Introduction to software architecture

Denis Conan

Septembre 2021

Foreword

- The content of these slides is extracted from the following references:
  - L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, 3rd Edition*. Addison-Wesley, 2012.
  - P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architecture: Views and Beyond, 2nd Edition*. Addison-Wesley, 2011.
  - P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.
  - EIT Digital, "*Software Architecture for the Internet of Things*", Coursera MOOC, 2015
  - A. Sunyaev. *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies*. Springer, 2020.

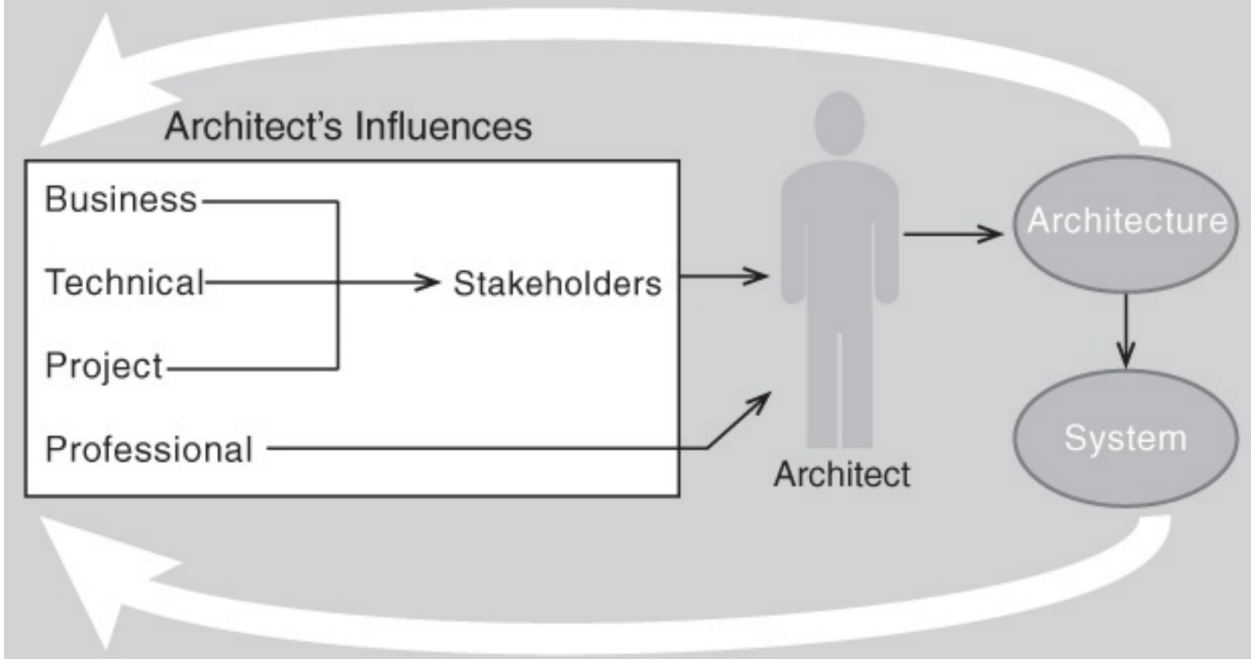
1 Motivations and objectives

1.1 On a technical perspective

- Software architecture is about preliminary design
  - Each stakeholder (customer, user, project manager, coder, tester, and so on) is concerned with different characteristics of the system
- Software architecture is about design at large
  - Provides a language in which different concerns can be expressed, negotiated, and resolved at a level that is manageable (by one person) for large, complex systems
- The early design decisions carry enormous weight with respect to the system's remaining development, its deployment, and its maintenance life
- It is the earliest point at which these design decisions can be scrutinized

1.2 On a business perspective

- A documented architecture enhances communication among stakeholders
- An architecture channels the creativity of developers, reducing design and system complexity
- Architecture-based development focuses on finding a stable design and a stable (predictable) development plan



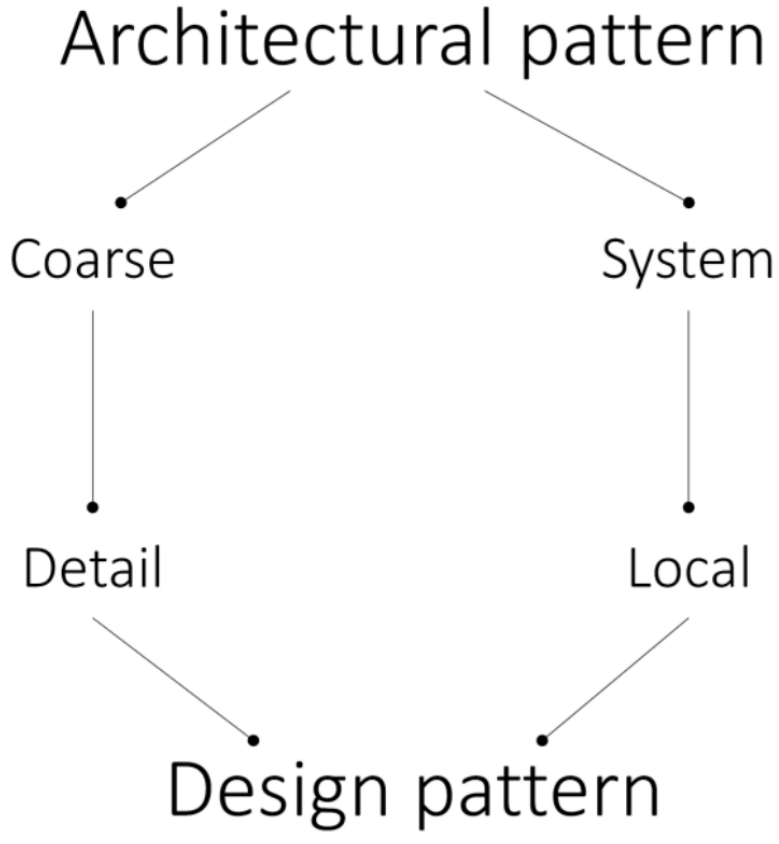
- Documentation of the architecture: See [ISO/IEC/IEEE, 2011]

1.3 Software architecture and Middleware

- Middleware is
  - *Middleware is software glue*
  - *Middleware is computer software that connects software components or applications. It is used most often to support complex, distributed applications.*
  - *Middleware is any software that allows other software to interact*
- In short, in the "Component-and-connector" view of a software architecture, middleware is about the "connector" part
  - Design patterns exist for the design of the connectors

    Home work: Study the slides "Introduction to design patterns for middleware":
    - Asynchronous call, Synchronous call, Buffered messages, Inversion of control, Proxy, Wrapper, Interceptor, Component/Container, etc.
- Middleware and architectural patterns are are strongly related
  - See M. Richards, "Software Architecture Patterns" Web site of the book

1.4 Architectural patterns Vs. Design patterns



2 Software architecture and Views

2.1 Definition of "Software architecture"

- "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" [Bass et al., 2012]
- Software architecture = an abstraction —i.e. omits certain information
  - Elements interact with each other by means of interfaces that partition details into public and private parts
  - Architecture focuses on the public side of this division
- Desirable properties of software architectures:
  - Can be constructed, evaluated, and documented
  - Answer to requirements to satisfy stakeholders
  - Have a repertoire of patterns and description languages (Architecture Description Language, ADL)

2.1.1 Examples of sets of software structures

- Module decomposition structures = Implementation units
  - What is the primary functional responsibility, e.g. assigned to each element?
  - What other elements is an element allowed to use?
  - What other software does it actually use and depend on?
- Component-and-connector structures = runtime entities, e.g.
  - What are the major runtime elements and how do they interact?
  - What are the major shared data stores?
  - Which parts of the system are replicated? Can run in parallel?
  - Can the system's structure change as it executes and, if so, how?
- Allocation structures = mapping from software structures to organizational, developmental, installation, e.g.
  - What is the assignment of each software element to development teams?
    - Brook's Mythical Man-Month: group inter-communication = $O(n^2)$
    - Conway's law: design structure = a copy of the organization's communication structure

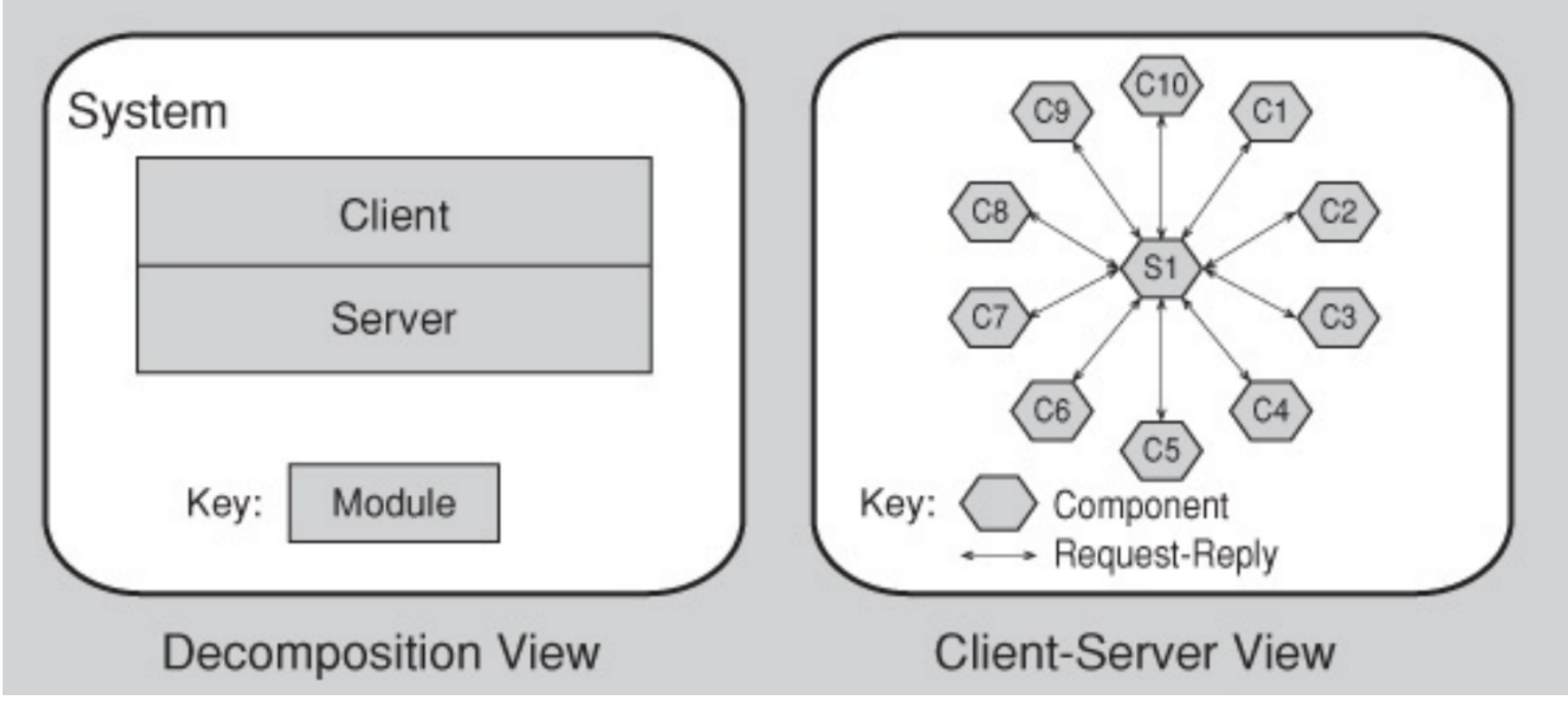
2.2 Other architectures: System and Enterprise

- System architecture
  - Is concerned with a total system, including hardware, software, and humans
  - In this presentation, we limit ourselves to software architecture of software-intensive systems
    - E.g., we do not target hardware-software co-design of embedded systems
- Enterprise architecture
  - Software is only one concern of enterprise architecture
  - Other common concerns addressed by enterprise architecture are how the software is used by humans to perform business processes, and how it is organised into subunits that aligned with the organization's core goals and strategic direction
- Each type of architecture has its own specialized vocabulary and techniques

2.3 Views of a software architecture

- Each of the software structures provides a different perspective
  - E.g. module decomposition, component-and-connector, allocation
- Although they give different system perspectives, they are not independent
  - Elements of one structure will be related to elements of other structures
  - We need to reason about the relations
- A view is a representation of a set of elements and relations among them
  - Not all system elements, but those of a particular type
- Documenting an architecture is a matter of documenting the relevant views and then adding documentation that applies to more than one view

2.3.1 Example of Client–Server with two views

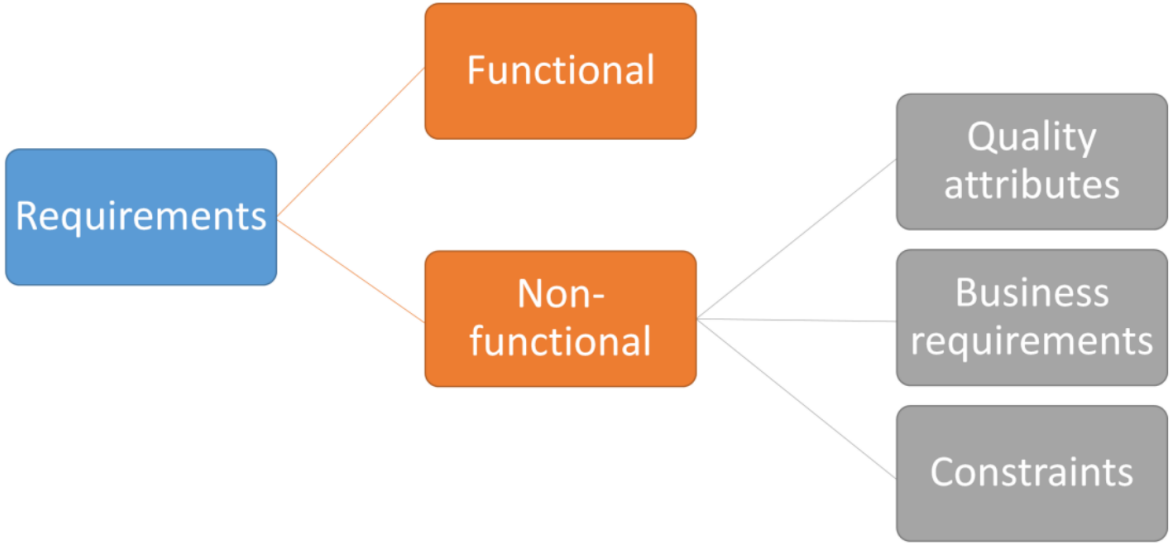


3 Attribute-Driven Design (ADD)

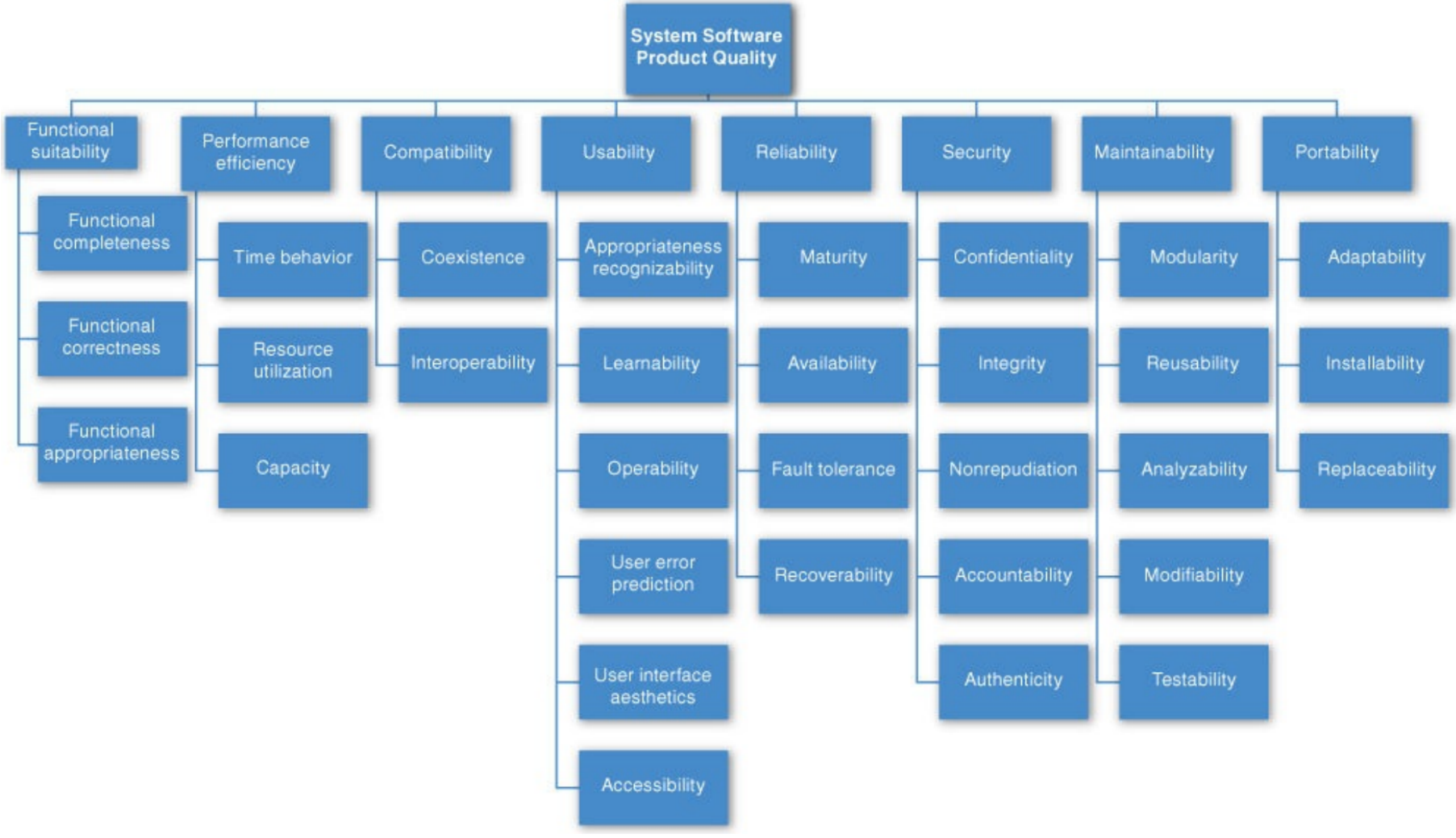
3.1 Quality attribute requirements

3.1.1 Definition of "Quality attribute"

- "A quality attribute is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders" [Bass et al., 2012]
- Quality is related to the functions as perceived by the user or customer
- Quality is about the extra-functional characteristics: modifiability, usability, testability, scalability, availability, security, etc.
- Quality attributes (when architecting) $\neq$ Constraints (taken before)



3.1.2 ISO/IEC 25010 product quality standard I



3.1.2 ISO/IEC 25010 product quality standard II

- Functional suitability: The degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions
- Performance efficiency: Performance relative to the amount of resources used under stated conditions
- Compatibility: The degree to which a product, system, or component can exchange information with other products, systems, or components, and/or perform its required functions, while sharing the same hardware or software environment
- Usability: The degree to which a product or system can be used by users to achieve goals with effectiveness, efficiency, and satisfaction in a context of use
- Reliability: The degree to which a system, product, or component performs specified functions under specified conditions for a specified period of time
- Security: The degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization
- Maintainability: The degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers
- Portability: The degree of effectiveness and efficiency with which a system, product, or component can be transferred from one hardware, software, or other operational or usage environment to another

3.1.3 Modelling quality attribute requirements I

- Quality attribute requirements are modelled into scenarios

- Let's take an example from the document that describes the micro-project:
  - Section 2.3, at page 7: "Each participant publishes his/her location. Each participant receives locations of other participants in the group."
  - Section 3, at page 11: "The system should be able to handle up to 3000 groups of tourists at a time, and handle up to 3000 $\times$ 10 tourists. The system notifies all the members of the group of the tourists within a maximum of 1 second after having received a location from a tourist."

- In Section 3, at page 11, of the document that describes the micro-project:
  - "In the report of the micro-project, we ask you to analyse the architecture of the application with regard to two quality attributes chosen among the following ones: (1) scalability, (2) security, and (3) interoperability."

3.1.3 Modelling quality attribute requirements II

Source	who or what	A tourist
Stimulus	does something	...broadcasts a location message to the members of their group
Environment	under certain conditions	...during normal operations, with at most 3000×10 tourists that are broadcasting at most one message per second,
Artifact	to the system or part of it	...to the group communication system.
Response	the system reacts with these actions	The group communication system notifies (sends notifications to) all the members of the group of the tourists
Resp. measure	which can be measured by these metrics	...within a maximum of 1 second after having received the broadcast message from the tourist.

3.1.3 Modelling quality attribute requirements III

- What to specify in stimulus-oriented requirements modelling

1. Source of stimulus: This is some entity (a human, a computer system, or any other actuator) that generated the stimulus
2. Stimulus: The stimulus is a condition that requires a response when it arrives at a system
3. Environment: The stimulus occurs under certain conditions (environment state). The system may be in an overload condition or in normal operation, or some other relevant state.
4. Artifact: Some artifact is stimulated. This may be a collection of systems, the whole system, or some piece or pieces of it
5. Response: The response is the activity undertaken by the system as the result of the arrival of the stimulus
6. Response measure: When the response occurs, it should be measurable in some fashion so that the requirement can be tested

- See Slide 2 in the appendices for some other illustrative scenarios

3.2 Tactics

3.2.1 Definition of “Tactic”
3.2.2 Working on “Tactics”

3.2.1 Definition of "Tactic"

- The quality attribute requirements specify the responses of the system that realize the goals of the business
- Techniques to achieve quality attributes are called architectural tactics
- "A tactic is a design decision that influences the achievement of **a** quality attribute response" [Bass et al., 2012]

- The focus of a tactic is on a single quality attribute response
  - Within a tactic, there is no consideration of tradeoffs
    - In this respect, tactics differ from architectural patterns, where tradeoffs are built into the pattern (See Slide 26)

3.2.2 Working on “Tactics” I

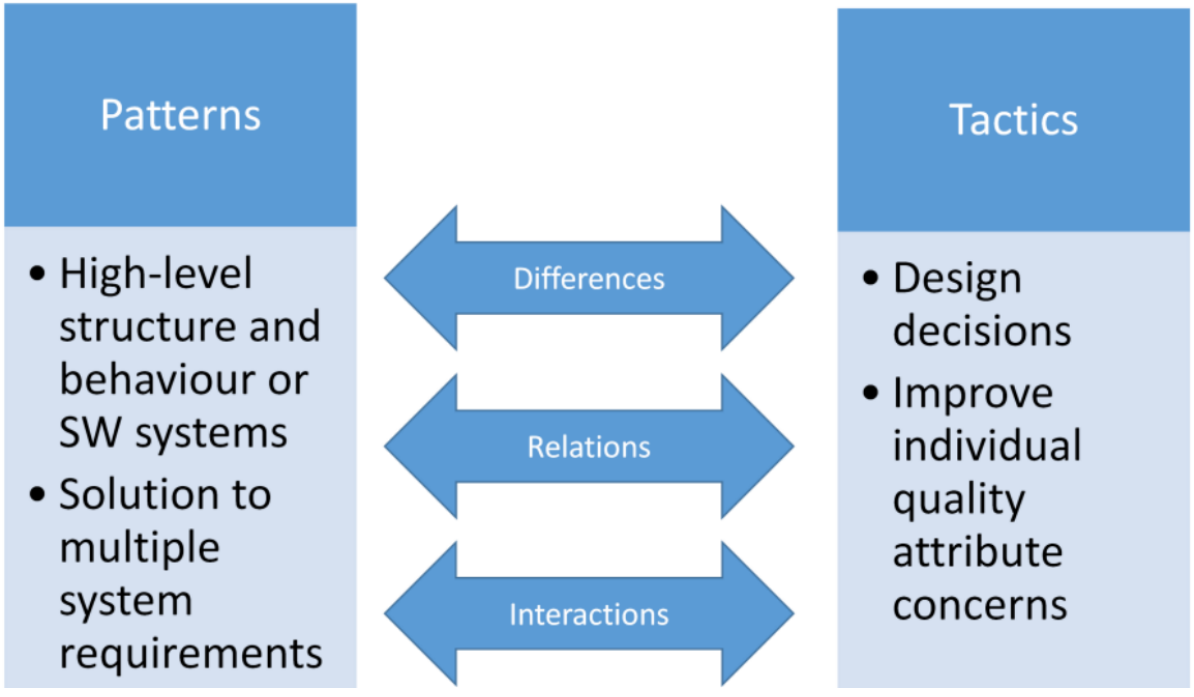
- We propose to work on tactics by studying chapters of Bass, Clements, and Kazman’s book [Bass et al., 2012]
  - Availability
  - Security
  - Interoperability
  - Modifiability
  - Scalability
  - Performance
- See Slide 10 in the appendices for some other examples of decisions/tactics

3.3 Architectural Pattern

- Architectural pattern = composition of architectural elements
  - is a bundle of design decisions that is found repeatedly in practice
  - has known properties that permet reuse
  - describes a class of architectures
- Exemples: layered pattern, shared-data or repository pattern, client-server pattern, multi-tier pattern, distributed event-based pattern
- Pattern cataloguers strive to understand how the characteristics lead to different behaviors and different responses to environmental conditions
  - There will never be a complete list of patterns
    - Patterns spontaneously emerge in reaction to environmental conditions
    - As long as those conditions change, new patterns will emerge

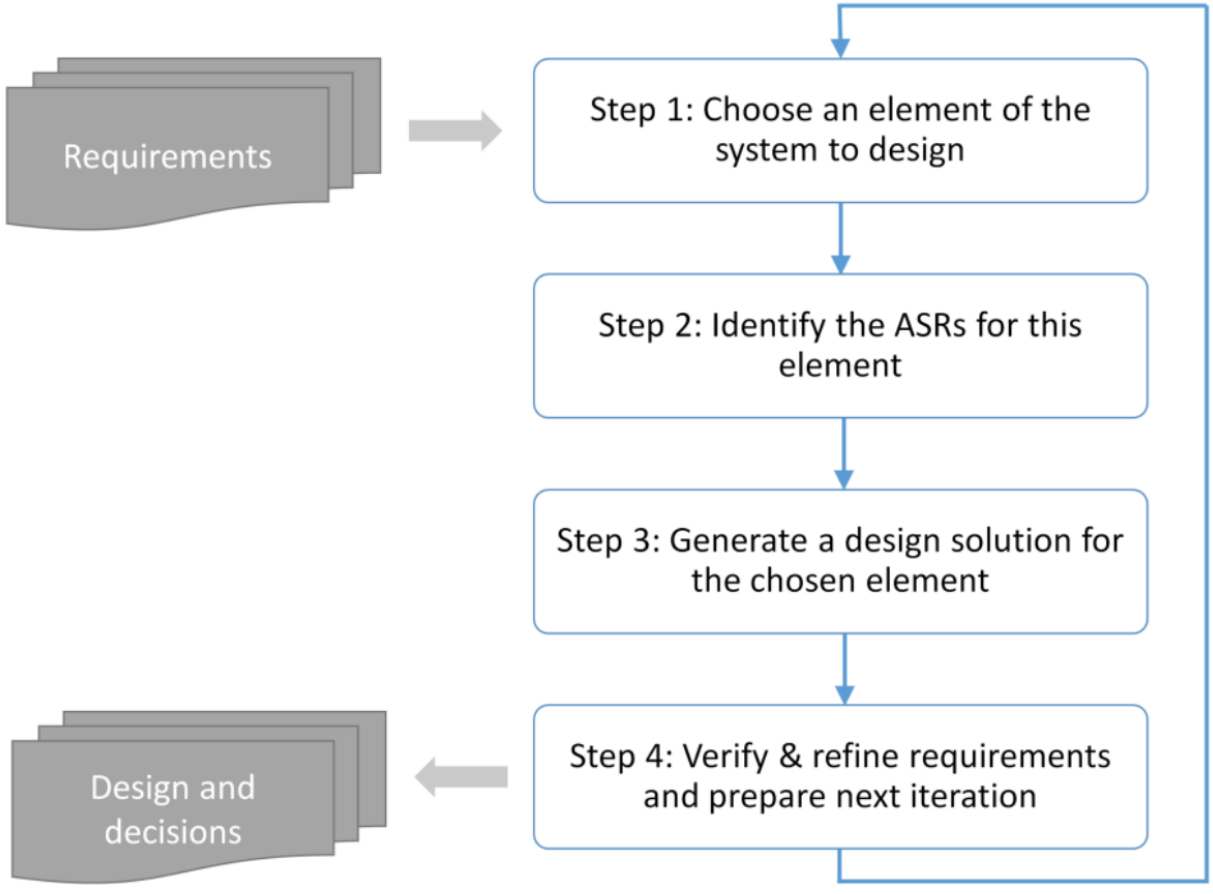
3.4 Tactics versus Architectural Patterns

- Architectural patterns consist of a bundle of design decisions/tactics
  - They are often difficult to apply as is. Architects need to modify/adapt them.



- By understanding the role of tactics, an architect can more easily assess the options for augmenting an existing pattern to achieve a quality attribute goal

3.5 ADD Methodology I



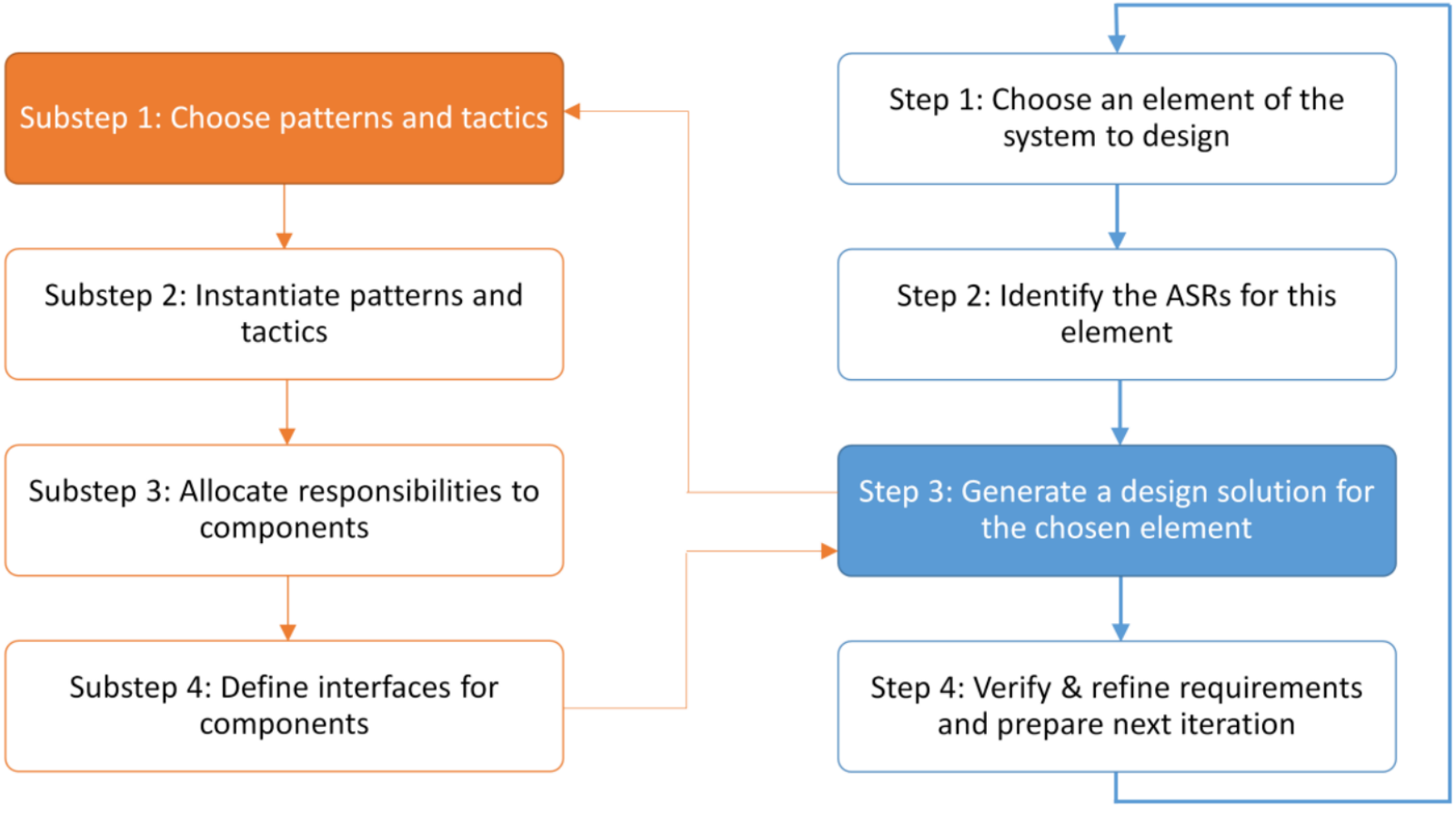
⟸ *Element is mostly a functional "component"*

⟸ *Design solution introduces "connectors" —i.e. middleware*

ASR = Architecture Significant Requirement: e.g. a quality attribute

Element = the whole system, a subsystem, or a component

3.5 ADD Methodology II



Instantiate patterns and tactics = Use the functional requirements to help instantiate the roles of the patterns and tactics

Responsabilities = Functionalities

References I

Bass, L., Clements, P., and Kazman, R. (2012).

Software Architecture in Practice, 3rd Edition.

Addison-Wesley.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., and Stafford, J. (2011).

Documenting Software Architecture: Views and Beyond, 2nd Edition.

Addison-Wesley.

Clements, P., Kazman, R., and Klein, M. (2002).

Evaluating Software Architectures: Methods and Case Studies.

Addison-Wesley.

ISO/IEC (1991).

Information technology — Software product evaluation — Quality characteristics and guidelines for their use.

International Standard ISO/IEC-9126, ISO/IEC Joint Technical Committee.

ISO/IEC/IEEE (2011).

Systems and software engineering — Architecture description.

International Standard ISO/IEC/IEEE-42010:2011, ISO/IEC/IEEE Joint Technical Committee.

References II

Richards, M. (2015).

Software Architecture Patterns: Understanding Common Architecture Patterns and When to Use Them.

O'Reilly.

Sunyaev, A. (2020).

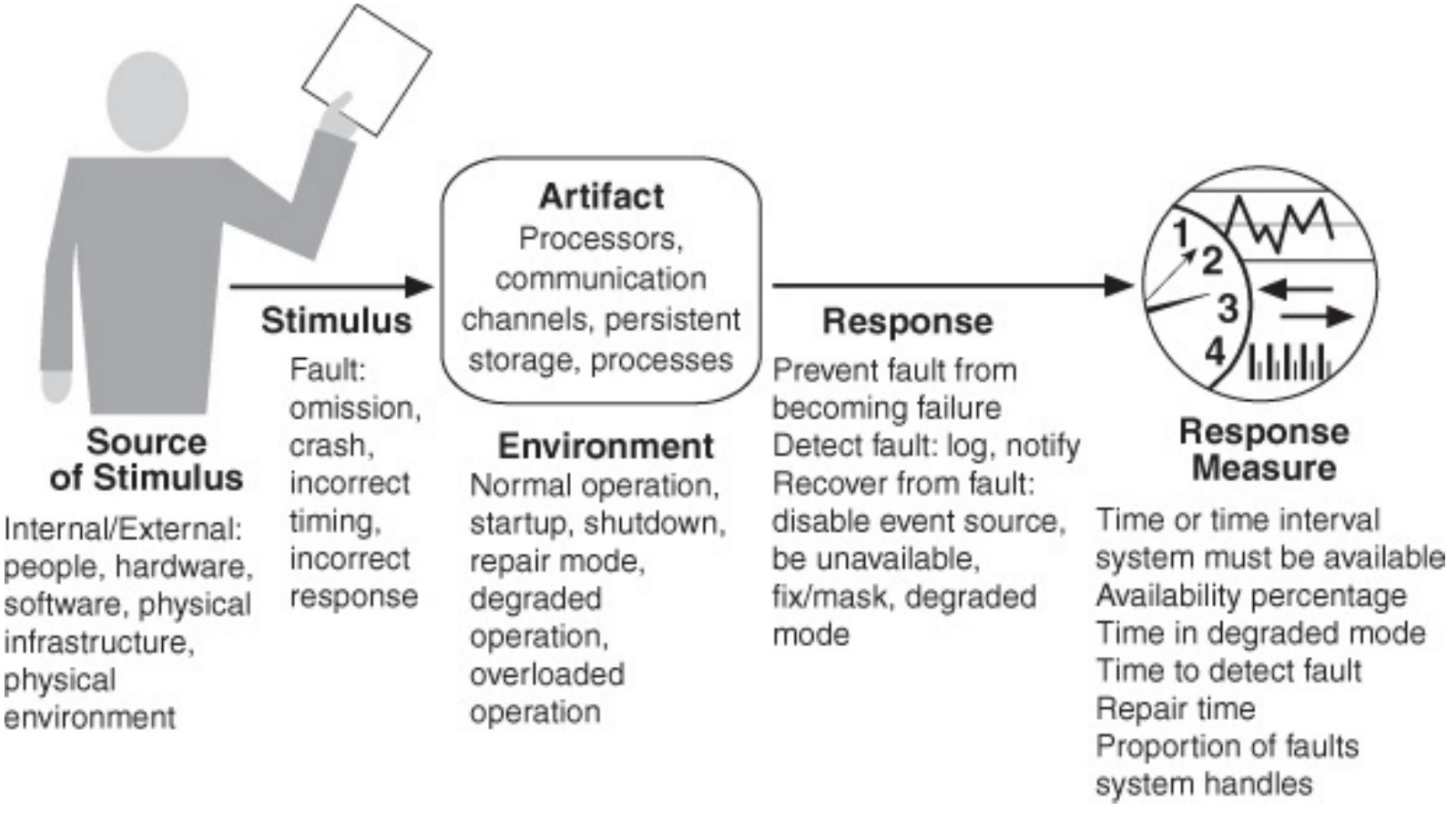
Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies.

Appendices

4. Examples of illustrative scenarios of quality attribute requirements
5. Example of decisions for some tactics
6. Example of a catalog of questions

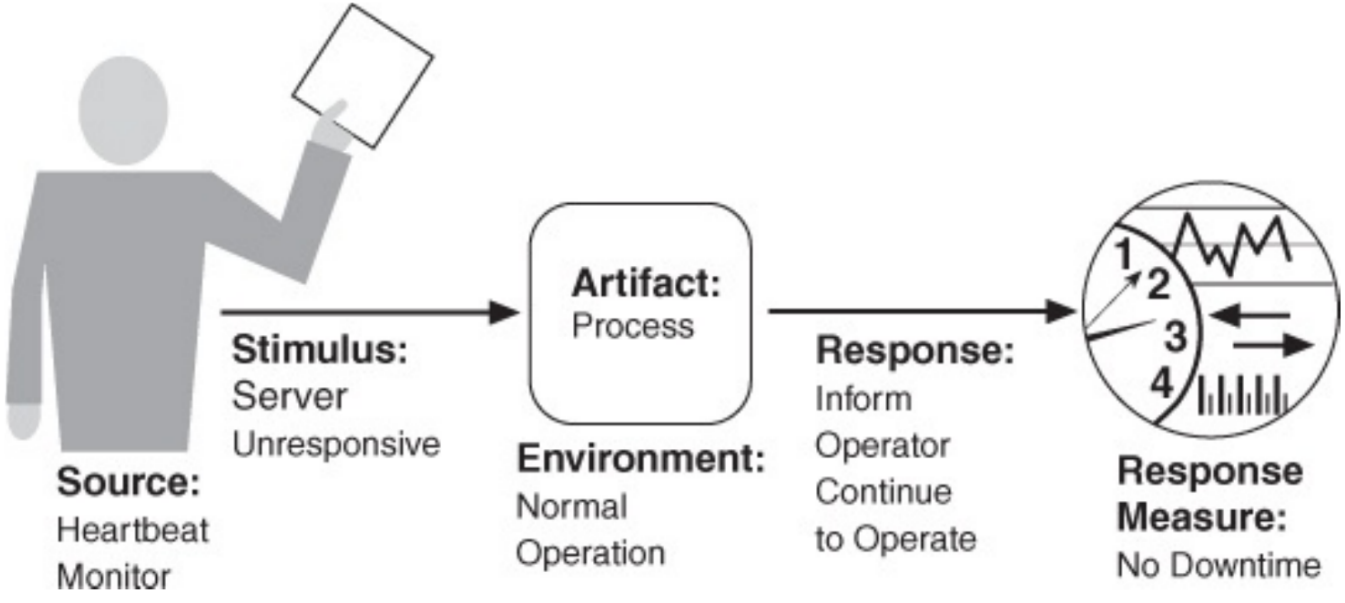
4 Examples of illustrative scenarios of quality attribute requirements I

- In [Bass et al., 2012], they are modelled in graphics



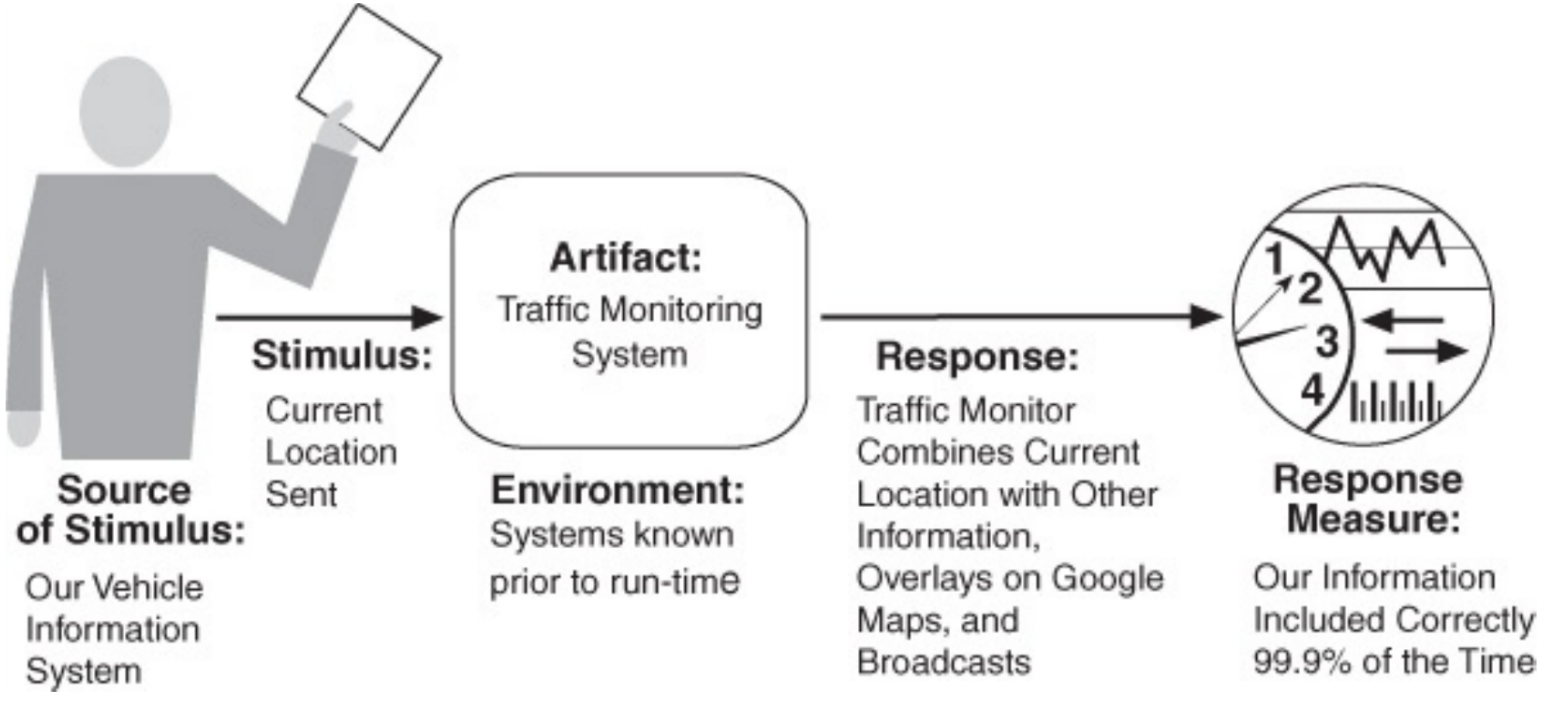
4 Examples of illustrative scenarios of quality attribute requirements II

- Availability



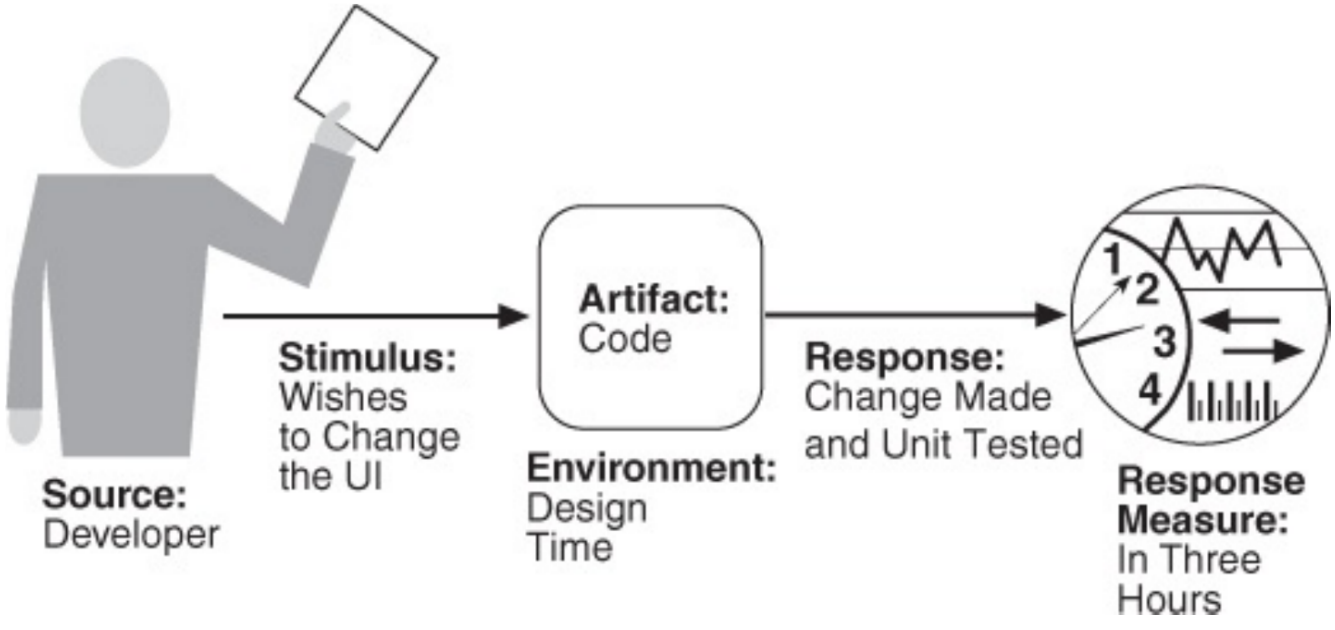
4 Examples of illustrative scenarios of quality attribute requirements III

- Interoperability



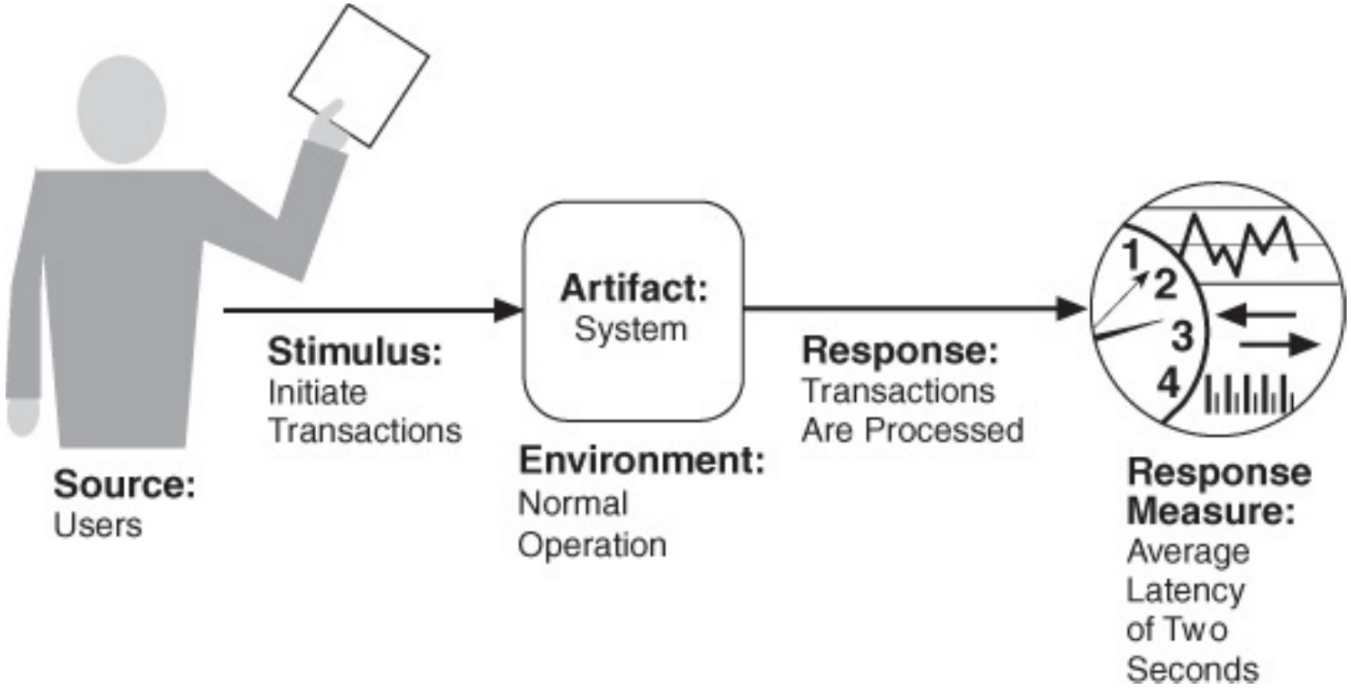
4 Examples of illustrative scenarios of quality attribute requirements IV

- Modifiability



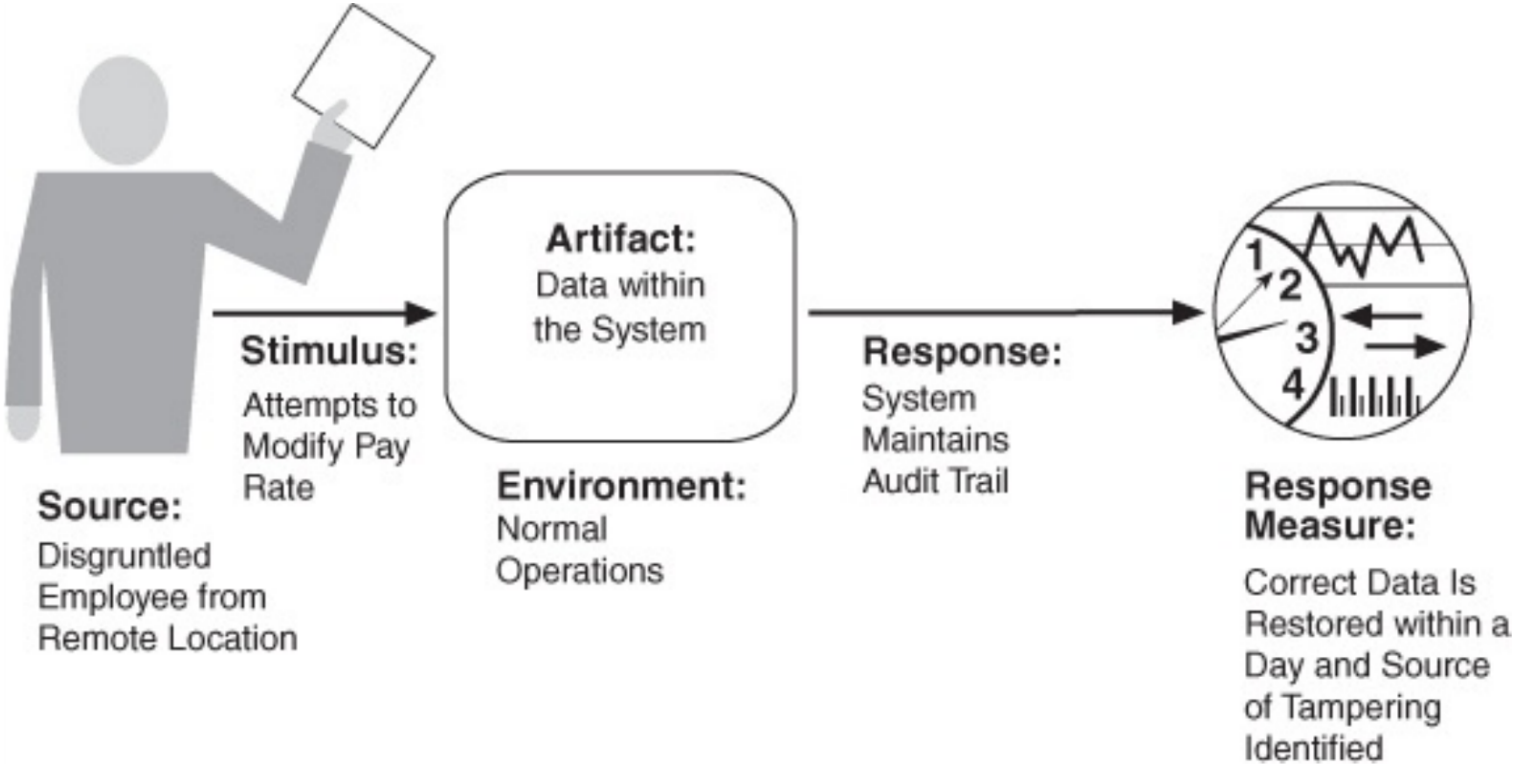
4 Examples of illustrative scenarios of quality attribute requirements V

- Performance



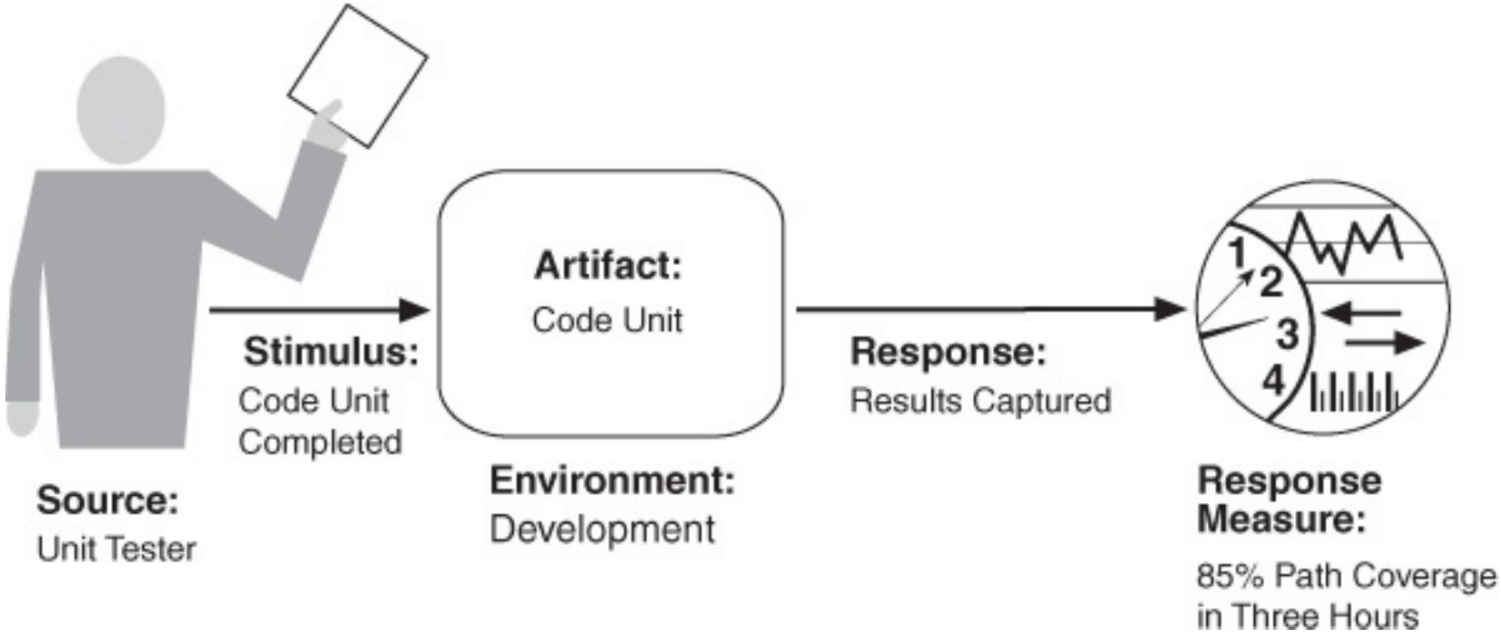
4 Examples of illustrative scenarios of quality attribute requirements VI

- Security



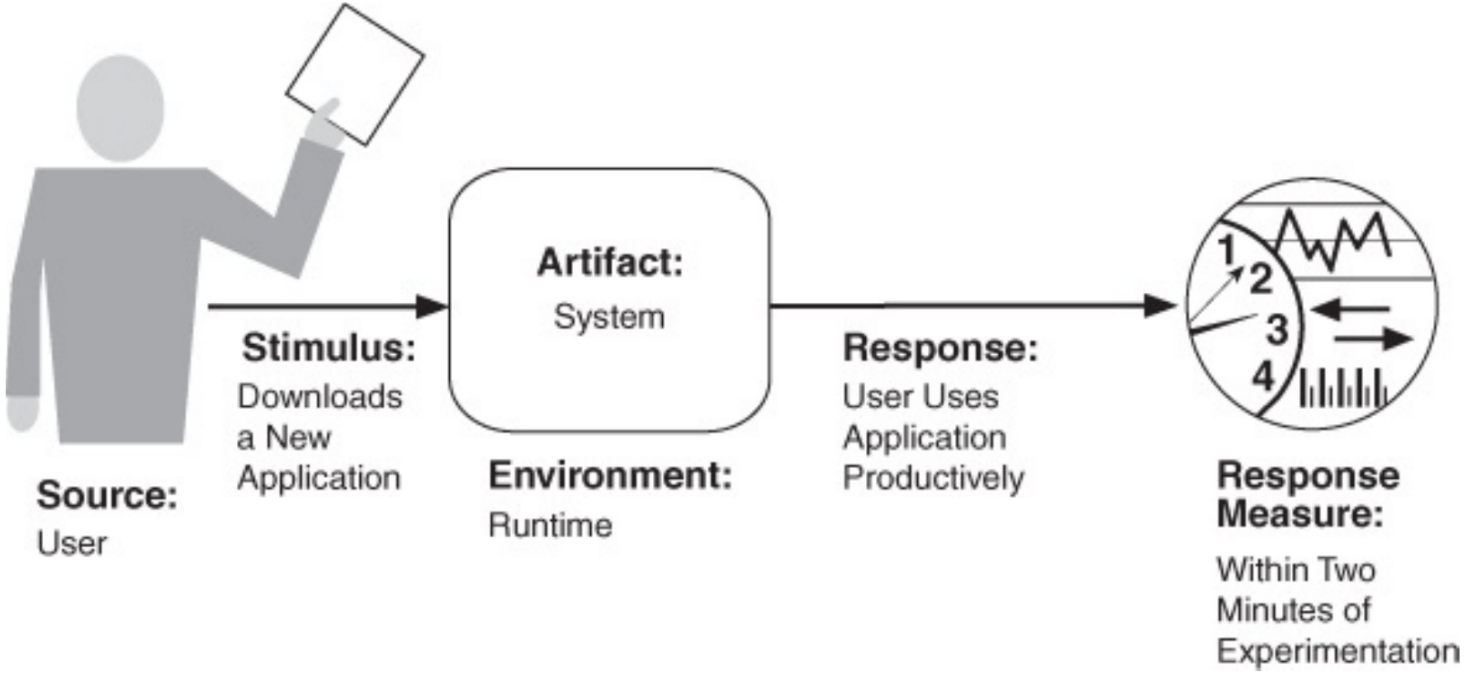
4 Examples of illustrative scenarios of quality attribute requirements VII

- Testability



4 Examples of illustrative scenarios of quality attribute requirements VIII

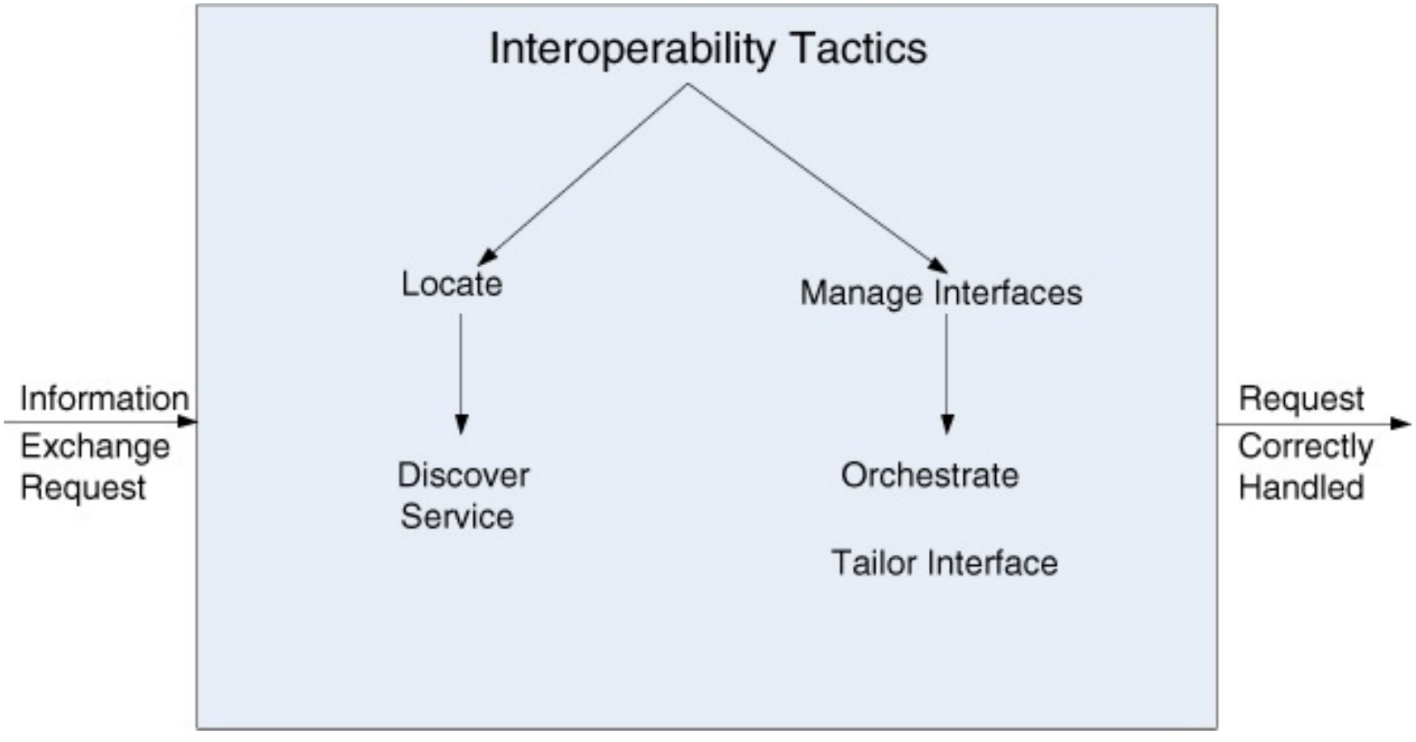
- Usability



5 Example of decisions for some tactics I

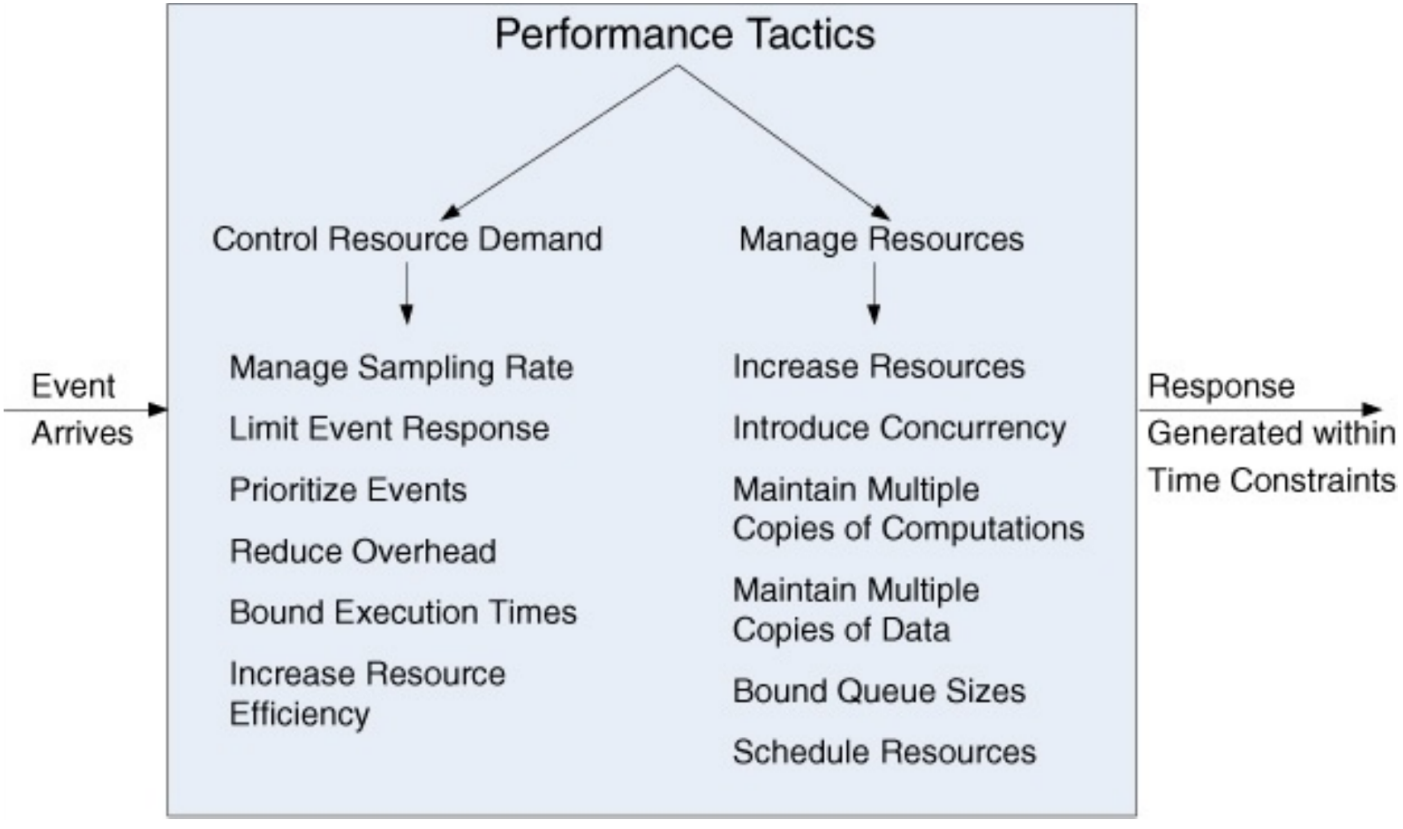
- In [Bass et al., 2012], sets of decisions are graphically displayed. The slides of this section contain some of these graphics.

- Interoperability tactics



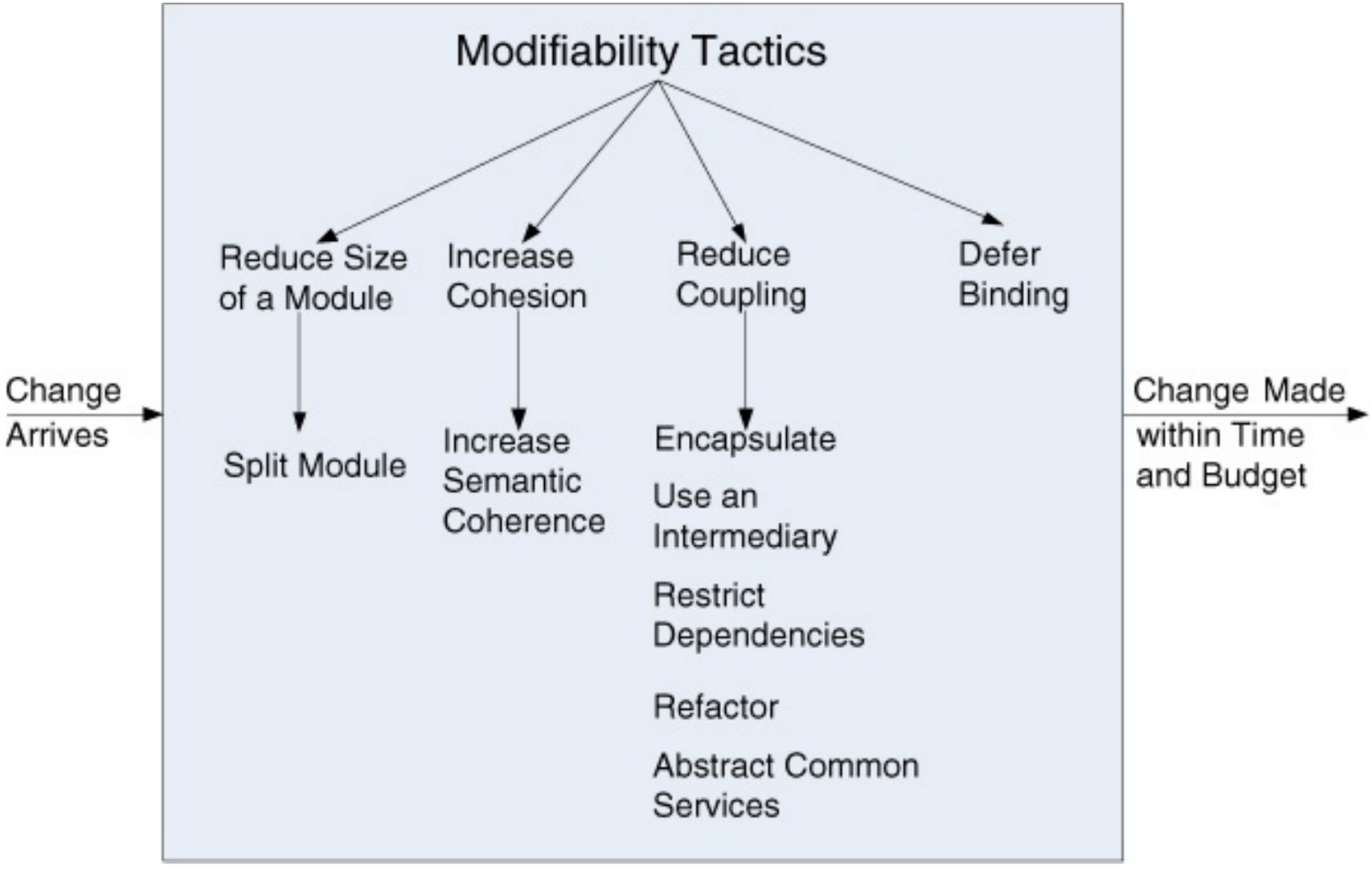
5 Example of decisions for some tactics II

- Performance tactics



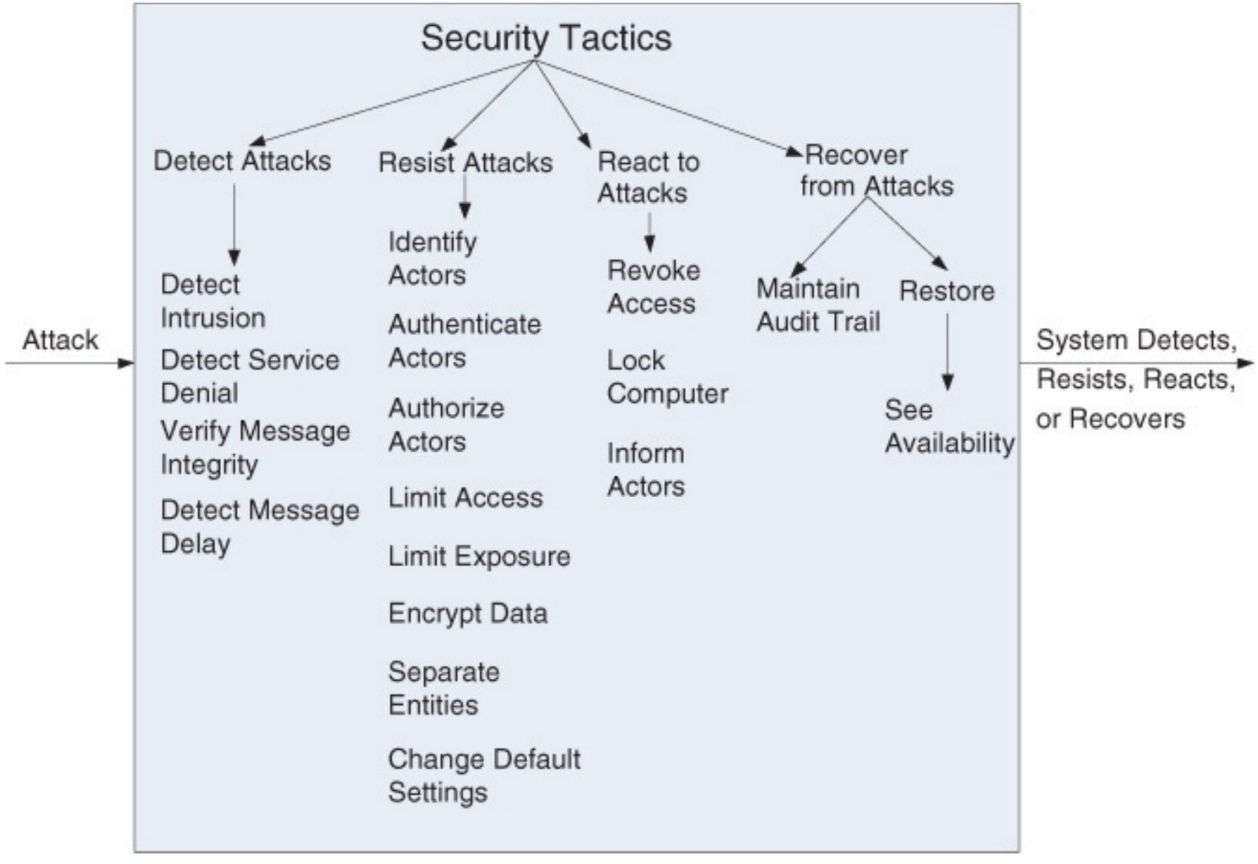
5 Example of decisions for some tactics III

- Modifiability tactics



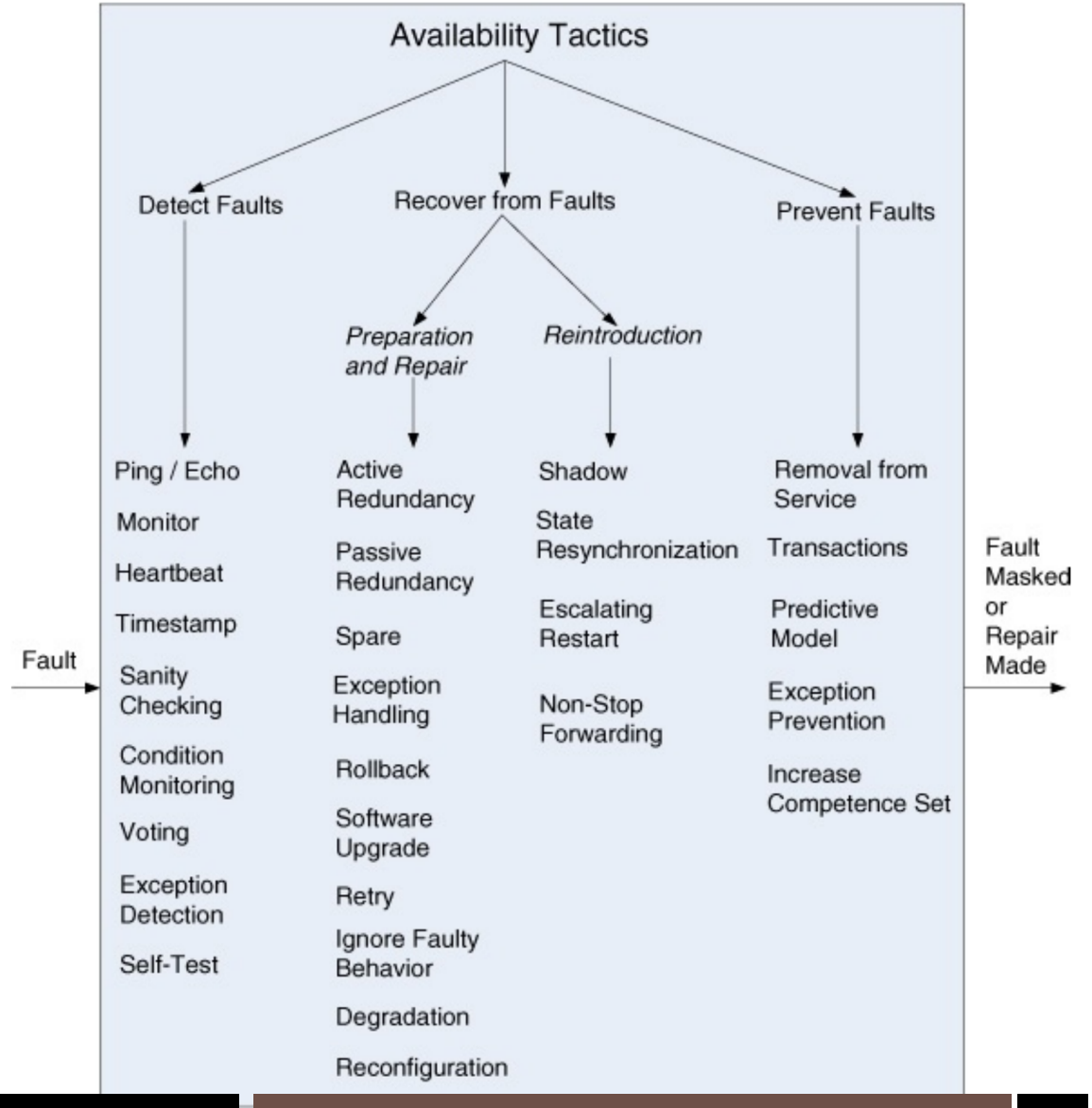
5 Example of decisions for some tactics IV

- Security tactics



5 Example of decisions for some tactics V

- Availability tactics



6 Example of a catalog of questions I

- We can view an architecture as the result of applying a collection of design decisions
- Slides of this section propose some initial categories of decisions
  - During the design of your solution, use these slides as a reminder
- These design decisions are fine-grained; this is why there are inserted in the appendix

6 Example of a catalog of questions II

- Allocation of responsibilities
  - Identifying the important responsibilities, including basic system functions, architectural infrastructure, and satisfaction of quality attributes
  - Determining how these responsibilities are allocated to non-runtime and runtime elements (namely, modules, components, and connectors)

6 Example of a catalog of questions III

- Coordination model
  - Identifying the elements of the system that must coordinate, or are prohibited from coordinating
  - Determining the properties of the coordination, such as timeliness, currency, completeness, correctness, and consistency
  - Choosing the communication mechanisms (between systems, between our system and external entities, between elements of our system)
    - Stateful versus stateless
    - Synchronous versus asynchronous
    - Guaranteed versus nonguaranteed delivery
    - Performance-related properties such as throughput and latency

6 Example of a catalog of questions IV

- Data model
  - Choosing the major data abstractions, their operations, and their properties
    - How the data items are created, initialized, accessed, persisted, manipulated, translated, and destroyed
  - Compiling metadata needed for consistent interpretation of the data
  - Organizing the data
    - In a relational database, a collection of objects, or both
      - If both, then the mapping between the two different locations of the data must be determined

6 Example of a catalog of questions V

- Management of resources
  - Identifying the resources that must be managed and determining the limits for each
  - Determining which system element(s) manage each resource
  - Determining how resources are shared and the arbitration strategies employed when there is contention
  - Determining the impact of saturation on different resources

6 Example of a catalog of questions VI

- Mapping among architectural elements
  - The mapping of modules and runtime elements to each other
    - The runtime elements that are created from each module
    - The modules that contain the code for each runtime element
  - The assignment of runtime elements to processors
  - The assignment of items in the data model to data stores
  - The mapping of modules and runtime elements to units of delivery

6 Example of a catalog of questions VII

- Binding time decisions
  - Binding time decision establishes the scope, the point in the life cycle, and the mechanism for achieving the variation
    - For allocation of responsibilities, you can have build-time selection of modules via a parameterized makefile
    - For choice of coordination model, you can design runtime negotiation of protocols
    - For resource management, you can design a system to accept new peripheral devices plugged in at runtime, after which the system recognizes them and downloads and installs the right drivers automatically
    - For choice of technology, you can build an app store for a smartphone that automatically downloads the version of the app appropriate for the phone of the customer buying the app.

6 Example of a catalog of questions VIII

- Choice of technology
  - Deciding which technologies are available to realize the decisions made in the other categories
  - Determining whether the available tools to support this technology choice (IDEs, simulators, testing tools, etc.) are adequate for development
  - Determining the extent of internal familiarity as well as the degree of external support available for the technology (such as courses, tutorials, examples, and availability of experts) and deciding whether this is adequate
  - Determining the side effects of choosing a technology, such as a required coordination model or constrained resource management opportunities
  - Determining whether a new technology is compatible with the existing technology stack
    - Can the new technology run on top of or alongside the existing technology stack?
    - Can it communicate with the existing technology stack?
    - Can the new technology be monitored and managed?