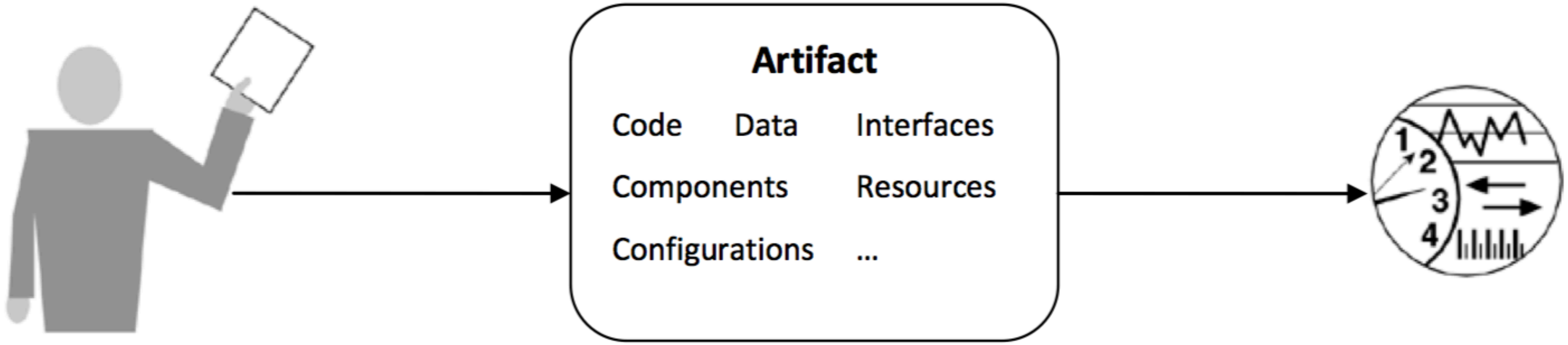
Prevent Faults

- Removal from Service
- Transactions
- Predictive Model
- Exception Prevention
- Increase Competence Set

Fault

Fault Masked or Repair Made

General modifiability scenario



Artifact

Code Data Interfaces

Components Resources

Configurations …

Source

End-user

Developer

System-administrator

Stimulus

Add / delete / modify functionality, quality attribute, capacity or technology

Environment

Runtime

Compile time

Build time

Initiation time

Design time

Response

Make modification

Test modification

Deploy modification

Measure

Cost in effort

Cost in money

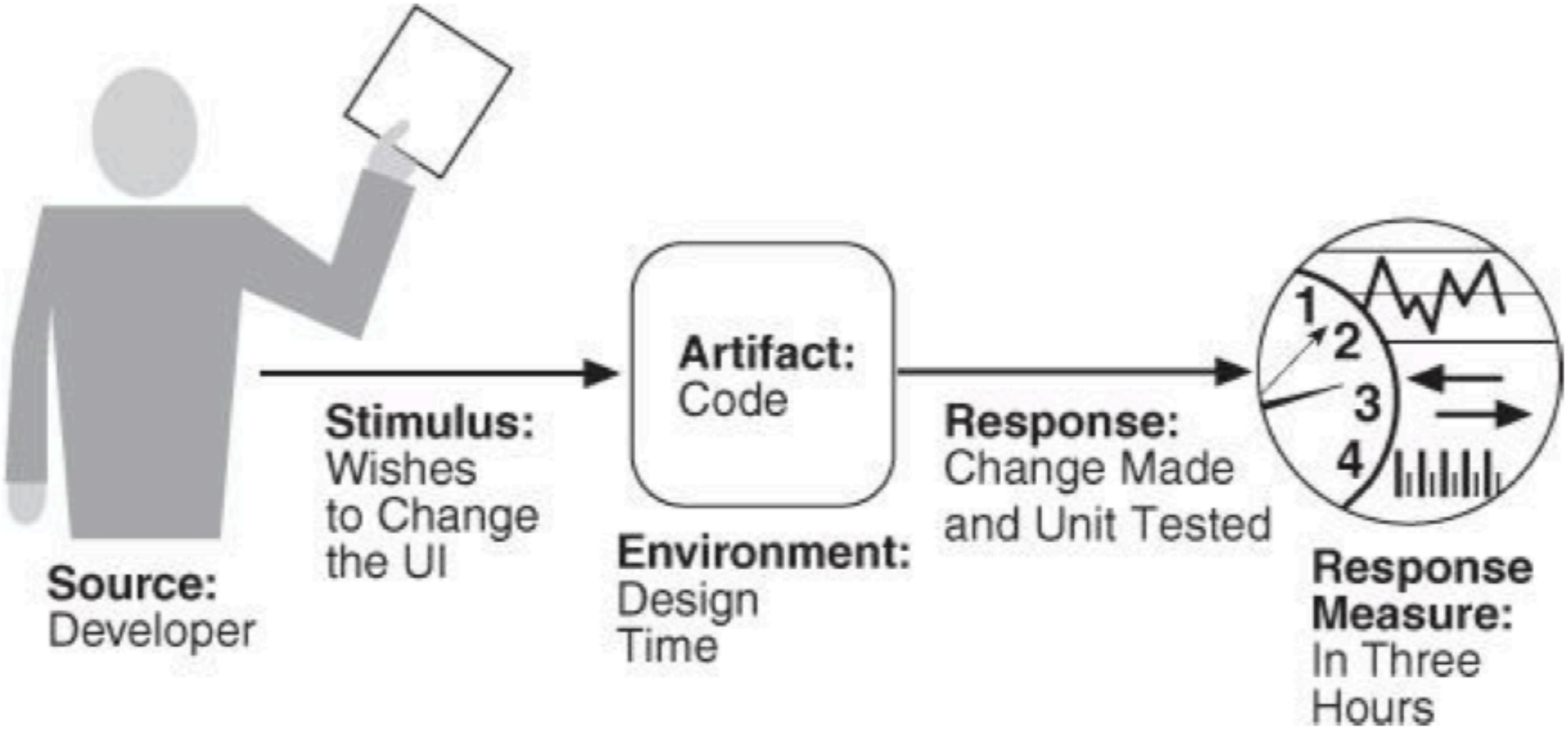
Cost in time

Cost in number, size, complexity of affected artifacts

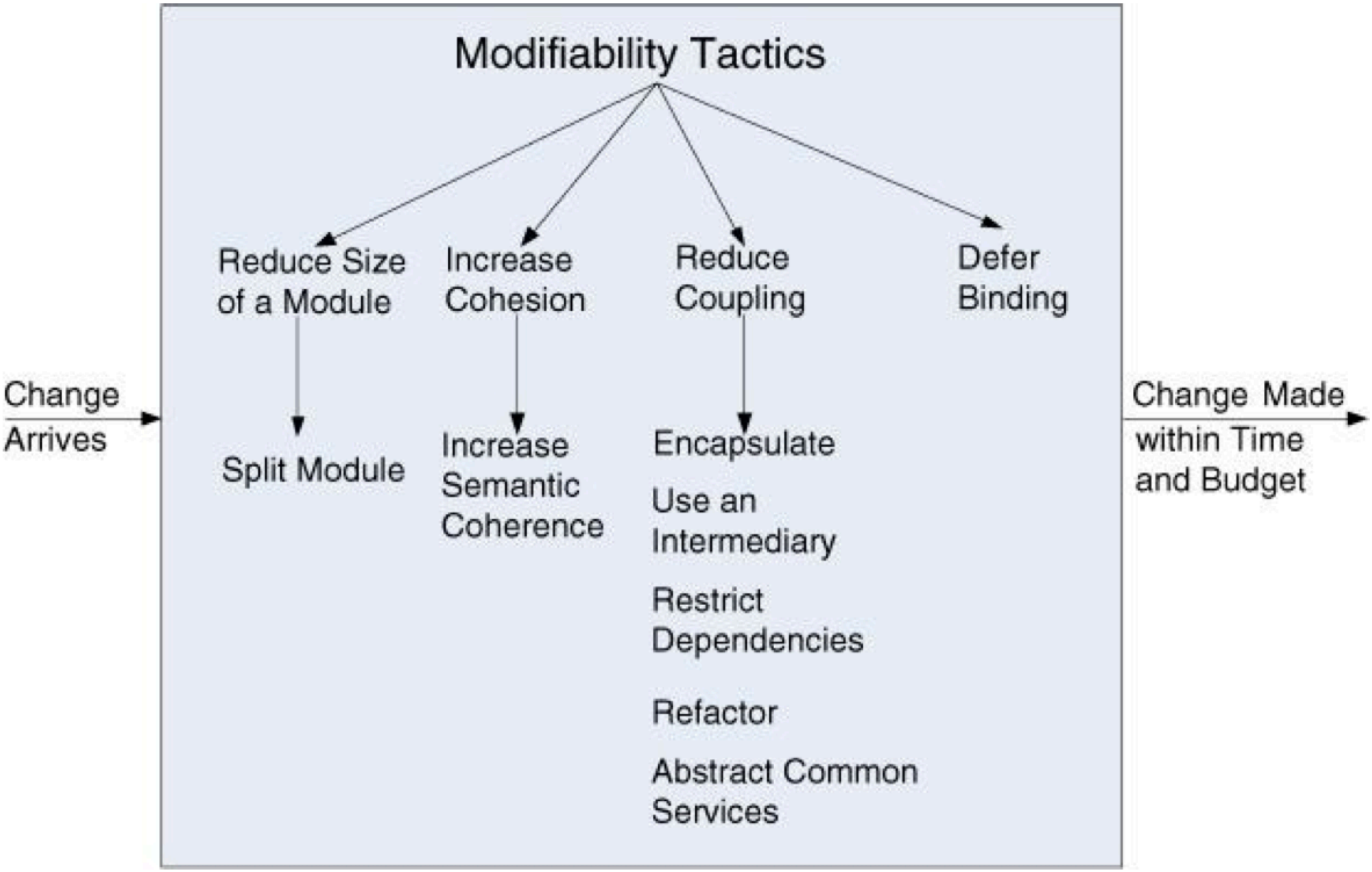
Extent affects other system functions or qualities

New defects introduced

Sample modifiability scenario



Modifiability tactics



Modifiability Tactics

Change Arrives

Reduce Size of a Module
- Split Module

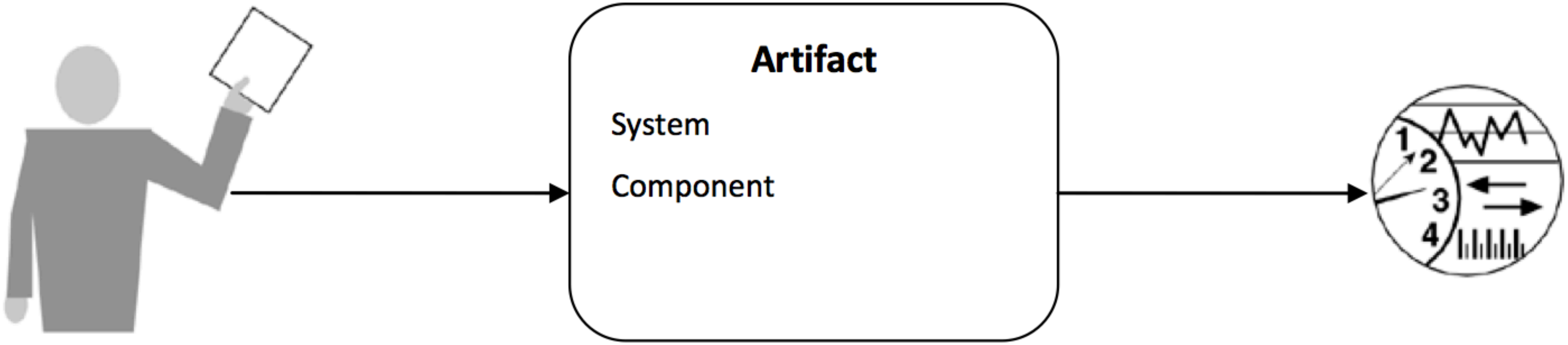
Increase Cohesion
- Increase Semantic Coherence

Reduce Coupling
- Encapsulate
- Use an Intermediary
- Restrict Dependencies
- Refactor
- Abstract Common Services

Defer Binding

Change Made within Time and Budget

General performance scenario



Artifact

System

Component

Source

Internal to the system

External to the system

Stimulus

Periodic events

Sporadic events

Bursty events

Stochastic events

Environment

Normal mode

Overload mode

Reduced capacity mode

Emergency mode

Peak mode

Response

Process events

Change level of service

Measure

Latency

Deadline

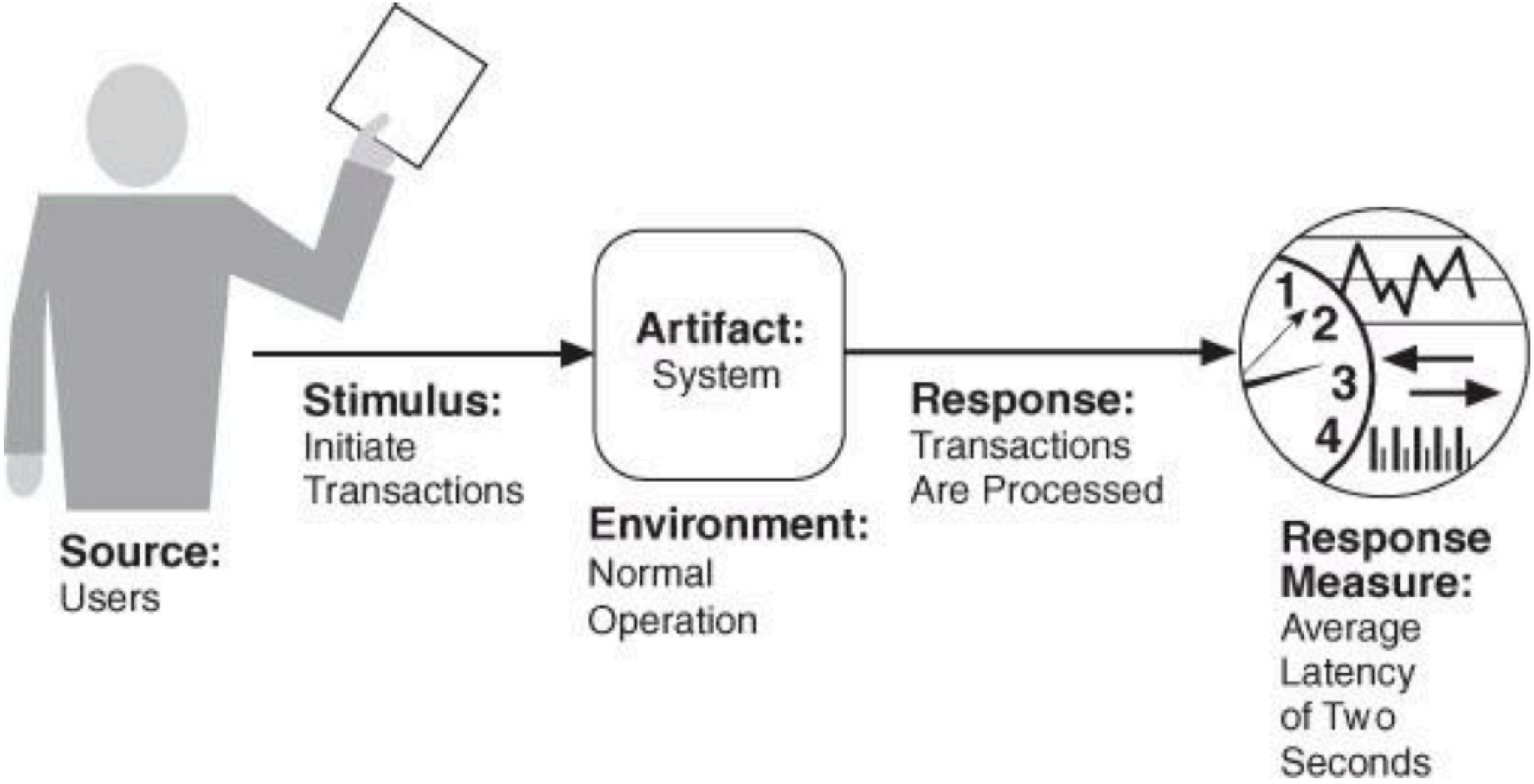
Throughput

Jitter

Miss rate

Data loss

Sample performance scenario



Source:
Users

Stimulus:
Initiate Transactions

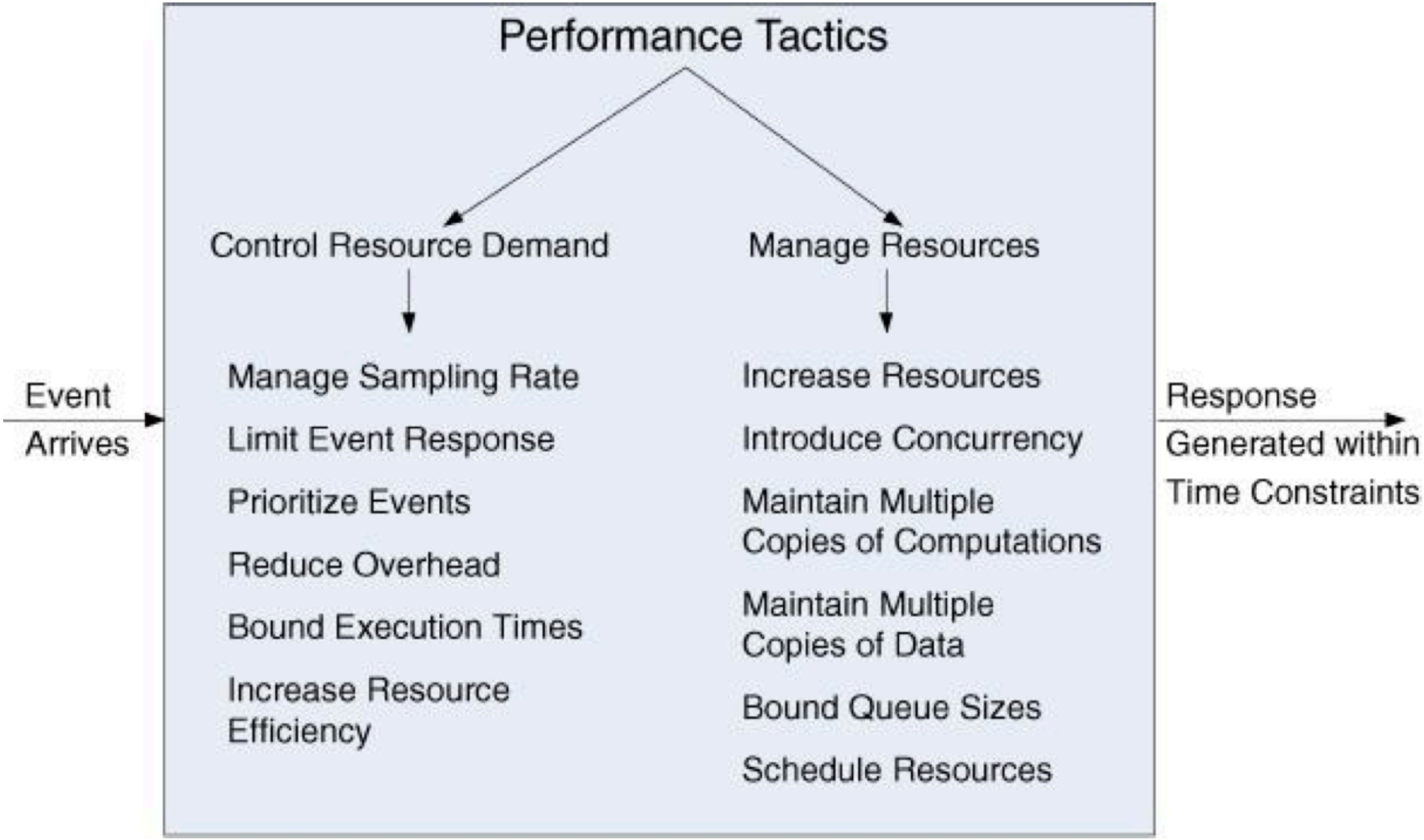
Artifact:
System

Environment:
Normal Operation

Response:
Transactions Are Processed

Response Measure:
Average Latency of Two Seconds

Performance tactics



Event Arrives

Performance Tactics

Control Resource Demand

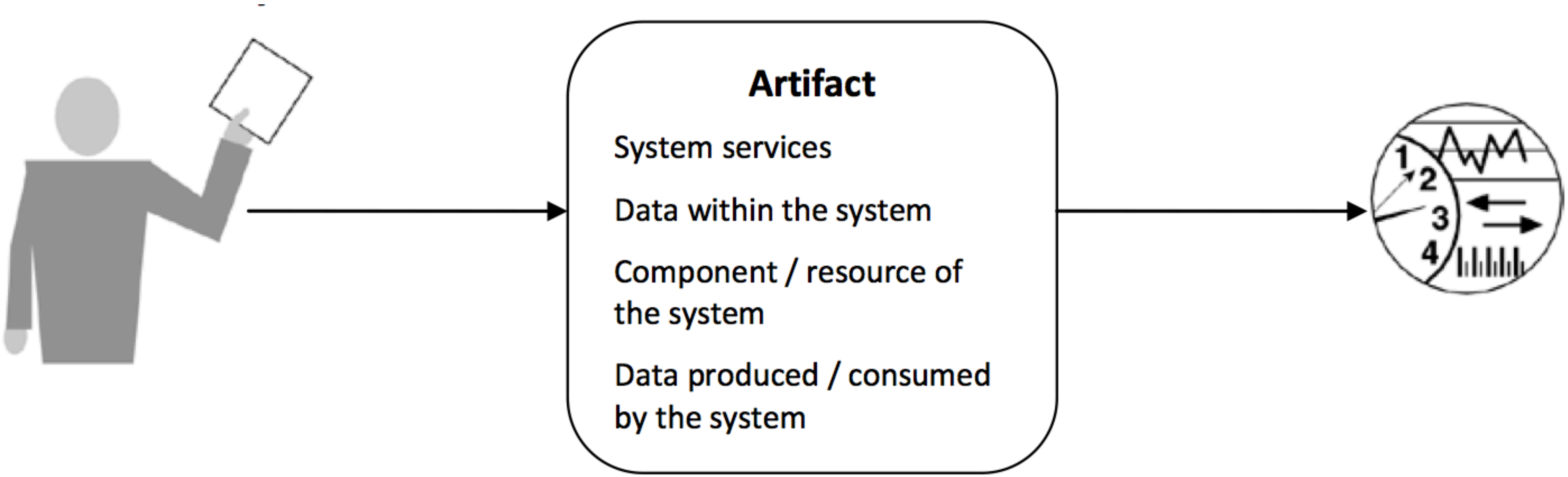
- Manage Sampling Rate
- Limit Event Response
- Prioritize Events
- Reduce Overhead
- Bound Execution Times
- Increase Resource Efficiency

Manage Resources

- Increase Resources
- Introduce Concurrency
- Maintain Multiple Copies of Computations
- Maintain Multiple Copies of Data
- Bound Queue Sizes
- Schedule Resources

Response Generated within Time Constraints

General security scenario



Artifact

System services

Data within the system

Component / resource of the system

Data produced / consumed by the system

Source

Identified user

Unknown user

Hacker from outside the organization

Hacker from inside the organization

Stimulus

Attempt to display data

Attempt to modify data

Attempt to delete data

Access system services

Change system's behavior

Reduce availability

Environment

Normal mode

Overload mode

Reduced capacity mode

Emergency mode

Peak mode

Response

Process events

Change level of service

Measure

Latency

Deadline

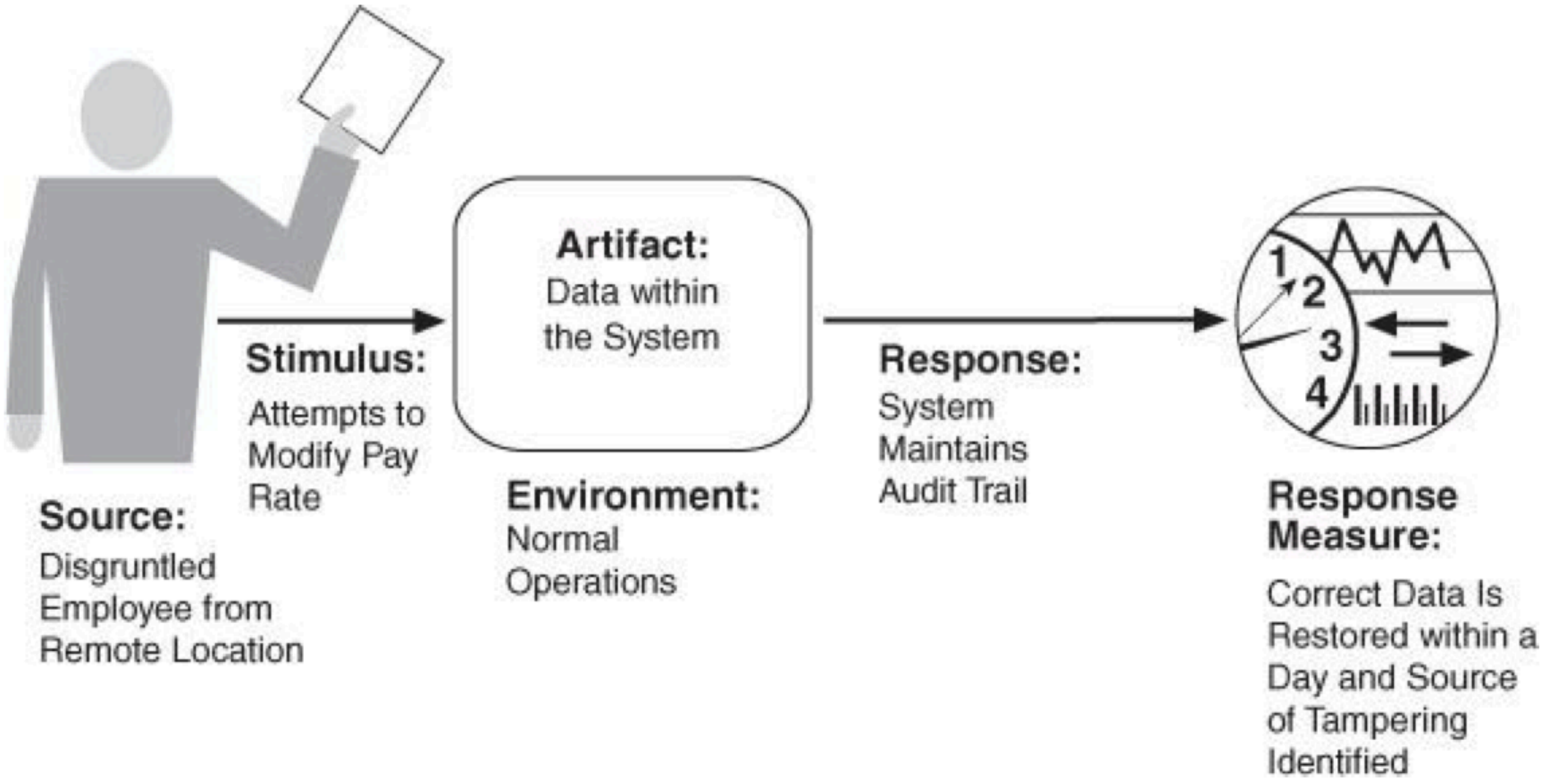
Throughput

Jitter

Miss rate

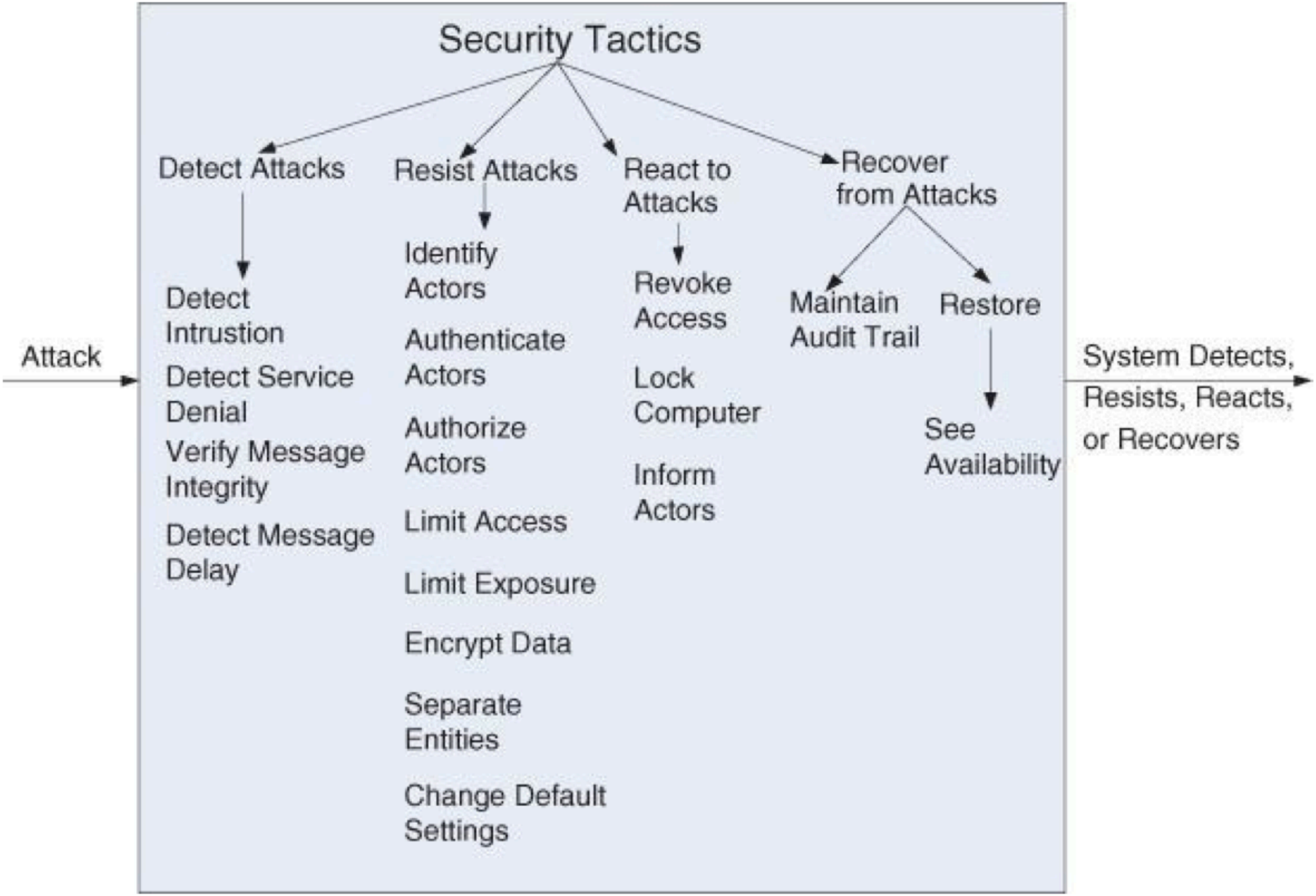
Data loss

Sample security scenario

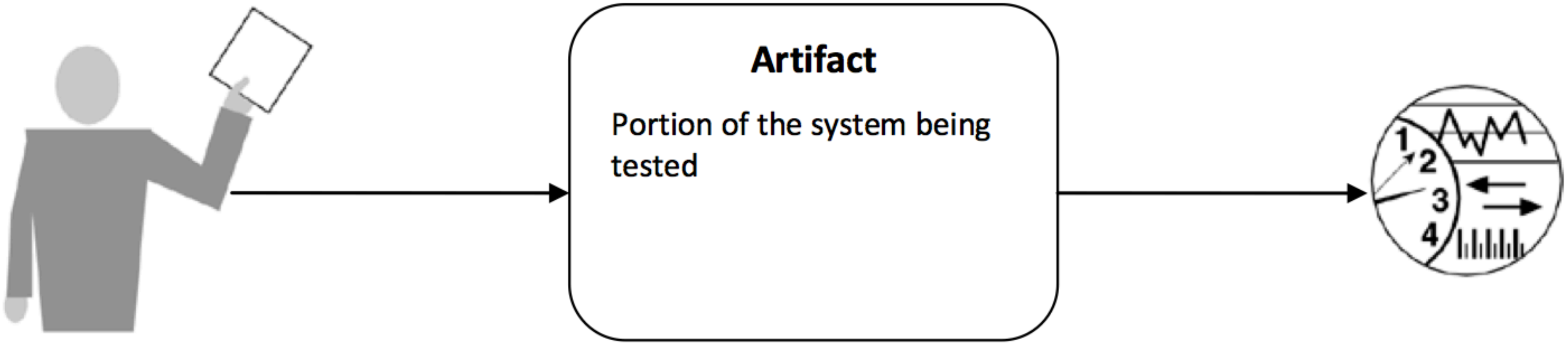


Source:
Disgruntled Employee from Remote Location
Stimulus:
Attempts to Modify Pay Rate
Artifact:
Data within the System
Environment:
Normal Operations
Response:
System Maintains Audit Trail
Response Measure:
Correct Data Is Restored within a Day and Source of Tampering Identified
1
2
3
4

Security tactics



General testability scenario



Artifact

Portion of the system being tested

Source

Unit tester

Integration tester

System tester

Acceptance tester

End user

Automated testing tools

Stimulus

Execution of tests due to completion of code increment

Environment

Design time

Development time

Compile time

Integration time

Deployment time

Run time

Response

Execute test suite & capture results

Capture cause of fault

Control & monitor state of the system

Measure

Effort to find fault

Effort to achieve coverage %

Probability of fault being revealed by next test

Time to perform tests

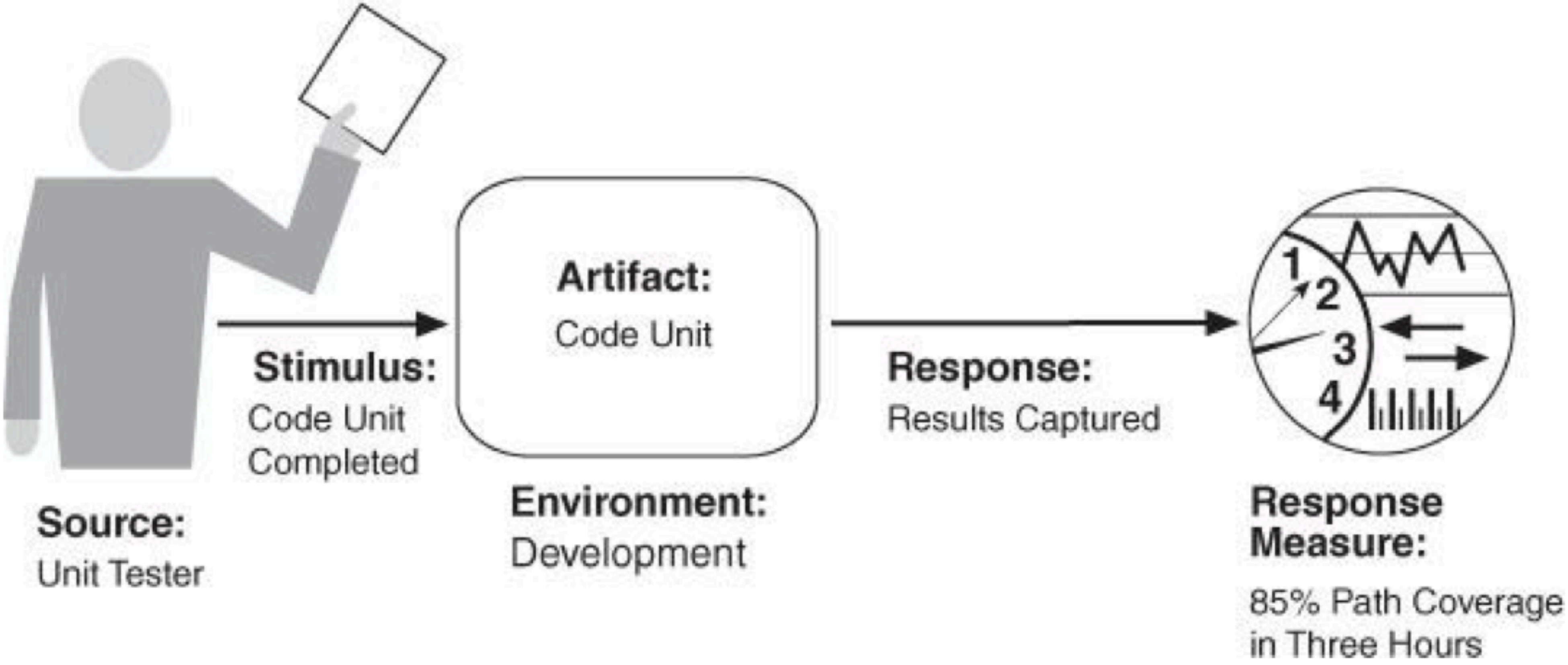
Effort to detect faults

Length of longest dependency chain

Time to prepare test environment

Reduction in risk exposure

Sample testability scenario



Source: Unit Tester

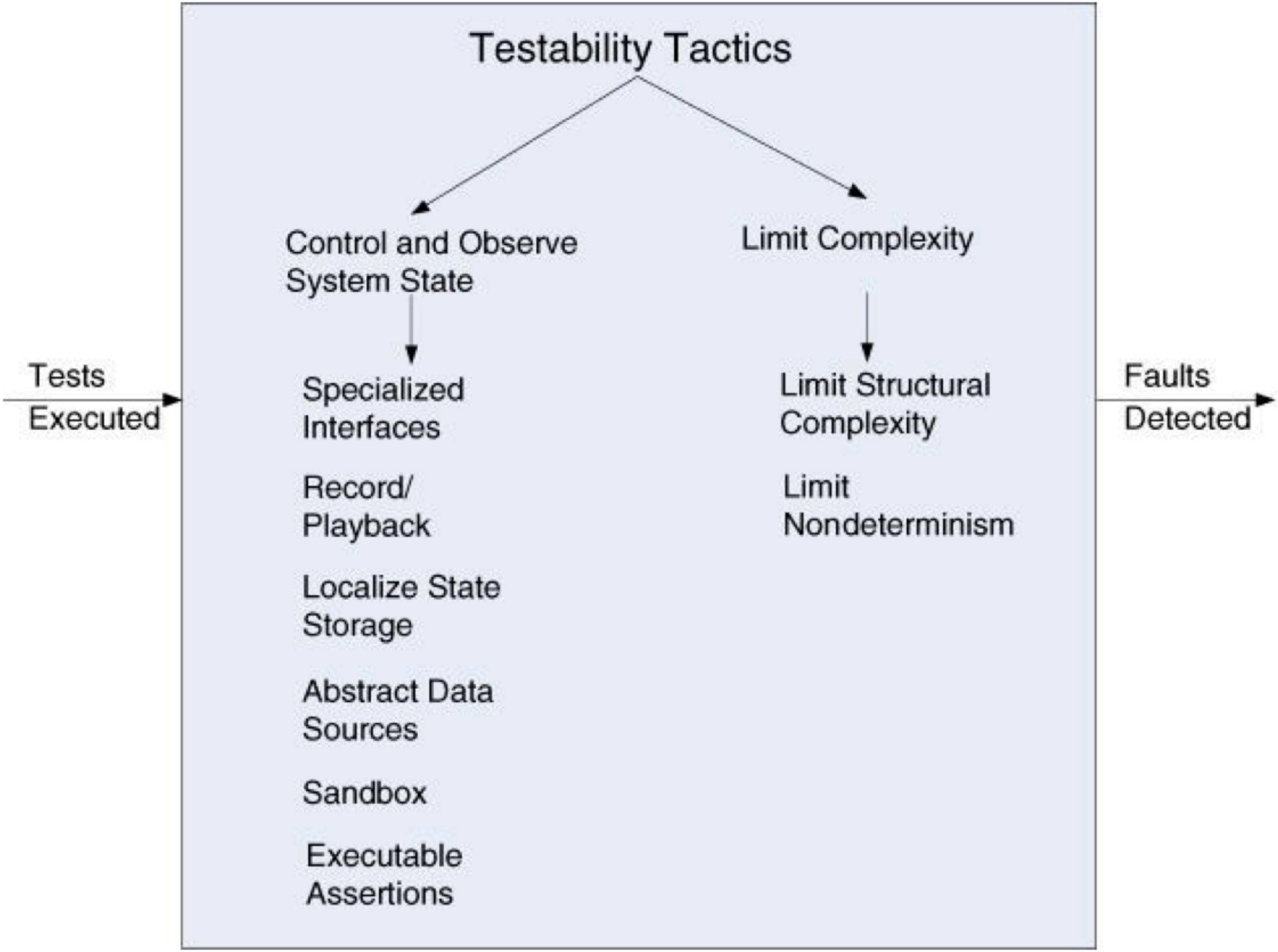
Stimulus: Code Unit Completed

Artifact: Code Unit

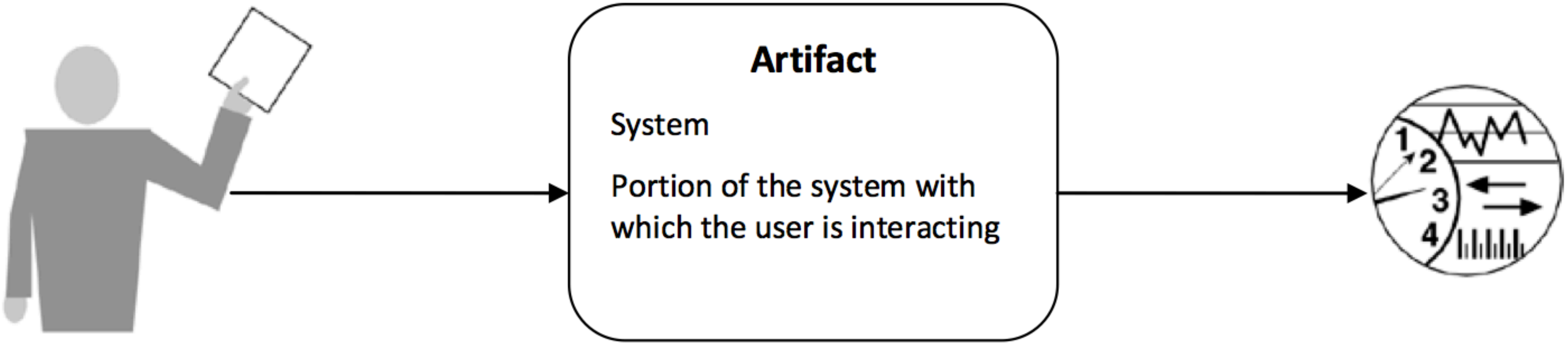
Environment: Development

Response: Results Captured

Response Measure: 85% Path Coverage in Three Hours


Testability Tactics
Control and Observe System State
Limit Complexity
Tests Executed
Specialized Interfaces
Record/ Playback
Localize State Storage
Abstract Data Sources
Sandbox
Executable Assertions
Limit Structural Complexity
Limit Nondeterminism
Faults Detected

General usability scenario



Artifact

System

Portion of the system with which the user is interacting

Source

End user (possibly special role)

Stimulus

Use the system efficiently

Learn to use the system

Minimize impact of errors

Adapt the system

Configure the system

Environment

Runtime

Configuration time

Response

Provide features needed

Anticipate the user's needs

Measure

Task time

Number of errors

Number of tasks accomplished

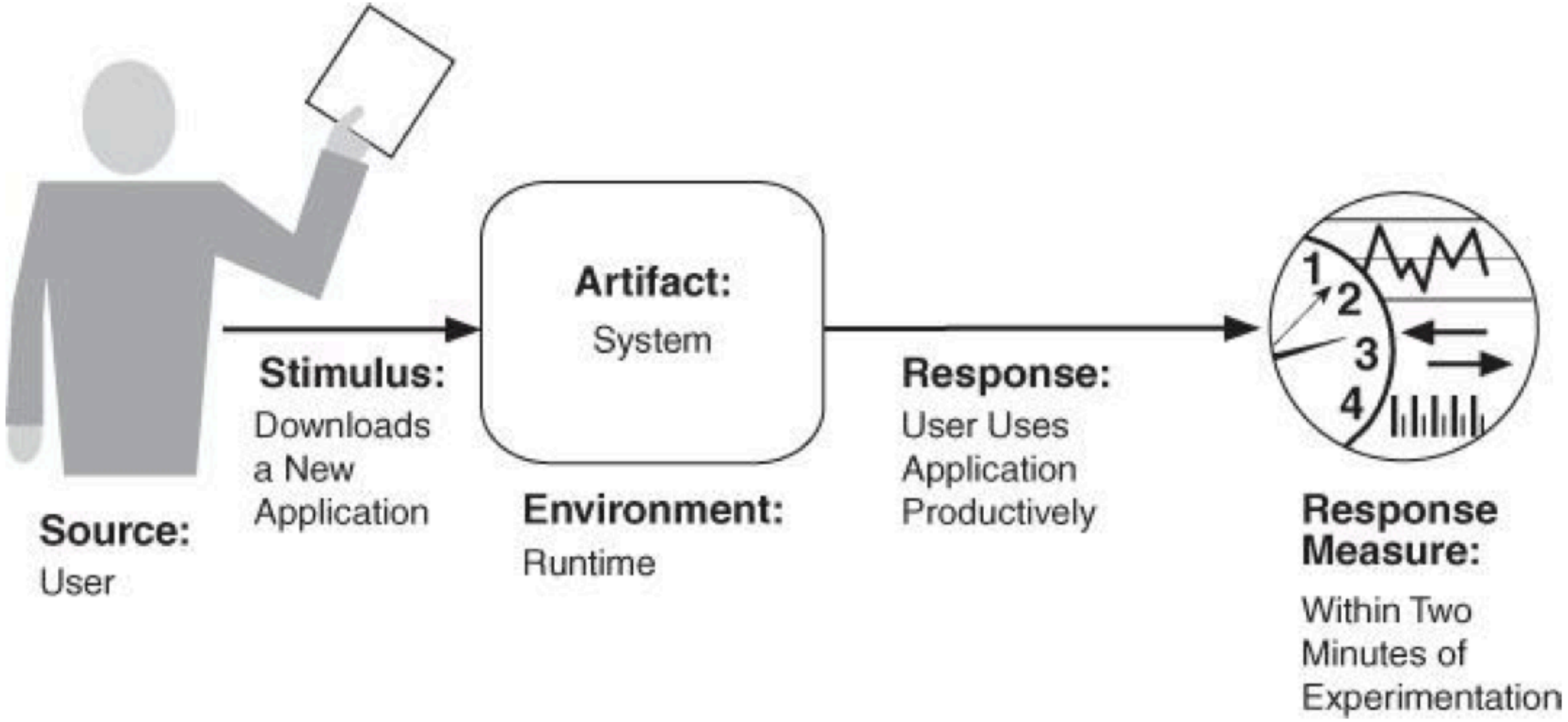
User satisfaction

Gain of user knowledge

Ratio of successful operations to total operations

Amount of time / data lost when error occurs

Sample usability scenario



Source: User

Stimulus: Downloads a New Application

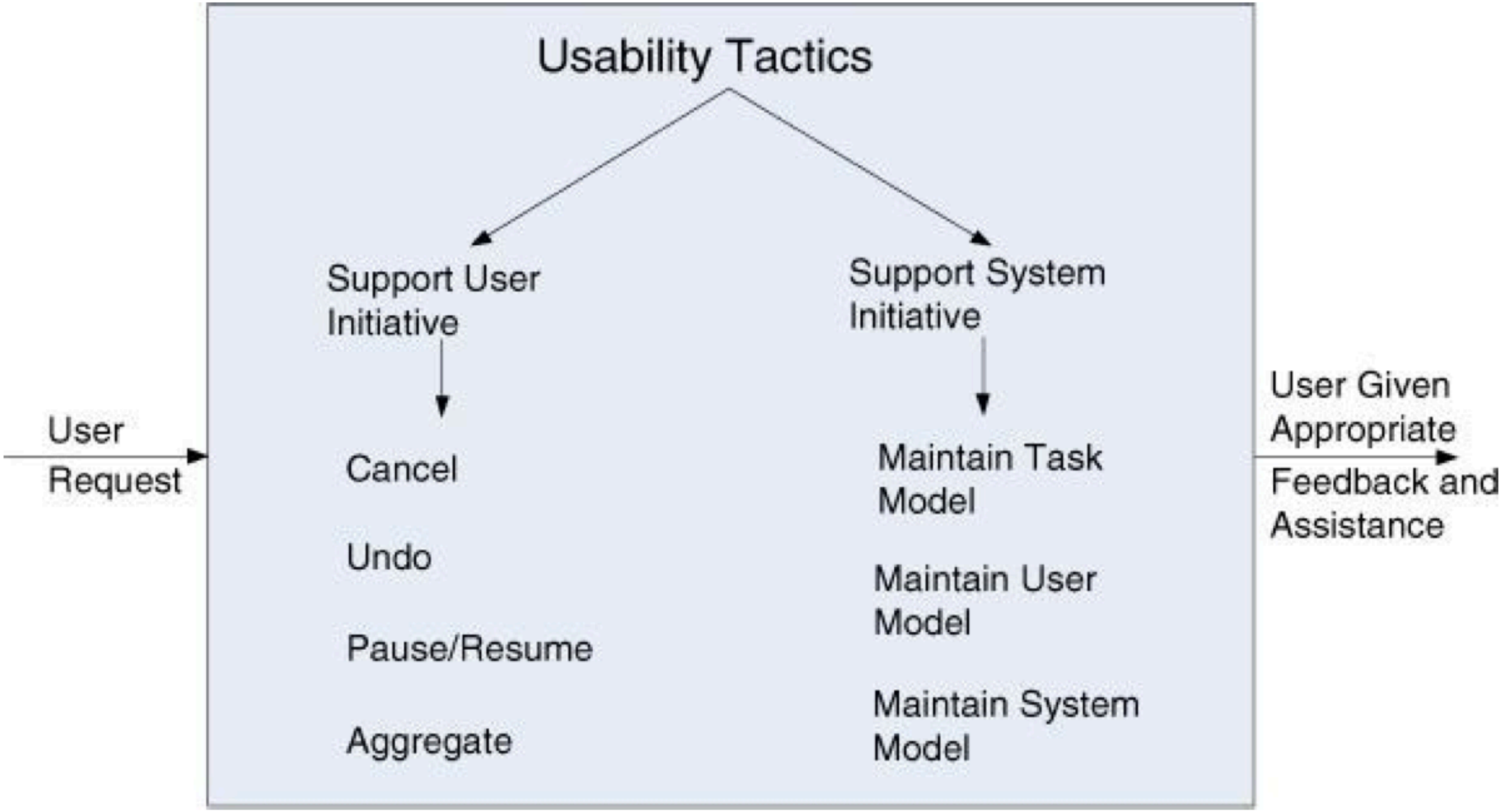
Artifact: System

Environment: Runtime

Response: User Uses Application Productively

Response Measure: Within Two Minutes of Experimentation

Usability tactics



References

- ISO/IEC 25010:2011 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models
- Cervantes & Kazman, Designing sw architectures, AW 2016
- Bass, *Software architecture in practice*, 3° ed
  http://www.ece.ubc.ca/~matei/EECE417/BASS/
- Babar, *Agile Software Architecture*, MK 2014

- http://www.spin.org.za/wp-content/uploads/2011/01/SPIN-21-QAS.pdf
- http://sa.inceptum.eu/sites/sa.inceptum.eu/files/Content/Quality%20Attribute%20Generic%20Scenarios-2.pdf
- Stoermer, “Moving Towards Quality Attribute Driven Software Architecture Reconstruction”, http://www.cs.vu.nl/~x/square/qadsar.pdf

Questions?