Principles & Tactics

Architectural Thinking for Intelligent Systems

Winter 2020/2021

Prof. Dr. habil. Jana Koehler

Agenda

- Implementation of functional and non-functional requirements applying principles & tactics
- 10 principles
  - Loose Coupling
  - High Cohesion
  - Design for Change
  - Separation of Concerns
  - Information Hiding
  - Abstraction
  - Modularity
  - Traceability
  - Self-Documentation
  - Incrementality
- Tactics as a method to address a quality attribute

Assignment for this Lecture

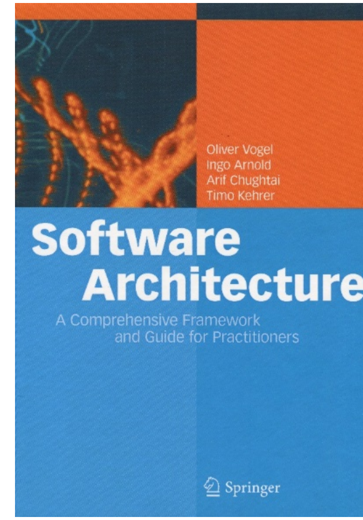
- We apply principles and tactics to further refine the architecture of our system.
- We choose the two most important architectural principles, which will guide architectural decision making.
- We also decide for specific tactics that help us to achieve the desired system qualities.

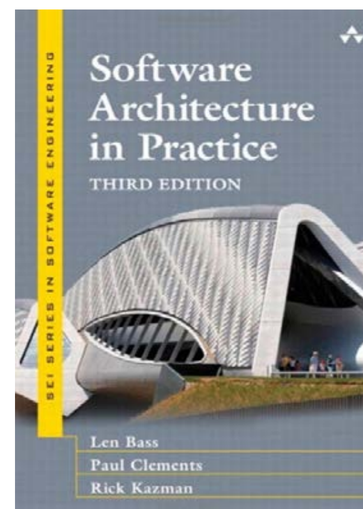
Recommended Reading

- Vogel et al.:
  - Chapter 6.1 (pages 115-139)

- Bass et al.:
  - Chapter 4.5 (pages 70-72)
  - See also http://www.ece.ubc.ca/~matei/EECE417/BASS/index.html

- Optional reading
  - Chapters 5-12 of the book by Bass discuss each quality attribute in detail and provide specific tactics in the second subchapters
  - Read the tactics for the 2 quality attributes that you selected in Assignment 6 and use a few of them in Assignment 8





Conception of the Architecture

- Use cases, user stories & scenarios are clarified such that we can proceed with an acceptable level of risks
- The system idea has been developed
- The context view has been reviewed with stakeholders to reach agreement on what will be built and what the system require from/provides to the environment
- We have a good understanding of the most important architectural decisions that we need to address

➢ We can start developing the architecture and <u>prototype</u> critical parts of the system

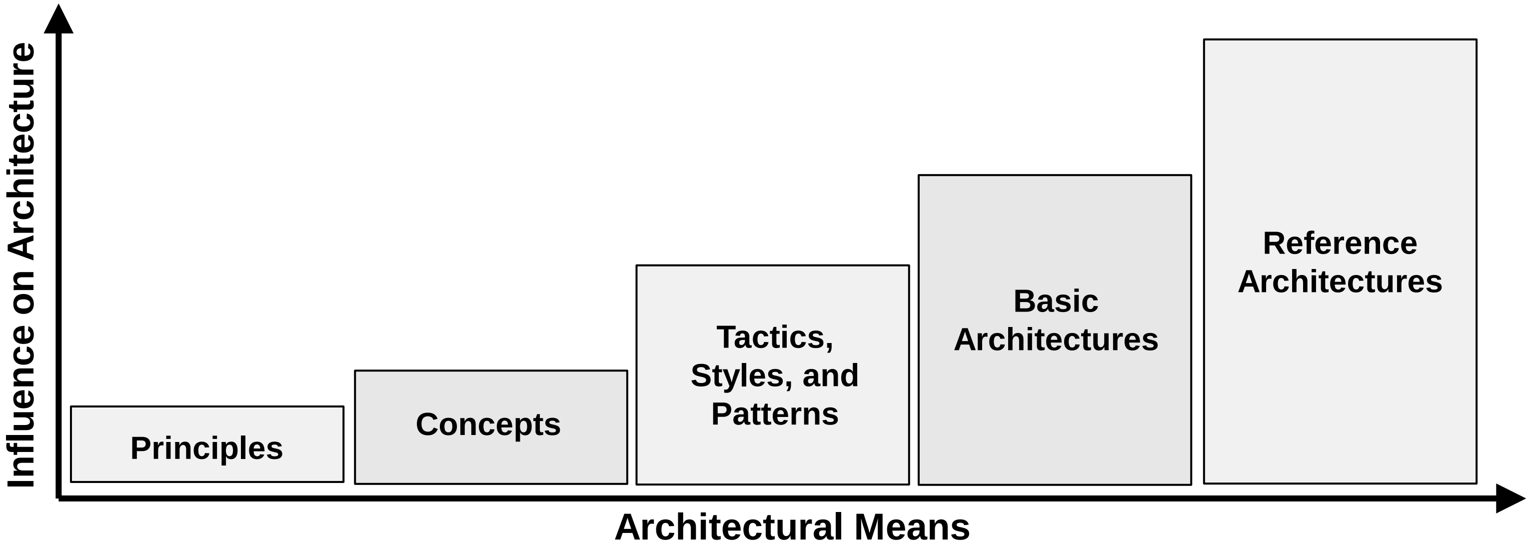
➢ Principles & Tactics & Styles & Pattern help to implement the solution using proven experience

Principles

"It is only through the relationships between the components of a system that an architecture really takes effect."

Influence of Architectural Means on Architecture

- Principles, such as cohesion or coupling, provide general guidelines
- Architectural styles, tactics and patterns provide detailed solutions for concrete design decisions
- Concepts, such as object orientation or aspect orientation, help implementing principles in the software design



Architectural Principles

- Architectural principles provide proven foundations on which the architecture can be built
- 2 main objectives
  - Reduction of complexity
  - Increased flexibility/changeability by using a good system structure
- Do not say anything about how these principles are applied and implemented in a specific system, but give important orientation towards the quality of the system structure

10 Basic Principles

1. Loose Coupling
2. High Cohesion
3. Design for Change
4. Separation of Concerns
5. Information Hiding
6. Abstraction
7. Modularity
8. Traceability
9. Self-Documentation
10. Incrementality

Remark: For the exam, you should be able to explain each principle in your own words.

1. Loose Coupling

- Any type of dependency between two components leads to a coupling
  - Mutual calls, shared data, …

- For a given coupling, one component is the provider, the other the consumer
  - A calls B: A is consumer, B provider
  - A includes B: A is consumer, B provider
  - A writes in a message queue, from which B reads: …
- Coupling is **tight**, if changes in the provider affect the consumer(s)
- Goal must be a loose coupling where providers can change without affecting consumers

2. High Cohesion

- Describes dependencies between structures (subcomponents) <u>within</u> one component
  - Example: methods calling each other within a class

- Components should include all elements, which implement the relevant and connected behaviors of this component
  - Check: Can I understand and change a component without understanding/changing other components?
  - How easy is the component to understand?

- Encapsulate related functionality in one component

3. Design for Change

- Anticipate *foreseeable* changes in the architecture
  - e.g. from open requirements that had to be moved to a next release of a software

- Ignore unforeseeable requirements!
  - Problem of a "too flexible" architecture

- Design components based on goal hierarchies and interrelated user stories

4. Separation of Concerns

- Separate different aspects of a problem from each other and deal with each of these subproblems separately

- Each functionality is implemented in exactly one component and only there

- Break down
  - Requirements
  - organizational responsibilities
  - system into a structure of subsystem
  - complex architecture description into views
  - process of architecture creation into subprocesses

5. Information Hiding

- Only reveal those information entities to a component, the component needs to function correctly
- Hide all other information entities
- Examples
  – OOP: data fields are „private“, data access only via getter and setter methods
  – Fassade pattern: shields complex systems and controls access to system components
  – Layers: layer n only uses layer n-1, does not know about other layers

6. Abstraction

- Identify important aspects, neglect unimportant details
  – Special case of information hiding

- Most widely used: interface abstraction

- Find commonalities in things that appear to be different at a first glance (entities, value objects, events, services)

- Use when negotiating functional requirements to identify these commonalities

7. Modularity

- Structure the system such that each component has a clearly defined functional responsibility
  - One problem solved in one place, completely and only there
- System components easily exchangeable and self-contained
  - manageable, understandable, easy to maintain and reusable
- Achieve simple and stable architectural relationships by finding the right balance between separating concerns and high cohasion

8. Traceability

- Ability to follow structures and architectural decisions from requirements to code (remember SMART)
- How easily can your architecture be understood?
- Traceability as a key to achieve long-term viability
- Foundation to map different views to each other and achieve a consistent description of the system
  - Elementary implementation: use uniform naming conventions

9. Self-Documentation

- Every information required to understand a component or system should be a direct part of the component or system
  - Reality: documentation and code get out of sync quickly

- The system documents itself - human documentation activity is reduced to a minimum
  - Self-speakng naming conventions
  - Clarity in structures and relationships by domain-driven design
  - The architecture becomes evident by looking at the code only!
  - Human-crafted documentation only simplifies the understanding

10. Incrementality

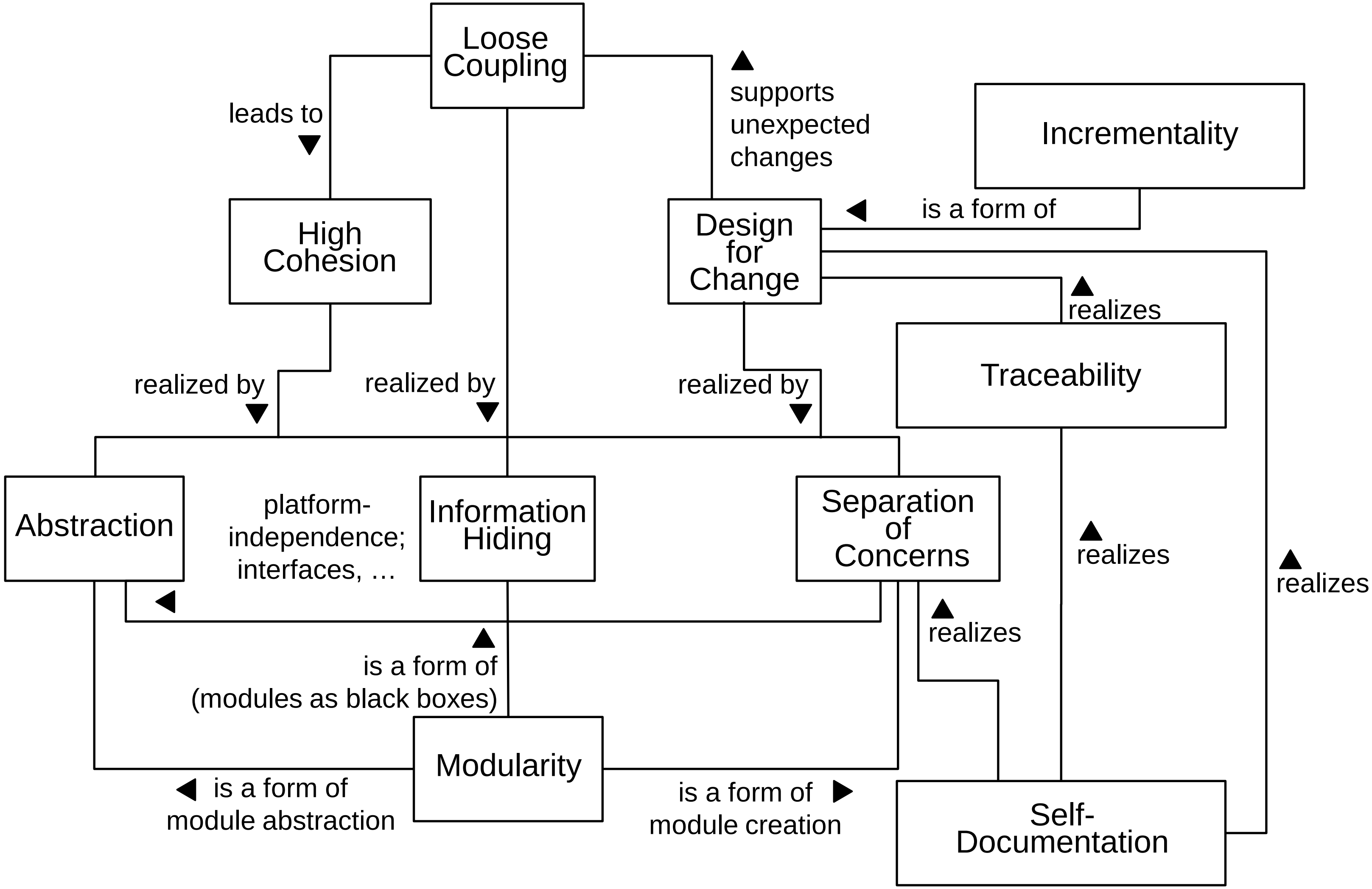
- Work in iterations when developing the architecture and build the system in stages
  - Work in phases, define milestones, review results

- Early prototyping
- Early feedback from stakeholders

- Build large systems in iterations (agile development)

- Allow for piecemeal growth through good release planning
  - Goal hierarchies and scenarios help

How Principles Support Each Other

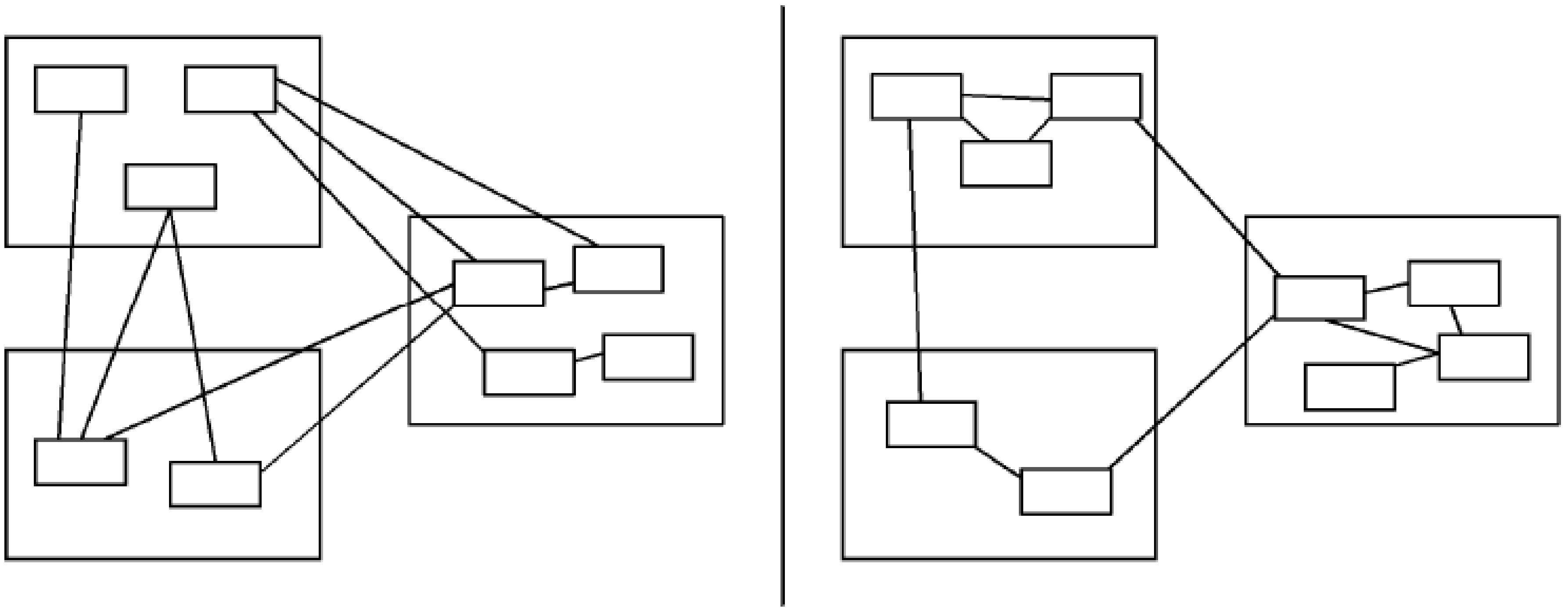


Vogel et al., page 119

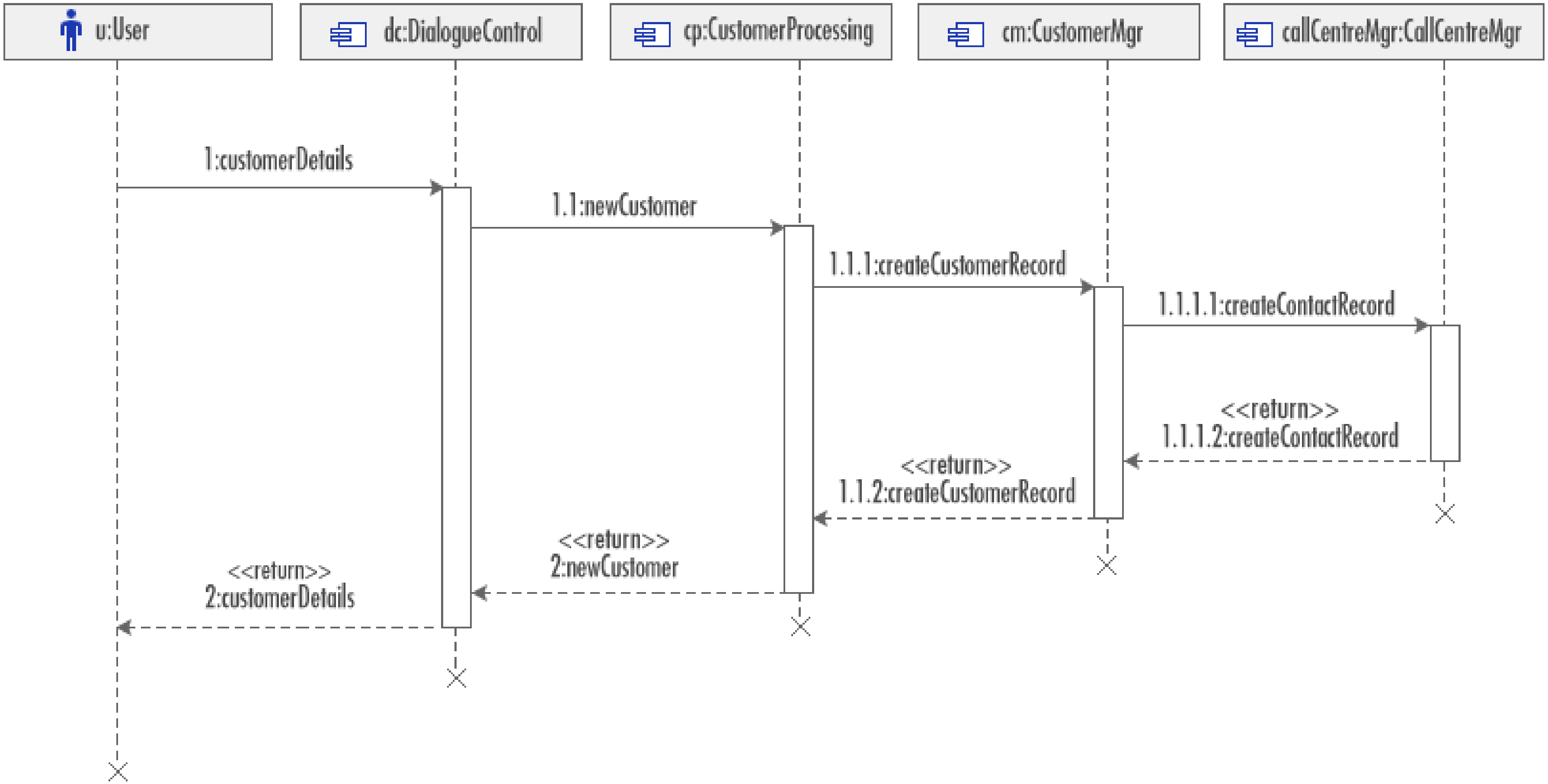
Principles help to implement other Principles

- High cohesion can be achieved by abstraction, separation of concerns und information hiding
- Loose coupling and high cohesion can be achieved by modularity
- Design for change can be achieved by loose coupling, abstraction, modularity, separation of concerns and information hiding
- Abstraction helps to implement loose coupling, modularity
- Modularity combines abstraction, separation of concerns and information hiding and supports high cohesion and loose coupling
- Traceability supports loose coupling and design for change
- Self-Documentation supports design for change and traceability

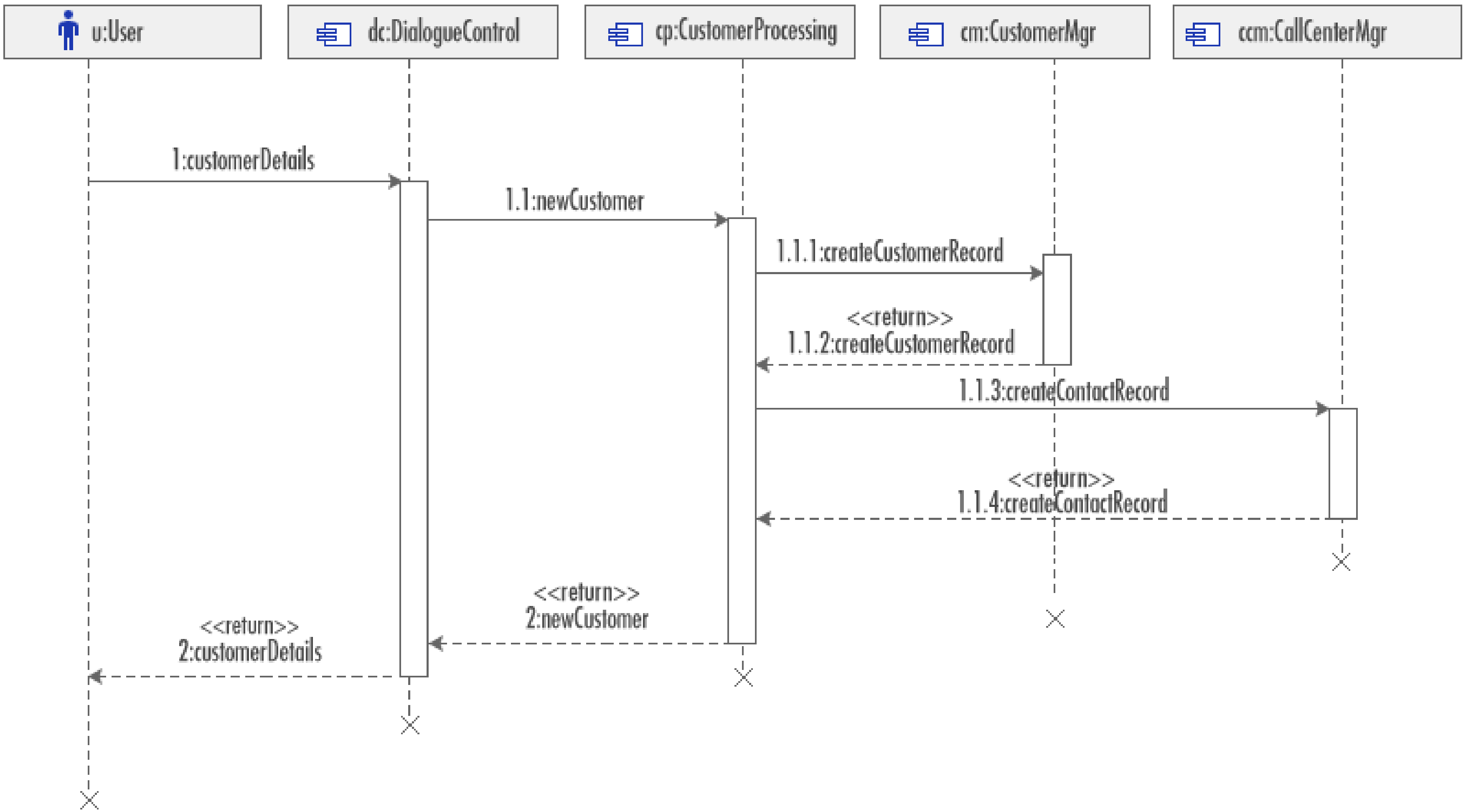
How do you assess Coupling and Cohesion in both Systems?



How do you assess the Coupling in this Example?

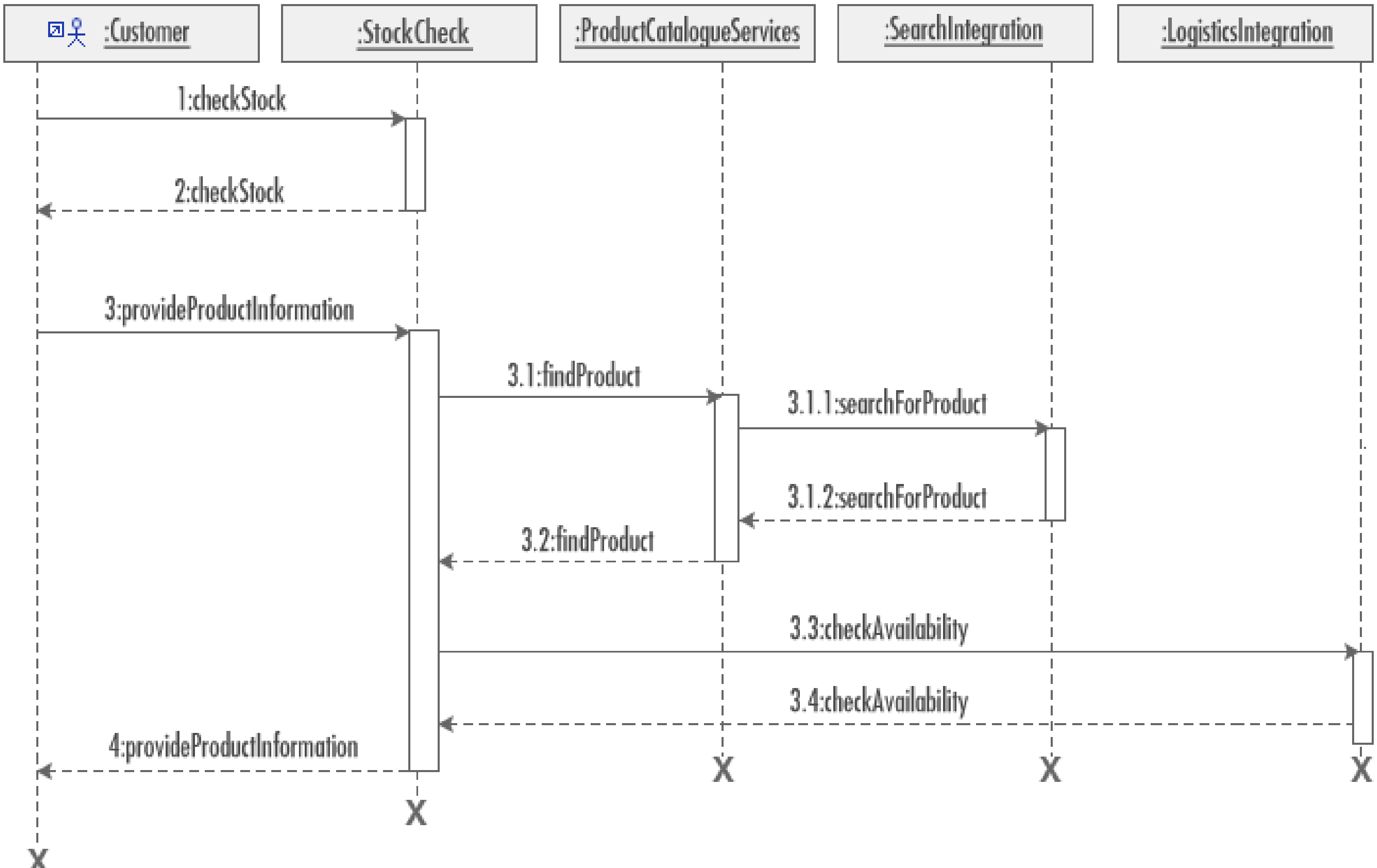


And here?

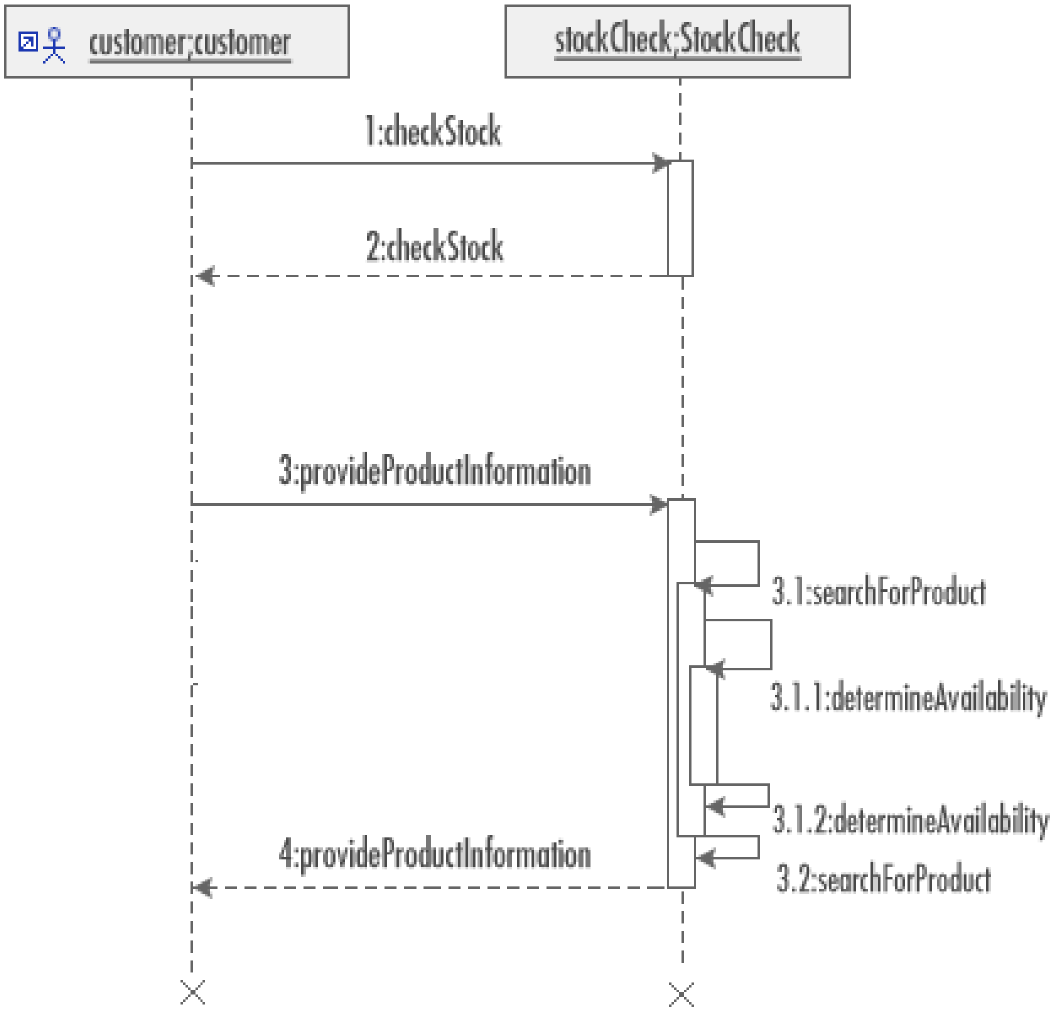


Quelle: IBM

How about the Cohesion of the StockCheck component?

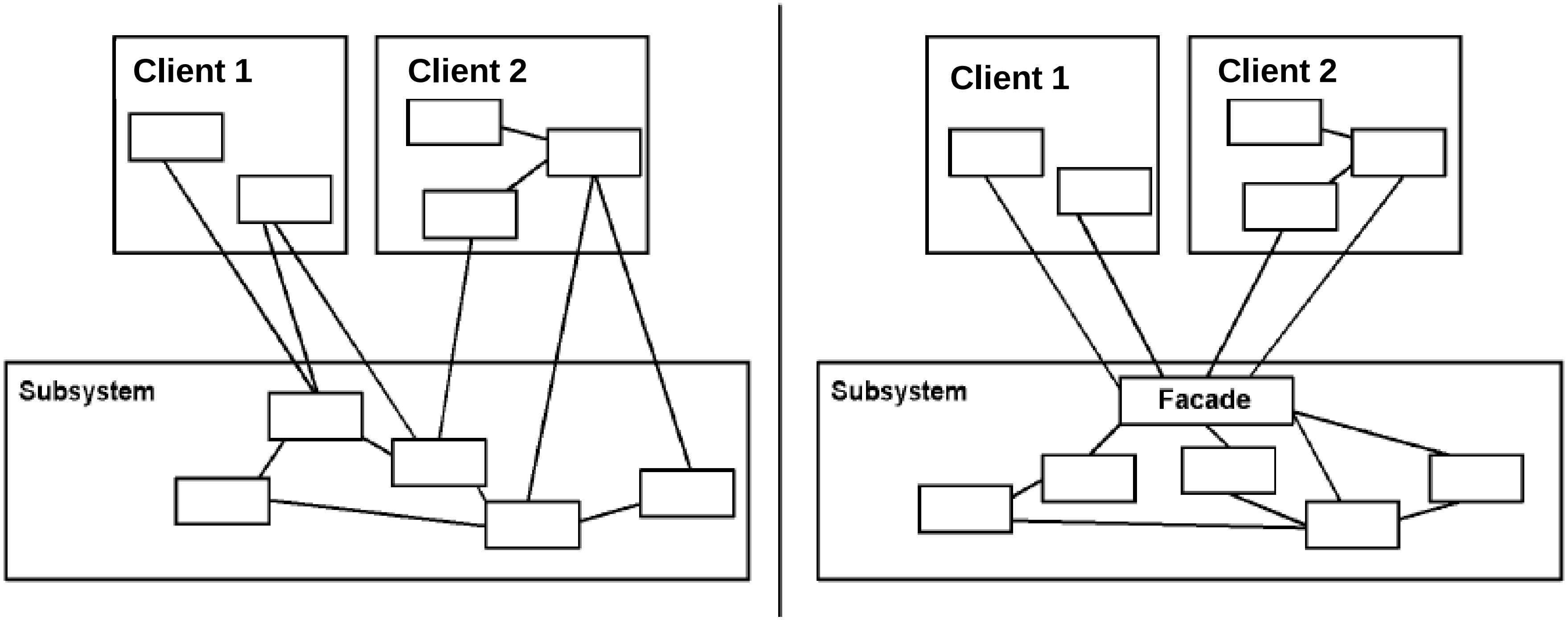


Cohesion in this Version?



Information Hiding via Facade

- Advantages and disadvantages of both systems?
- Which other principles can be recognized here?



Interface Abstraction

- Find a good balance between general and specific interfaces
- Segregation of interfaces
  - No client should be forced to depend on methods it does not use
  - Split generic interfaces into more specific ones such that clients only need to know about methods they need
  - Keep a system decoupled, achieve loose coupling
- Design by Contract
  - Specify pre-/postconditions, invariants of an interface
  - Currently: Apache Assertions, Google Guava preconditions

Open and Closed Components

- Open for change
- Closed for access to internal details by other components
- Achieve openess through design for change
- Achieve closedness through interface abstraction and information hiding

Components can be Coupled through …

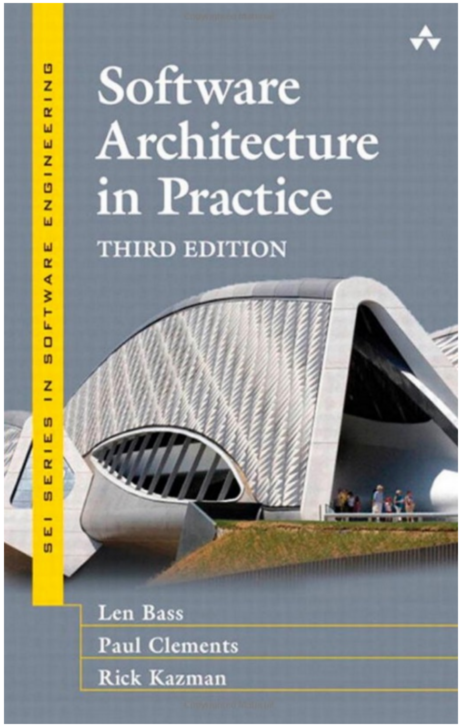
- Calls
- Creation/Instantiation
- Data dependencies
- Hardware or runtime environments
- Temporal dependencies

➢ Thoroughly review these couplings and arrive at decisions that reduce couplings to the required necessary minimum

Tactics

"There are many ways to do design badly, and just a few ways to do it well."

Bass, Clements, Kazman
Software Architecture in Practice



Tactics

- **A tactic is a design decision that influences the realization of the response of a quality attribute scenario**

- Specific technical solutions that help to achieve desired system qualities
  – E.g. Undo Command für Usability

- Determine the response of the system to react to a stimulus
  – Each tactic uses one specific structure or mechanism
  – Ignores trade-offs & compromises

Example: System Availability

Availability	Downtime/90 Days	Downtime/Year
99.0 %	21 hr, 36 min	3 days, 15.6 hr
99.9 %	2 hr, 10 min	8 hr, 0 min, 46 sec
99.99 %	12 min, 58 sec	52 min, 34 sec
99.999 %	1 min, 18 sec	5 min, 15 sec
99.9999 %	8 sec	32 sec

$$\alpha = MTBF / (MTBF + MTTR)$$

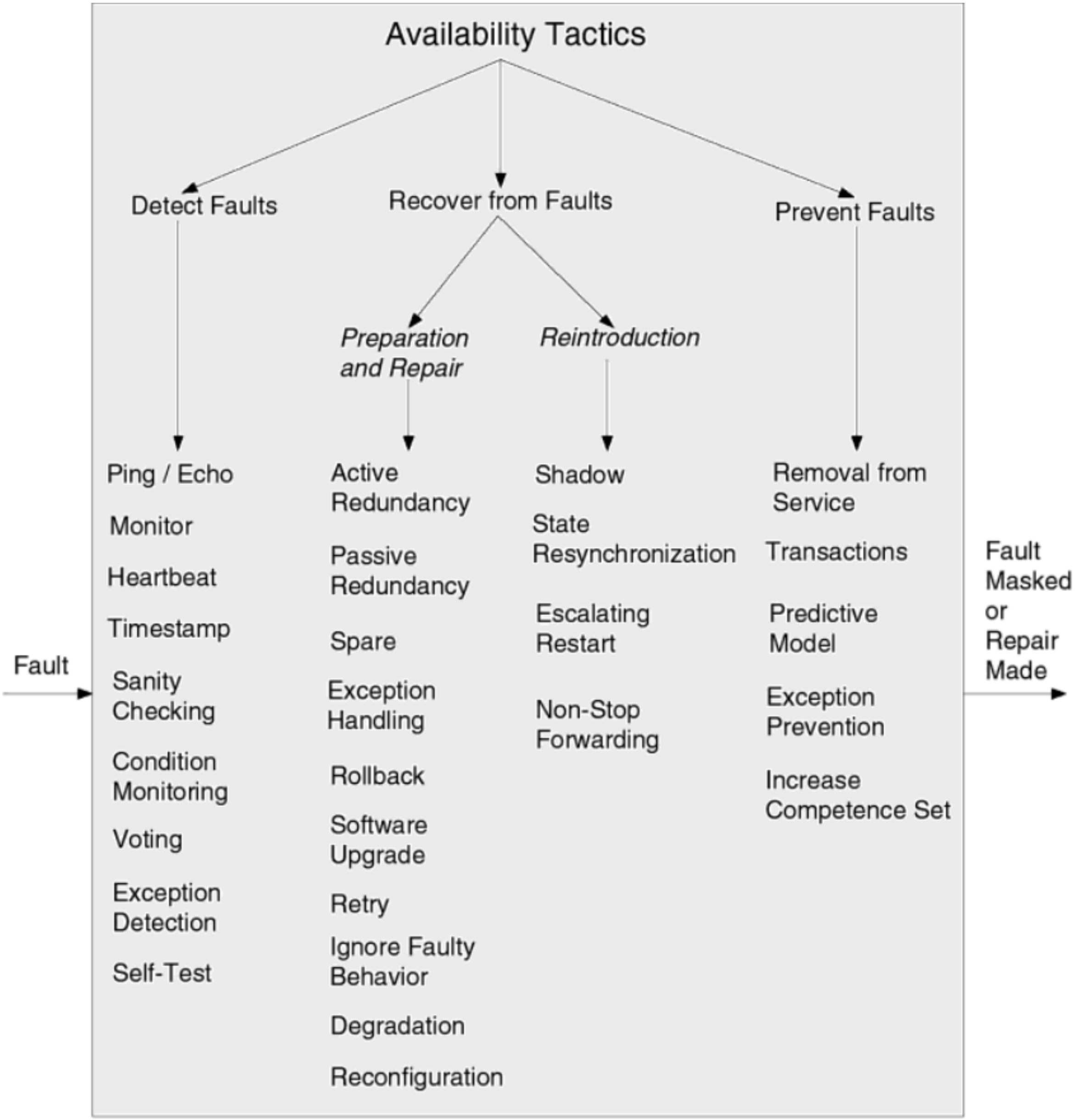
(steady-state) availability α
MTBF - mean time between failures
MTTR- mean time to repair

Scott/Kazman: Realizing and Refining Architectural Tactics: Availability
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9087

How to Achieve Availability?

- Need to achieve high MTBF and low MTTR
  - prevent faults
  - detect faults quickly
  - recover quickly and reliably
- Failure: System no longer provides a service or does not provide it as specified and expected
  - the system fails and the failure can be perceived by the actors acting in/with the system
- Fault: A defect with the potential to trigger a failure
- ➢ Availability tactics focus on building systems that can withstand faults or intercept faults in such a way that they do not become failures
  - minimal tactics: limit effects of failures and facilitate repair

Tactics for Availability



Decisions Specific wrt. Availability I

1. **Allocation of Responsibilities**
   – What must be highly available?
   – Are there any responsibilities in the system that can be used to determine faults and failures?
     • logging, notification, disabling fault causing events, be temporarily unavailable, fix/mask the fault/failure, operate in degraded mode
2. **Coordination Model**
   – Can coordination mechanisms detect availability problems (e.g. guaranteed delivery of messages?)
   – Are system parts exchangeable?
   – Does the coordination work under limited operation?

Decisions Specific wrt. Availability II

3. Data Model

- Which data sources/data operations can cause Faults/Failure?
- Ensure that these sources/operations can be disabled, temporarily unavailable, fixed/masked
  - For example, cache write requests when a server is down and write later when server is up again

4. Mapping among architectural elements

- Which artifacts may produce a fault? Is the mapping/remapping of architectural elements flexible enough to recover, e.g. can data & processes be restored from backups?

5. Resource Management

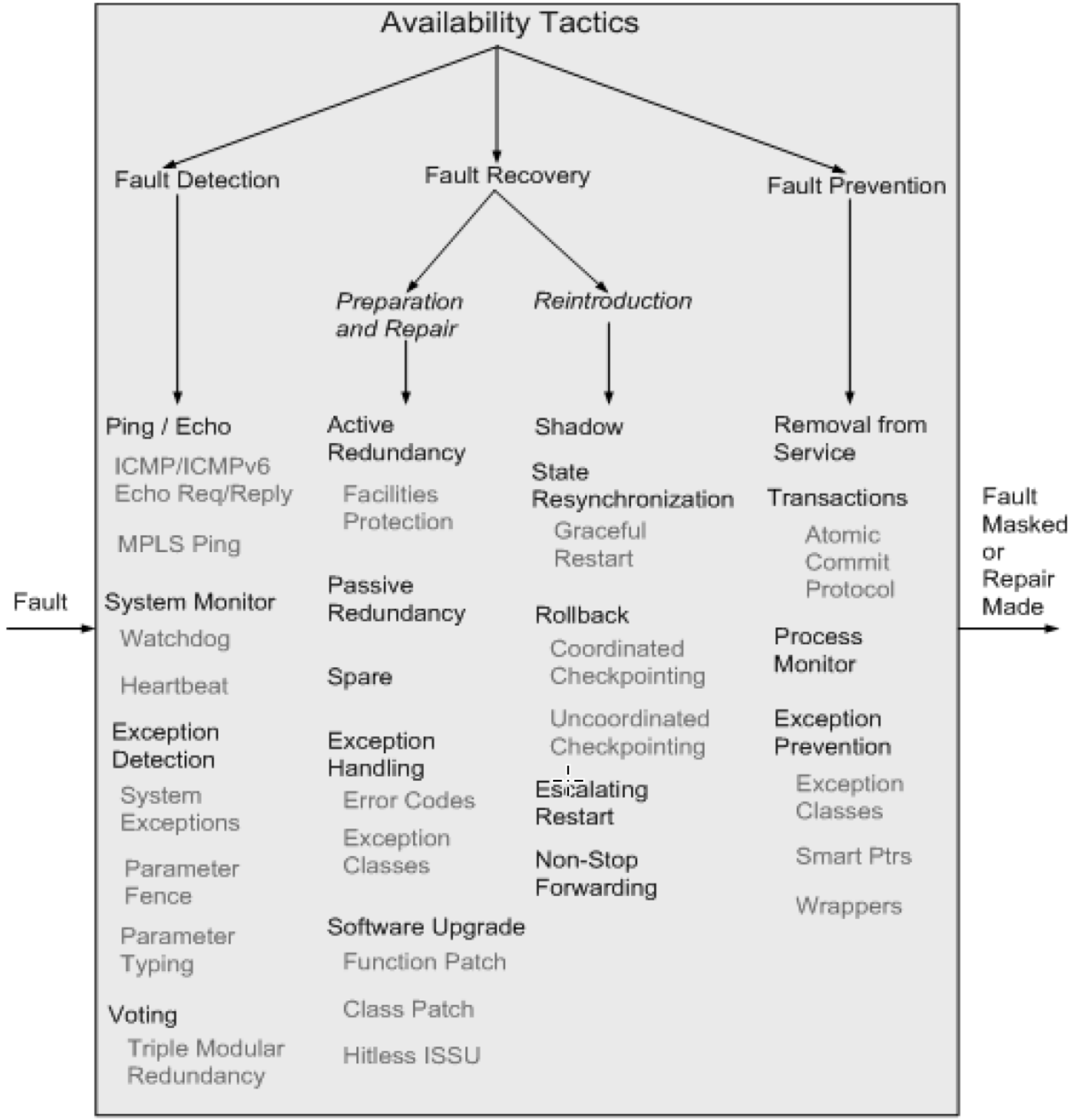
- Which critical resources must function in limited operation?

Decisions Specific wrt. Availability III

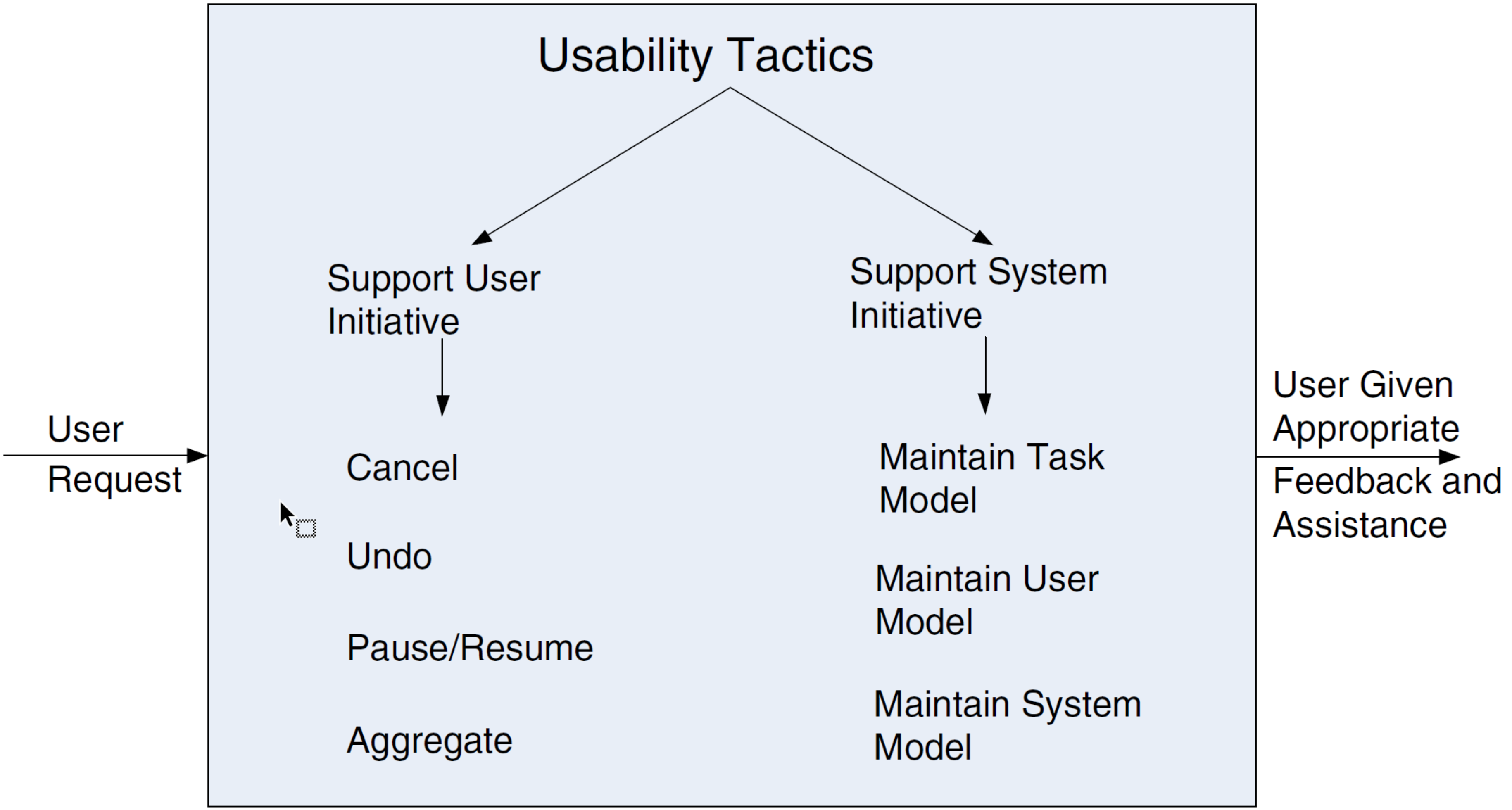
6. **Binding Time**
   - Is late binding used? What happens if involved components are affected by faults?
     - For example, how long can the response of a process be delayed until a fault must be anticipated?
7. **Choice of Technology**
   - What is available for logging, recovery, …
   - From which errors can the technology recover.
   - What faults can a technology bring into the system?

Refined Availability Tactics

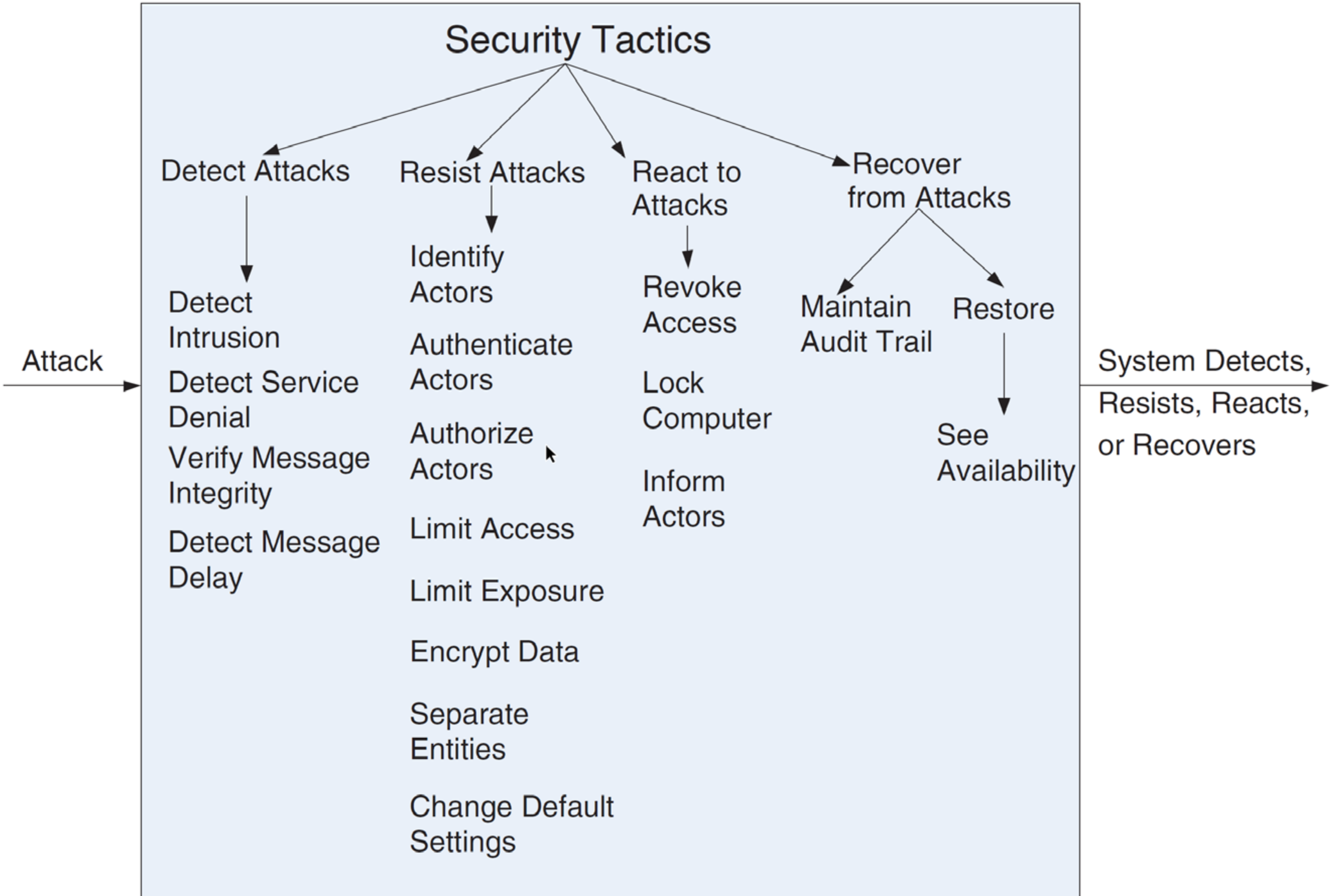
Scott/Kazman: Realizing and Refining Architectural Tactics: Availability https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9087



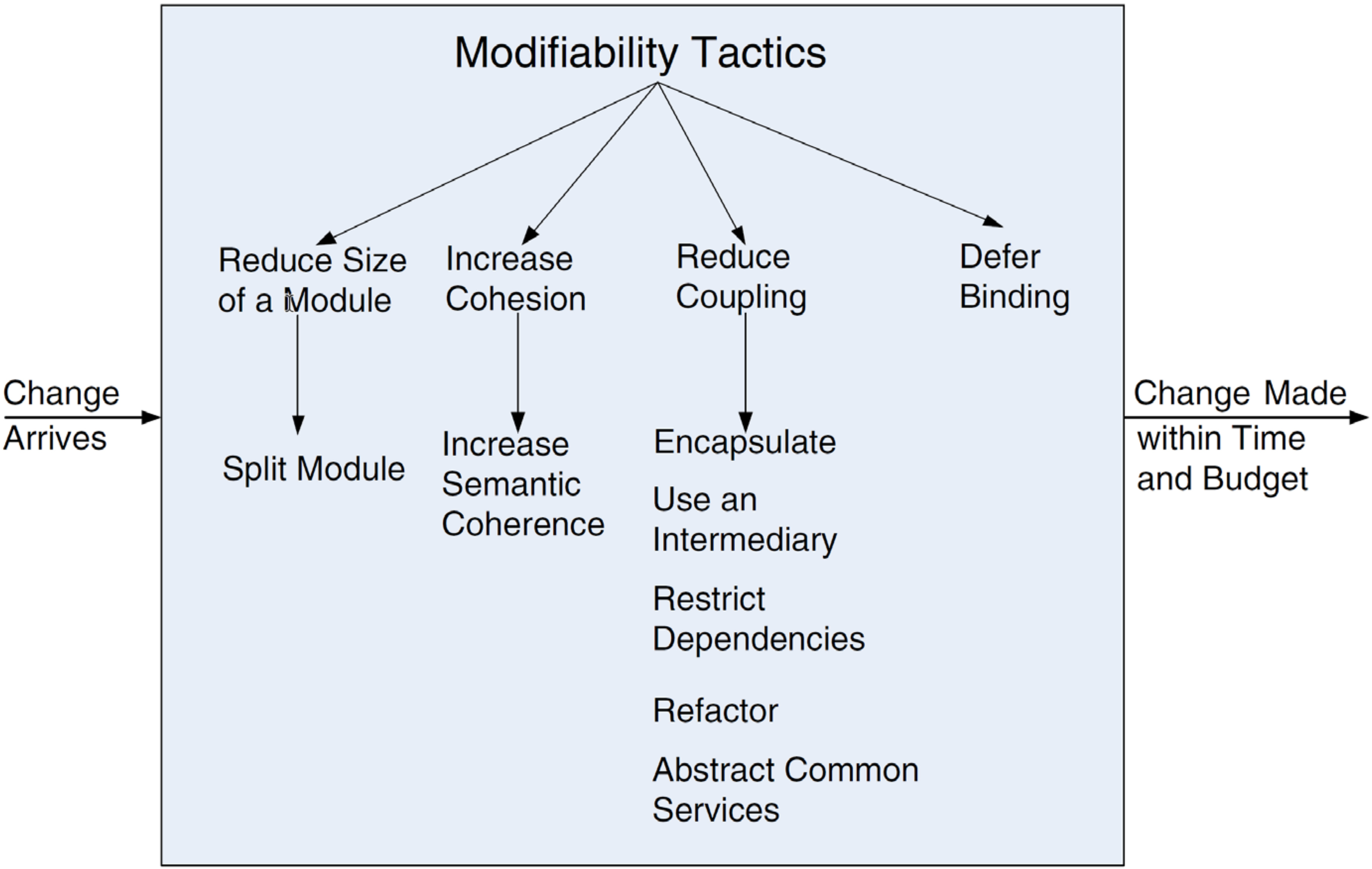
Usability


Usability Tactics
Support User Initiative
Support System Initiative
User Request
Cancel
Undo
Pause/Resume
Aggregate
Maintain Task Model
Maintain User Model
Maintain System Model
User Given Appropriate Feedback and Assistance

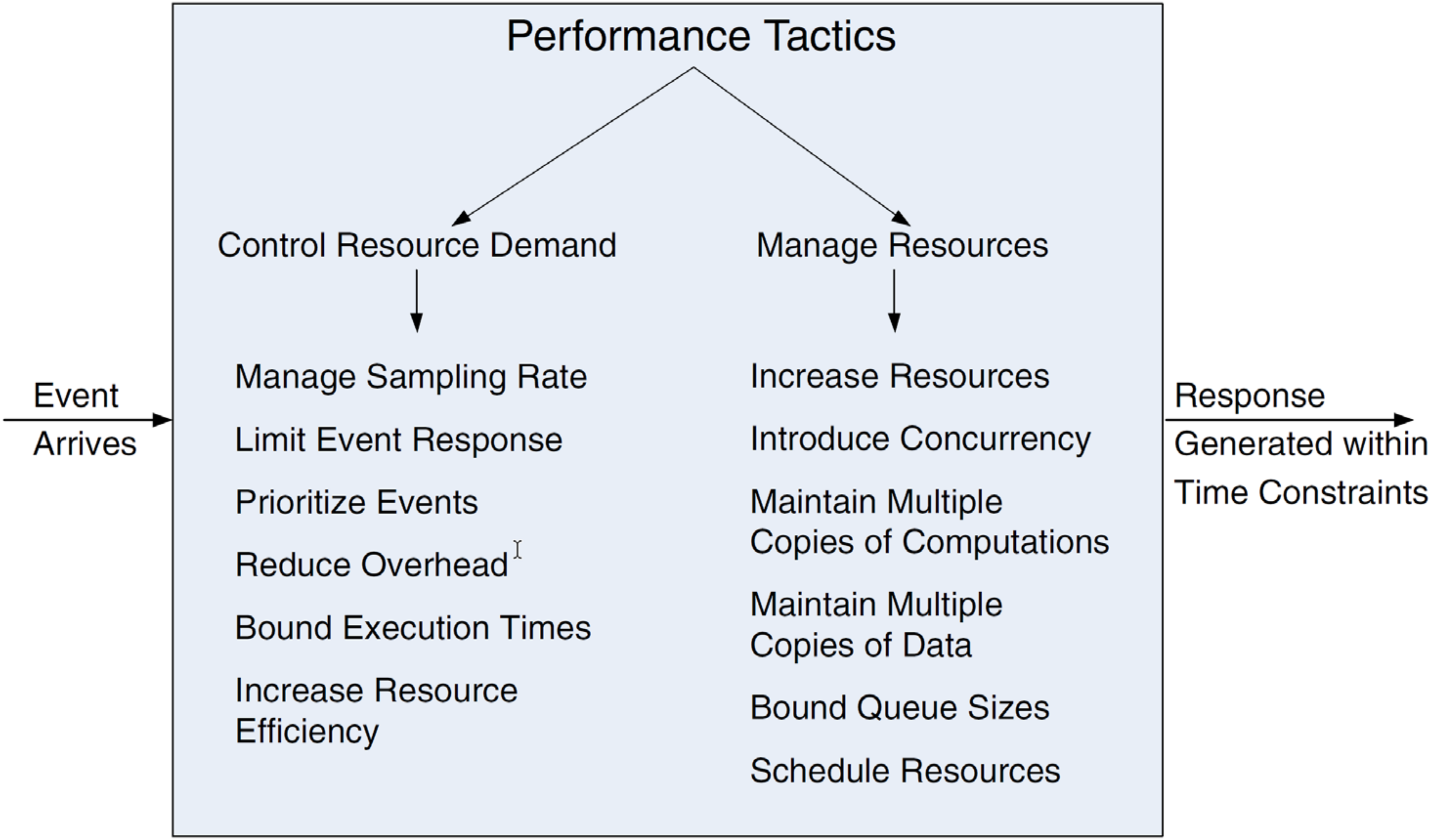
Security


Security Tactics
Attack
Detect Attacks
Resist Attacks
React to Attacks
Recover from Attacks
Detect Intrusion
Detect Service Denial
Verify Message Integrity
Detect Message Delay
Identify Actors
Authenticate Actors
Authorize Actors
Limit Access
Limit Exposure
Encrypt Data
Separate Entities
Change Default Settings
Revoke Access
Lock Computer
Inform Actors
Maintain Audit Trail
Restore
See Availability
System Detects, Resists, Reacts, or Recovers

Modifiability



Performance



Summary

- Principles and tactics as a basis to find a system structure to meet requirements
- Principles can support each other, must be made specific for a given architecture
- Tactics offer specific technical solutions to support a single specific quality attribute
- Architectural decisions are the responsibility of the architect
- Base decision on a prioritized list of quality attributes
- Work iteratively based on a close interaction with the development team, actively support agile development
- Constantly create and revise views for stakeholders
- Evaluate architectures early and repeat evaluation

Working Questions

1. What is an architectural principle?
2. Give examples of principles for a good architectural design and explain them.
3. Explain the relationship between 2 given architectural principles.
4. What are tactics?
5. What help tactics to achieve?
6. What do tactics NOT deal with?
7. Give examples of tactics and explain how a specific tactic helps to achieve a quality attribute of a system.

