ECE444: Software Engineering

Architecture2: Patterns, and Tactics

Shurui Zhou

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

About Milestone2

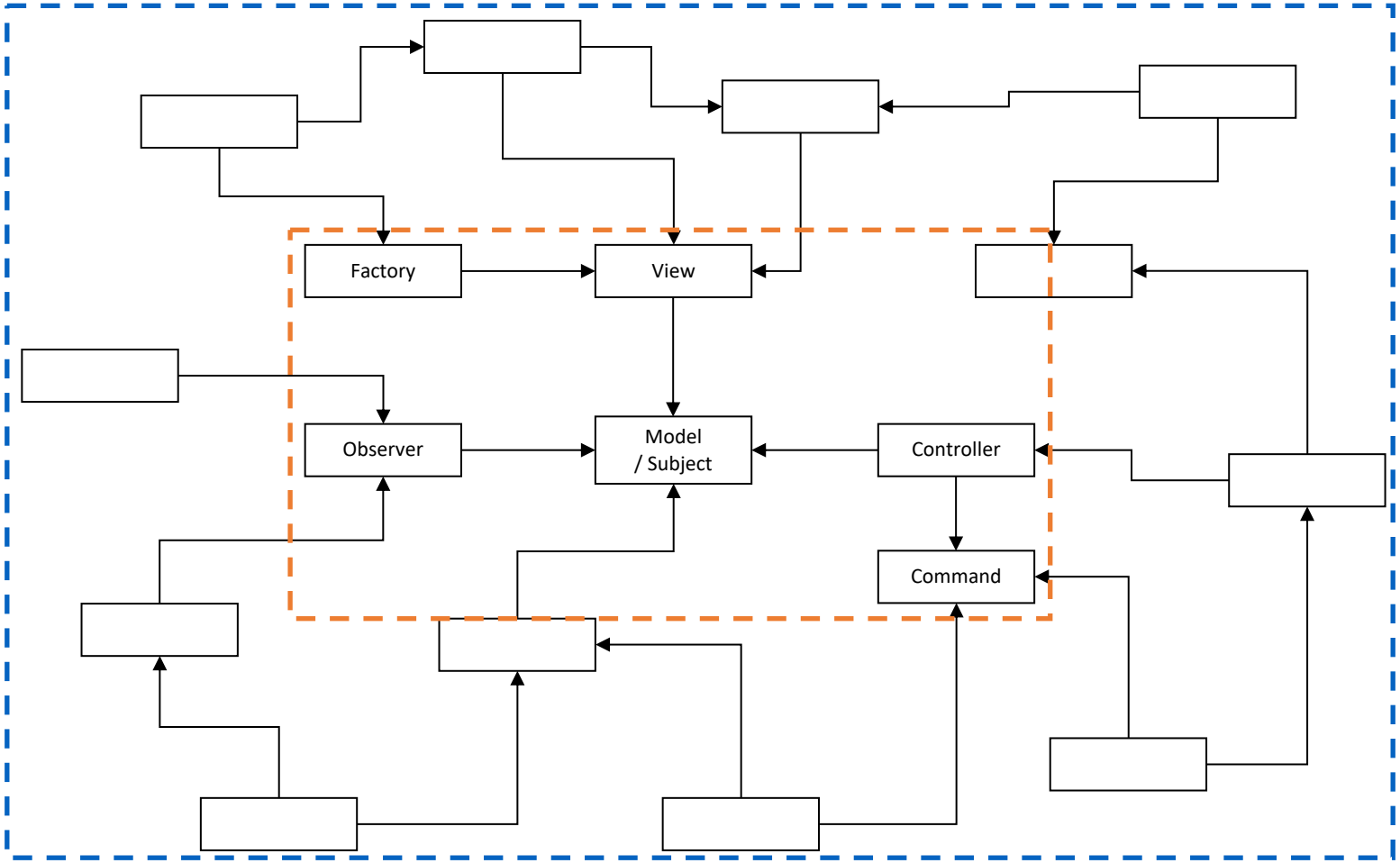
- About interview script, open&closed-ended questions, flow
- If you have questions, please schedule a meeting with me separately or join the office hour. (Fri 4-5pm)

Learning Goals

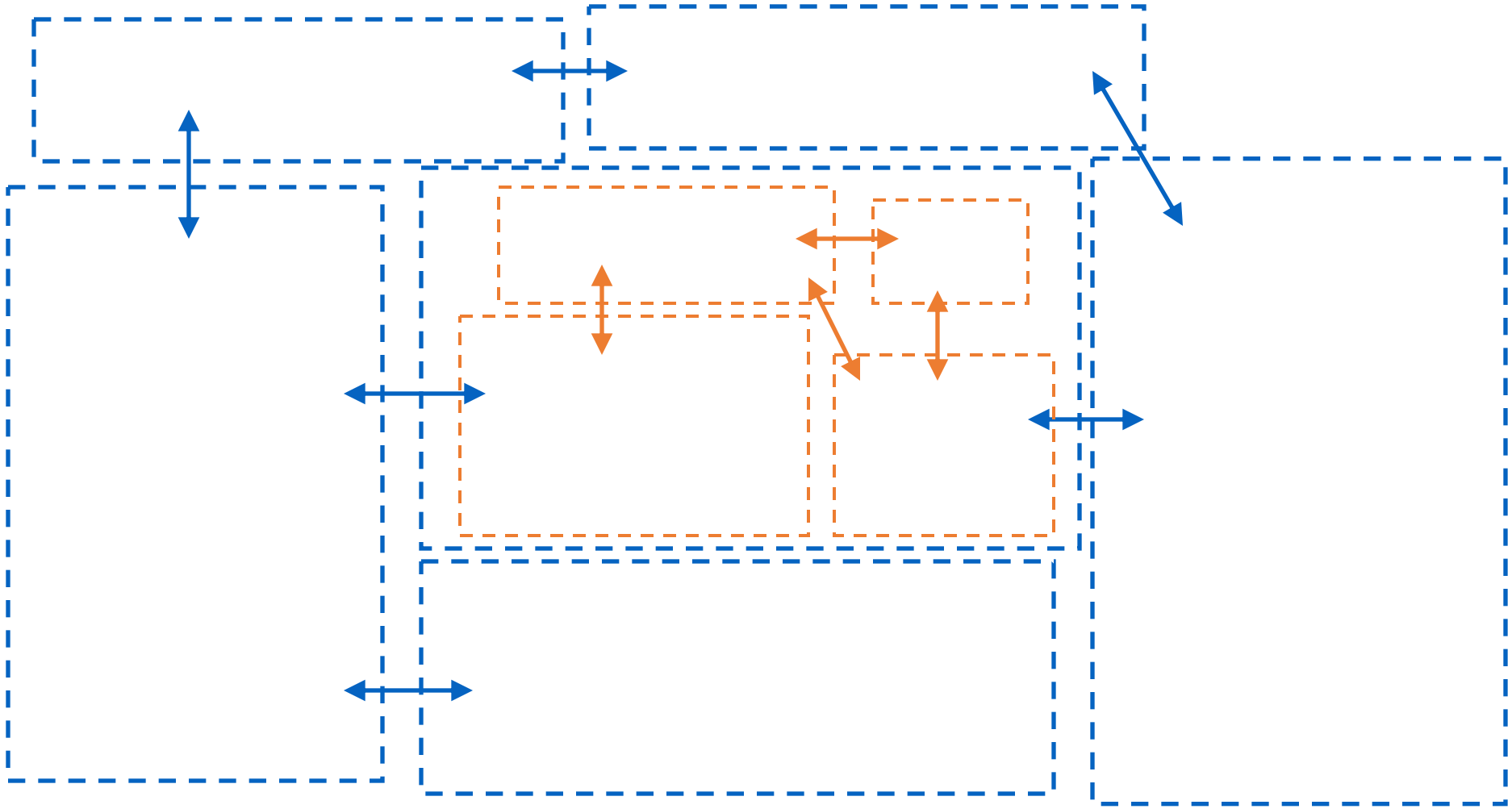
- Use diagrams to understand systems and reason about tradeoffs.
- Understand the utility of architectural patterns and tactics, and give a couple of examples.
- Understand Architecture in Agile and trade-offs

Architectural Tactics and Patterns

Design Patterns



Architecture



Common Views in Documenting Software Architecture

- **Modules (Static)**

Modules are assigned specific computational responsibilities, and are the basis of work assignments for programming teams

- **Dynamic** (Component-and-connector **C&C**)

Focus on the way the elements interact with each other at runtime to carry out the system's functions.

- **Allocation** (Physical, Deployment)

Mapping from software structures to the system's organizational, developmental, installation, and execution environments.

Architectural Patterns

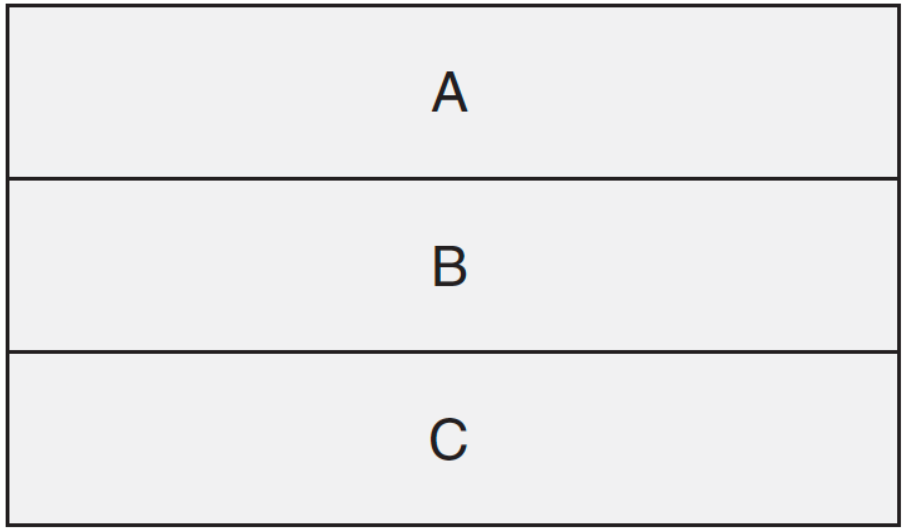
- Context + Problem + Solution
- Describes computational model
  - E.g., pipe and filter, call-return, publish-subscribe, layered, services
- Related to one of common view types
  - Static, dynamic, physical
- For example: a web-based system
  - 3-tier client server architectural pattern + replication, proxies, caches, firewalls, MVC, etc.

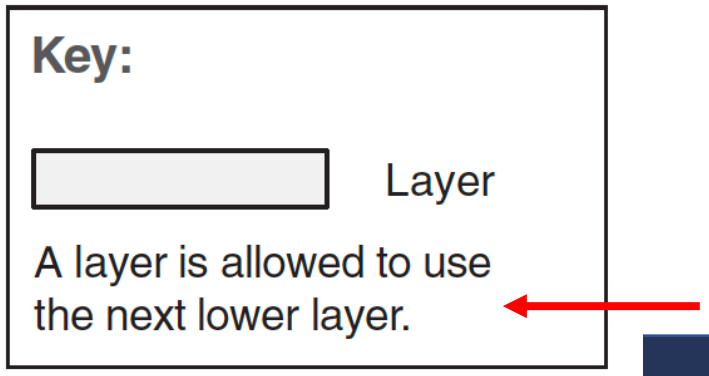
Example Architectural Patterns

- **Modules (Static)**
  - Layered Pattern
- **Dynamic** (Component-and-connector **C&C**)
  - Broker Pattern
  - MVC (Model-View-Controller) Pattern
  - Client-Server Pattern
- **Allocation** (Physical, Deployment)
  - Map-Reduce Pattern
  - Multi-tier Pattern

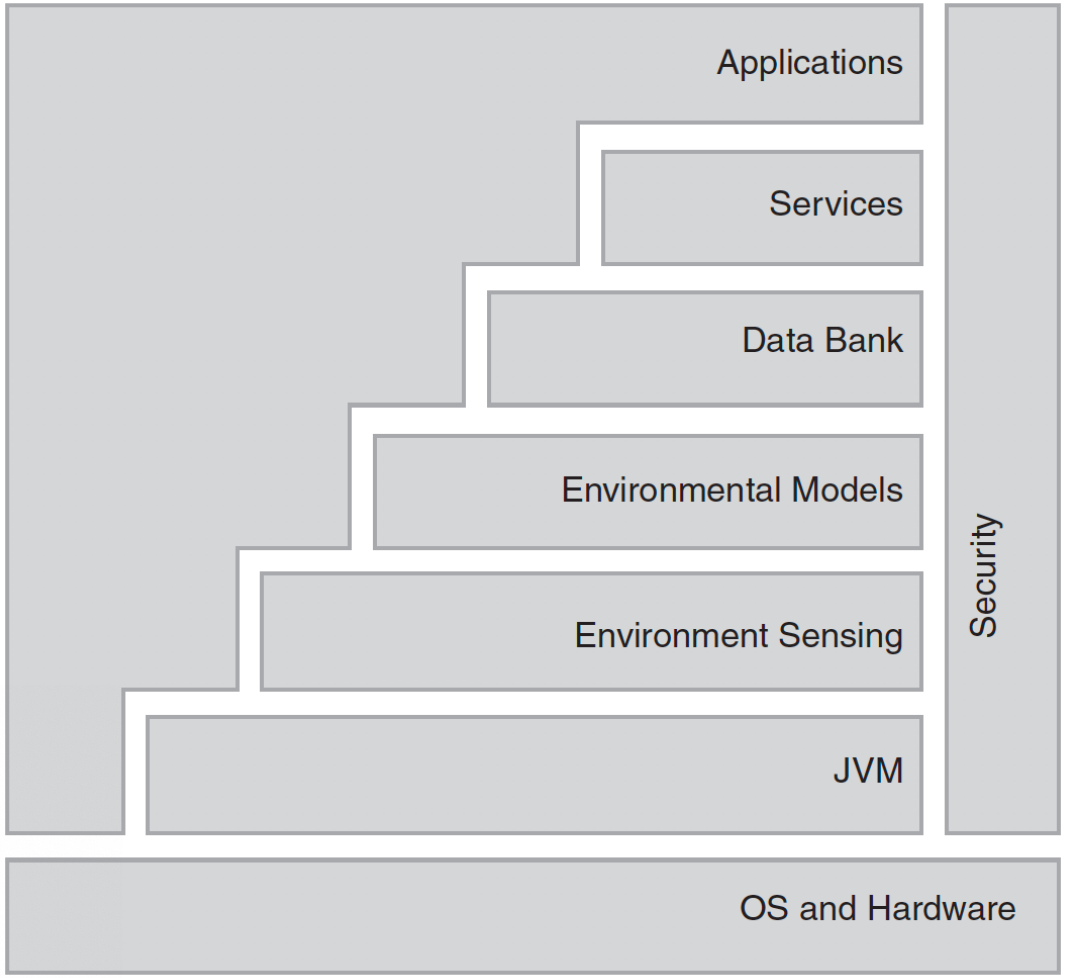
Layered Pattern

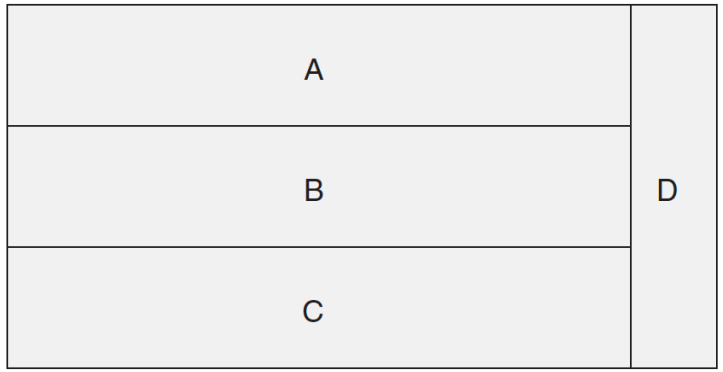
- Separation of concerns
- Constraints on the allowed-to-use relationship among the layers, the relations must be unidirectional
- Normally only next-lower-layer uses are allowed
- “above” and “below” matter





Layered Pattern





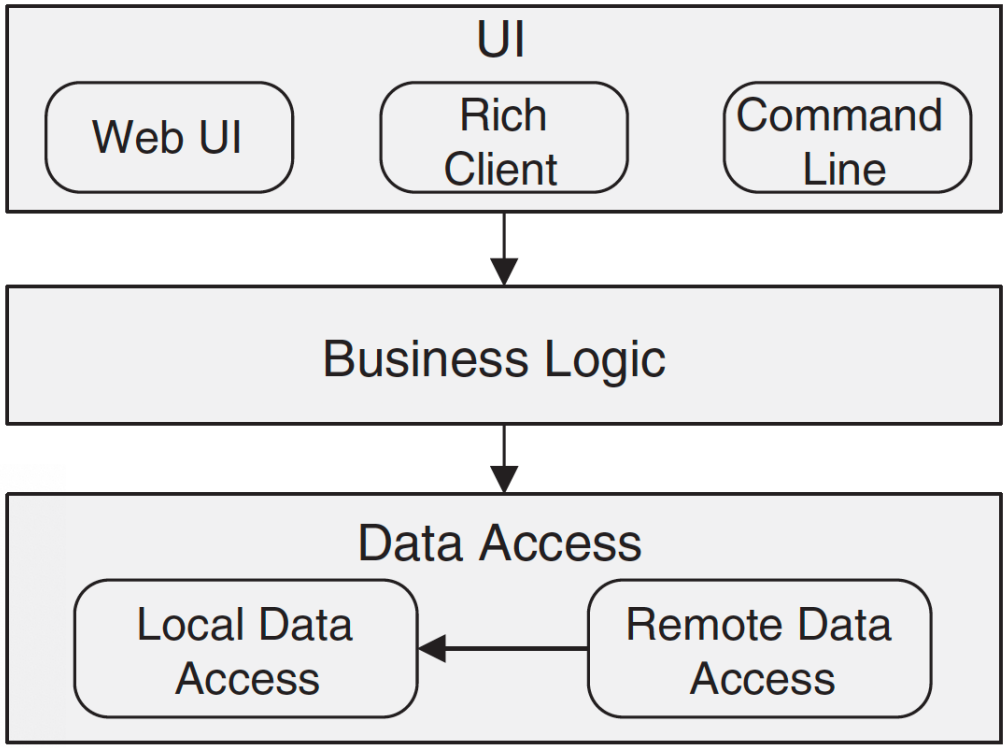
Layers with a “sidebar”

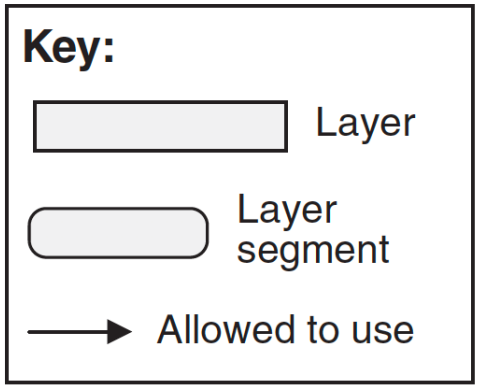
Key:

layer

Software in a layer is allowed to use software in the same layer, or any layer immediately below or to the right.

Layered Pattern





Layered design with segmented layers

Example Architectural Patterns

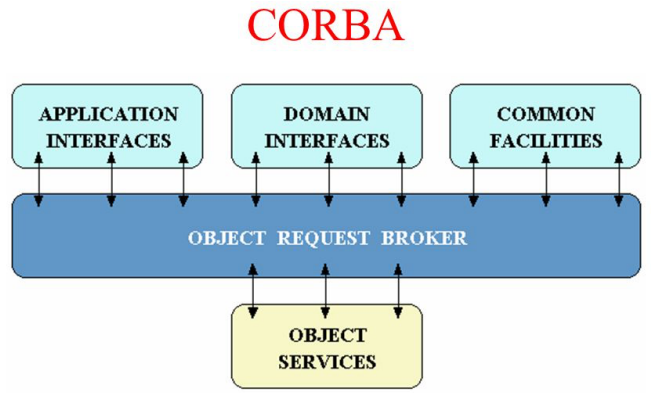
- **Modules (Static)**
  - Layered Pattern
- **Dynamic** (Component-and-connector **C&C**)
  - Broker Pattern
  - MVC (Model-View-Controller) Pattern
  - Client-Server Pattern
- **Allocation** (Physical, Deployment)
  - Map-Reduce Pattern
  - Multi-tier Pattern

Broker Pattern

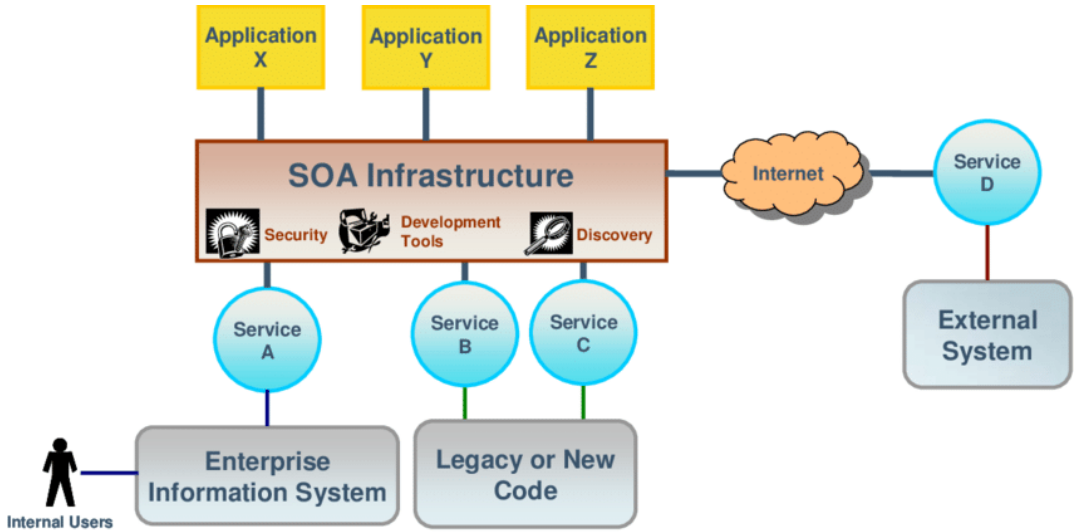
- A collection of services distributed across multiple servers
- Separates users of services (clients) from providers of services (servers) by inserting an intermediary, called a *broker*
- *Proxies* are commonly introduced as intermediaries in addition to the broker
- Benefit: modifiability, availability, performance
- Downside: add complexity, latency

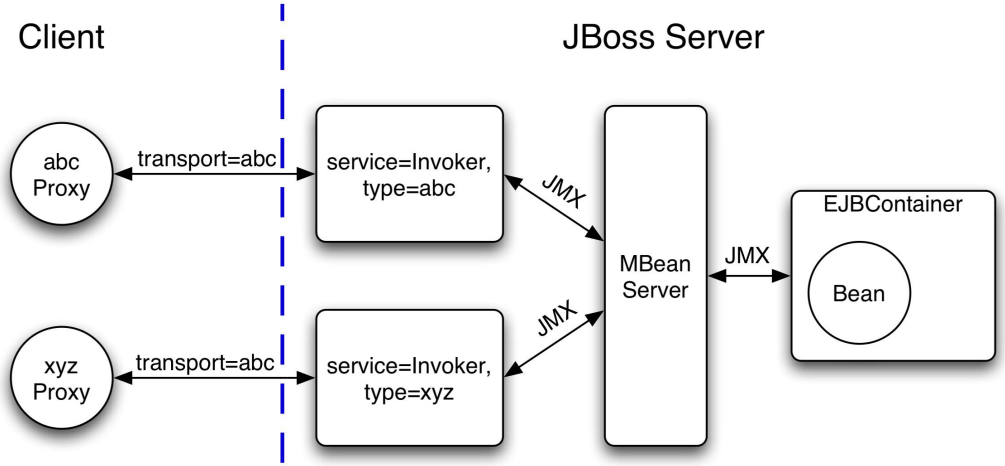
Real-world Application

- Common Object Request Broker Architecture (CORBA)
- Enterprise Java Beans (EJB)
- Microsoft's .NET platform
- SOA - Service-Oriented Architecture



OMG Reference Model architecture





Example Architectural Patterns

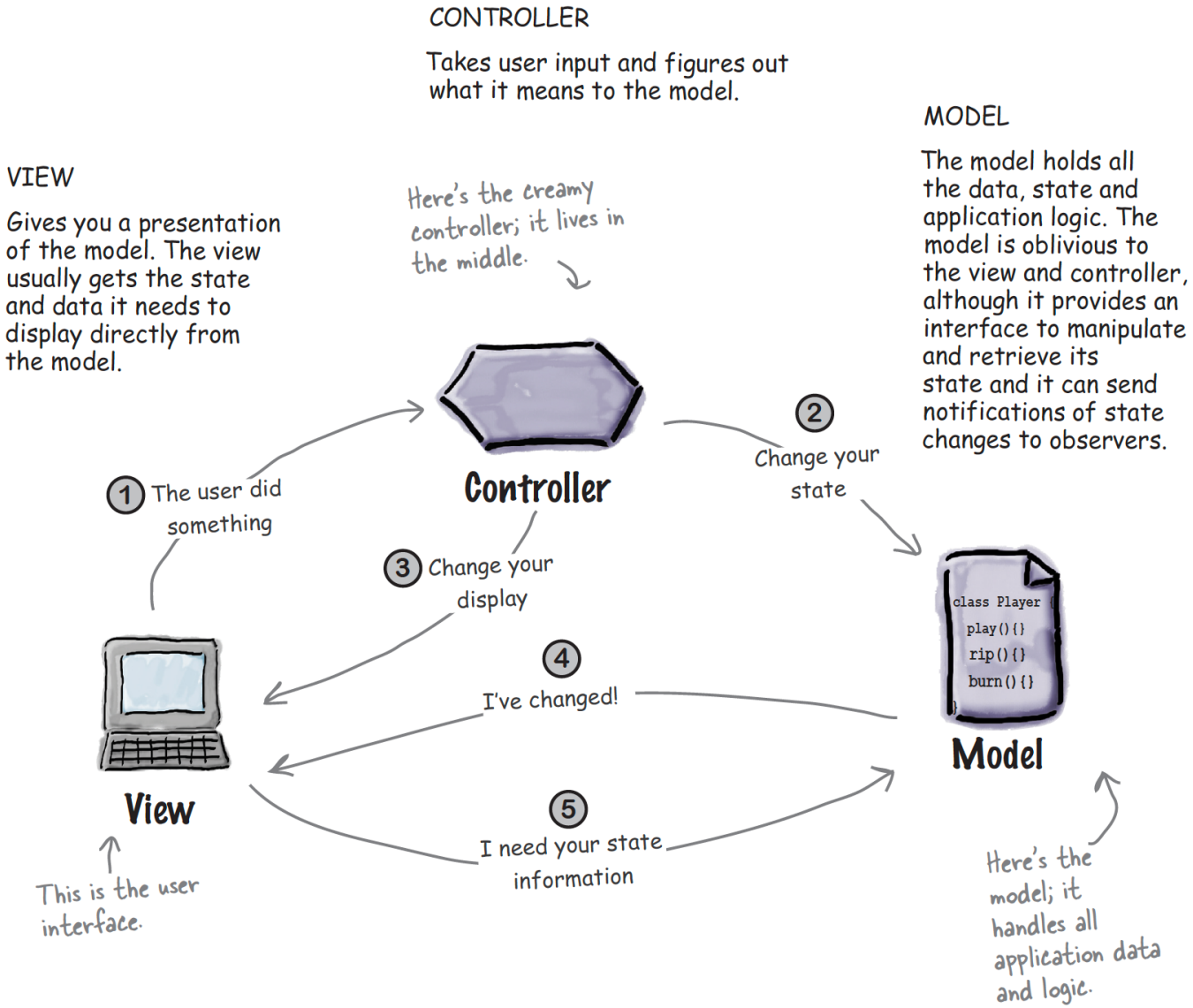
- **Modules (Static)**
  - Layered Pattern
- **Dynamic** (Component-and-connector **C&C**)
  - Broker Pattern
  - MVC (Model-View-Controller) Pattern
  - Client-Server Pattern
- **Allocation** (Physical, Deployment)
  - Map-Reduce Pattern
  - Multi-tier Pattern

MVC (Model-View-Controller) Pattern

- Separate UI functionality from the application functionality
- Multiple views of the user interface can be created, maintained, and coordinated when the underlying application data changes

Component	Description
Model	• Handles application data and data-management • Central component of MVC
View	• Can be any output representation of information to user • Renders data from model into user interface
Controller	• Accepts input and converts to commands for model/view

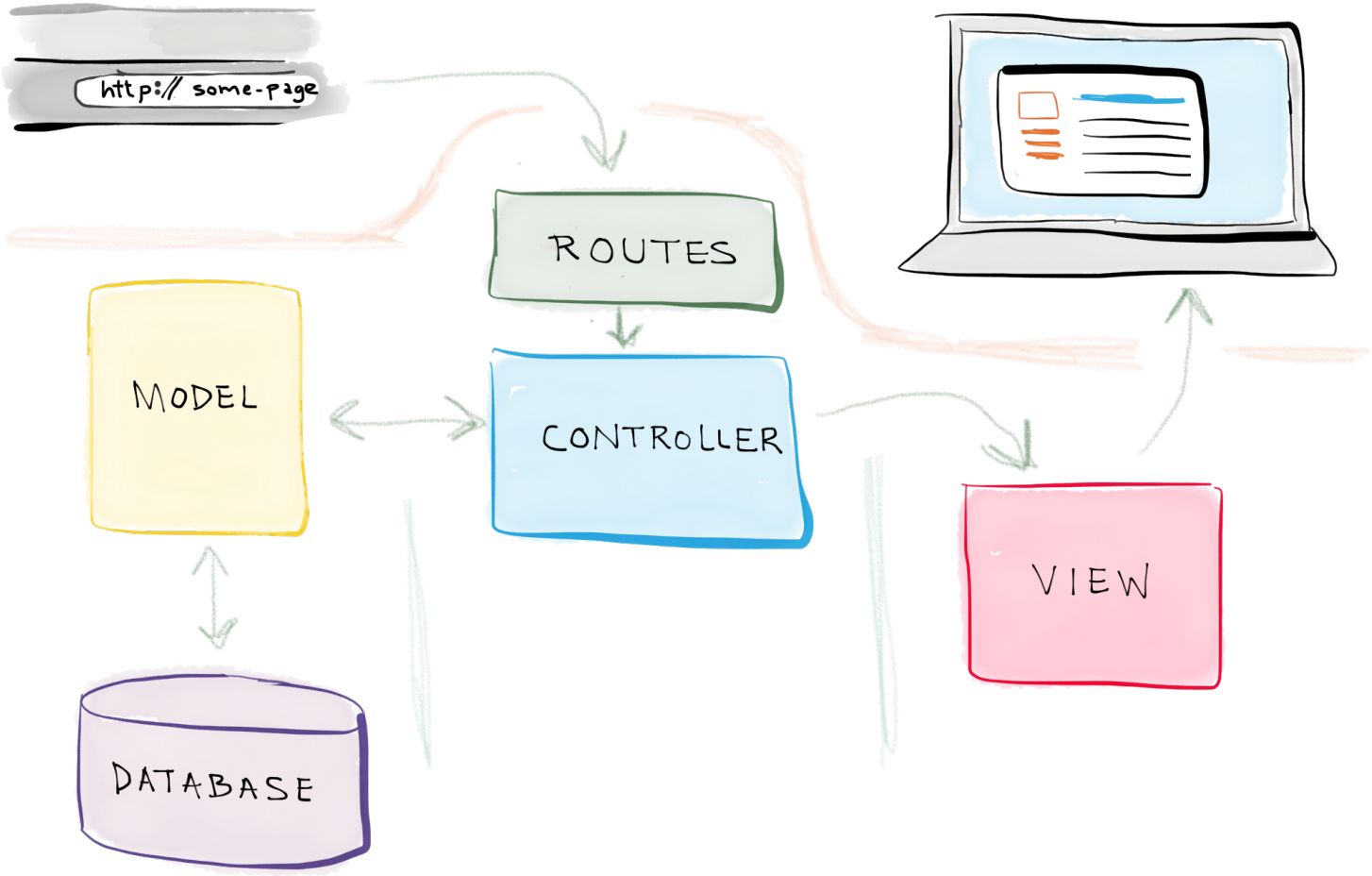
Example: MP3 player



Head_First_Design_Patterns (Chapter 12)

https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos/

MVC and the Web



MVC (Model-View-Controller) Pattern

- Weaknesses: The complexity may not be worth it for simple user interfaces.

Real-world Application

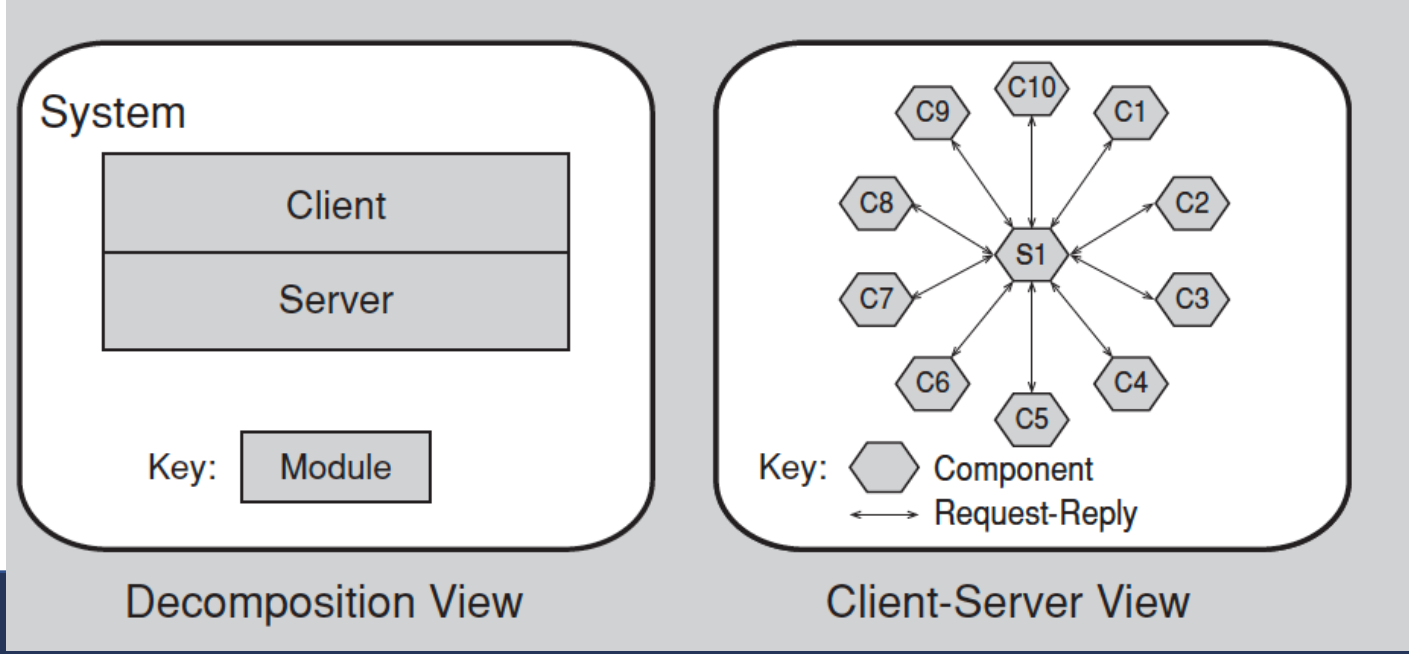
- Java's Swing classes
- ASP.NET
- Adobe's Flex software Development kit
- Nokia's Qt framework
- Flask + MVC
  - https://alysivji.github.io/flask-part2-building-a-flask-web-application.html
  - https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos/

Example Architectural Patterns

- **Modules (Static)**
  - Layered Pattern
- **Dynamic** (Component-and-connector **C&C**)
  - Broker Pattern
  - MVC (Model-View-Controller) Pattern
  - Client-Server Pattern
- **Allocation** (Physical, Deployment)
  - Map-Reduce Pattern
  - Multi-tier Pattern

Client-Server Pattern

- Context: There are shared resources and services that large numbers of distributed clients wish to access, and for which we wish to control access or quality of service.
- Modifiability, Reuse, Scalability, Availability
- Asymmetric or Synchronous



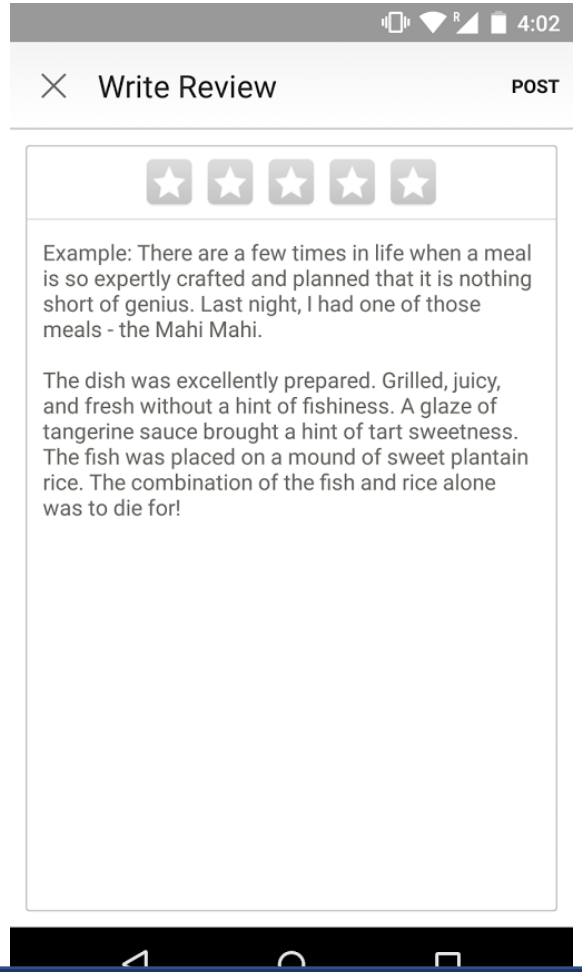
Client-Server Pattern

Disadvantages:

- the server can be a performance bottleneck and it can be a single point of failure
- decisions about where to locate functionality (in the client or in the server) are often complex and costly to change after a system has been built.

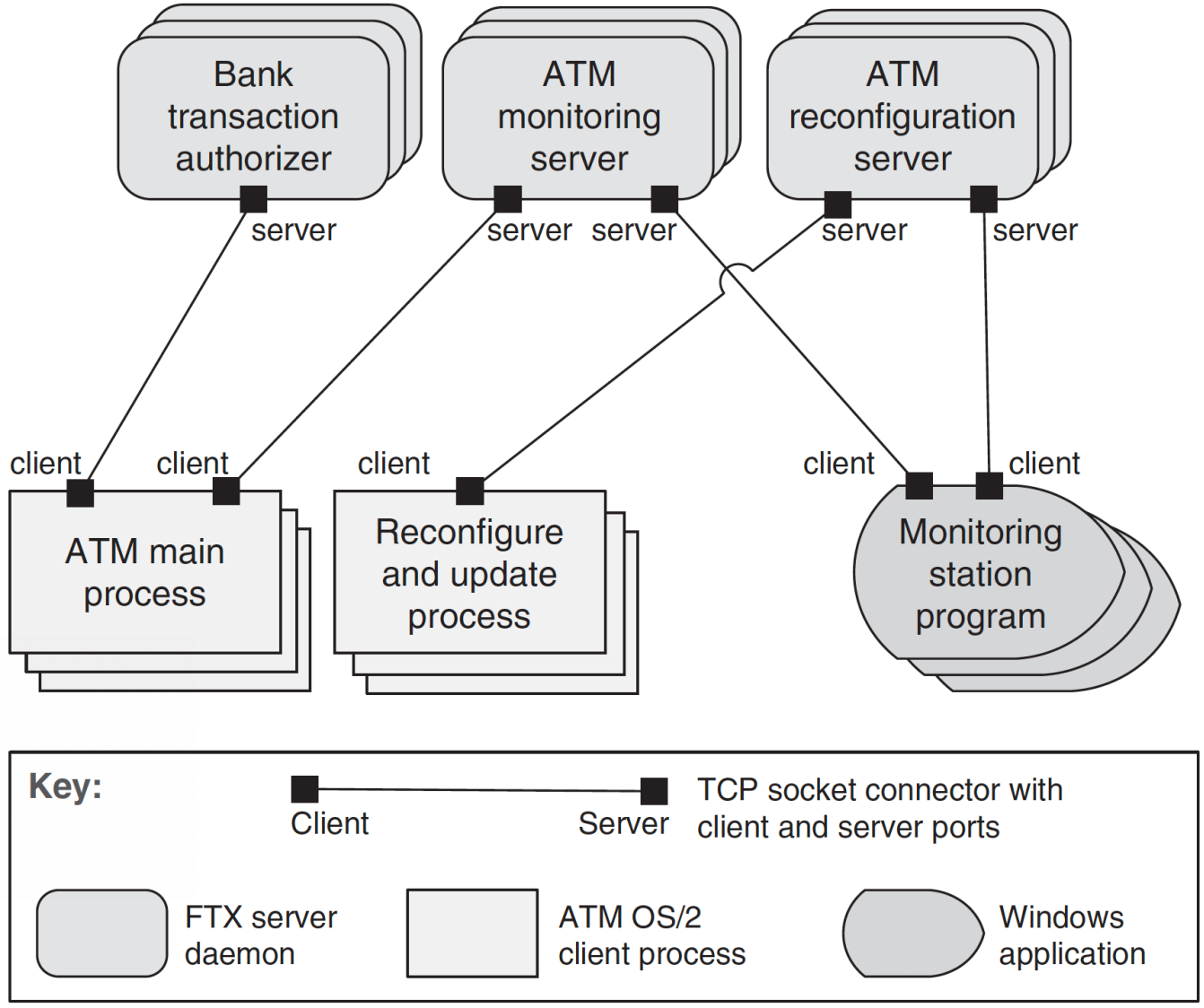
Where to validate user input?

Example: Yelp App



Real-world Example

- www
- ATM



Example Architectural Patterns

- **Modules (Static)**
  - Layered Pattern
- **Dynamic** (Component-and-connector **C&C**)
  - Broker Pattern
  - MVC (Model-View-Controller) Pattern
  - Client-Server Pattern
- **Allocation** (Physical, Deployment)
  - Map-Reduce Pattern
  - Multi-tier Pattern

Map-Reduce Pattern

- **Context**:
  - Petabyte scale of data → Programs for the analysis of this data should be easy to write, run efficiently, and be resilient with respect to hardware failure.
- **Solution**
  - a specialized infrastructure takes care of allocating software to the hardware nodes in a massively parallel computing environment and handles sorting the data as needed.
  - map function
  - reduce function

Multi-tier Pattern

- **Context**: In a distributed deployment, there is often a need to distribute a system's infrastructure into distinct subsets. This may be for operational or business reasons (for example, different parts of the infrastructure may belong to different organizations)
- **Solution**: The execution structures of many systems are organized as a set of logical groupings of components. Each grouping is termed a tier. The grouping of components into tiers may be based on a variety of criteria, such as the type of component, sharing the same execution environment, or having the same runtime purpose.

Multi-tier Pattern

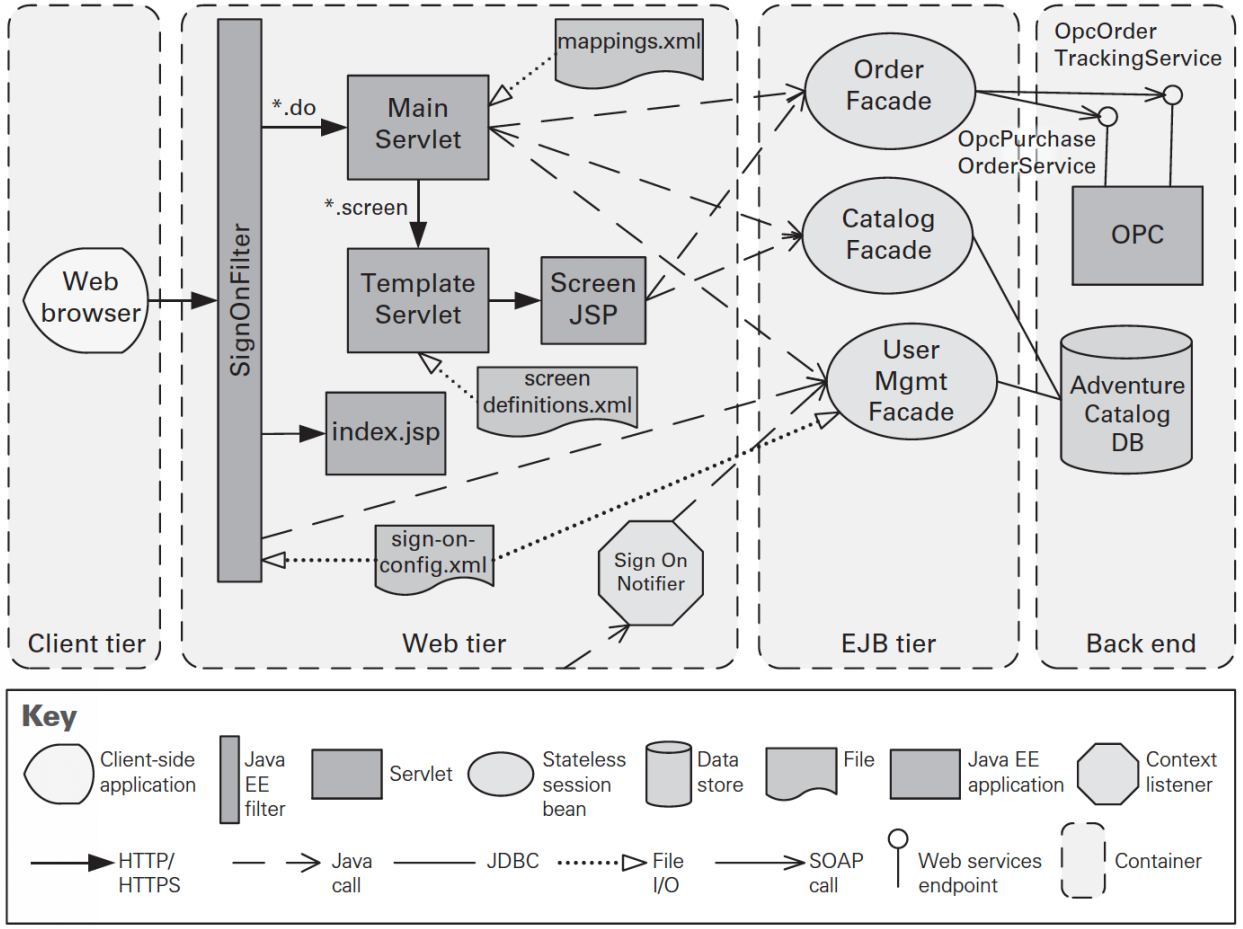


FIGURE 13.15 A multi-tier view of the Consumer Website Java EE application, which is part of the Adventure Builder system

Adventure Builder - Software Architecture Document (SAD)

Created by Todd Waits, last modified by Tamara L. Marshall-Keim on Sep 16, 2019

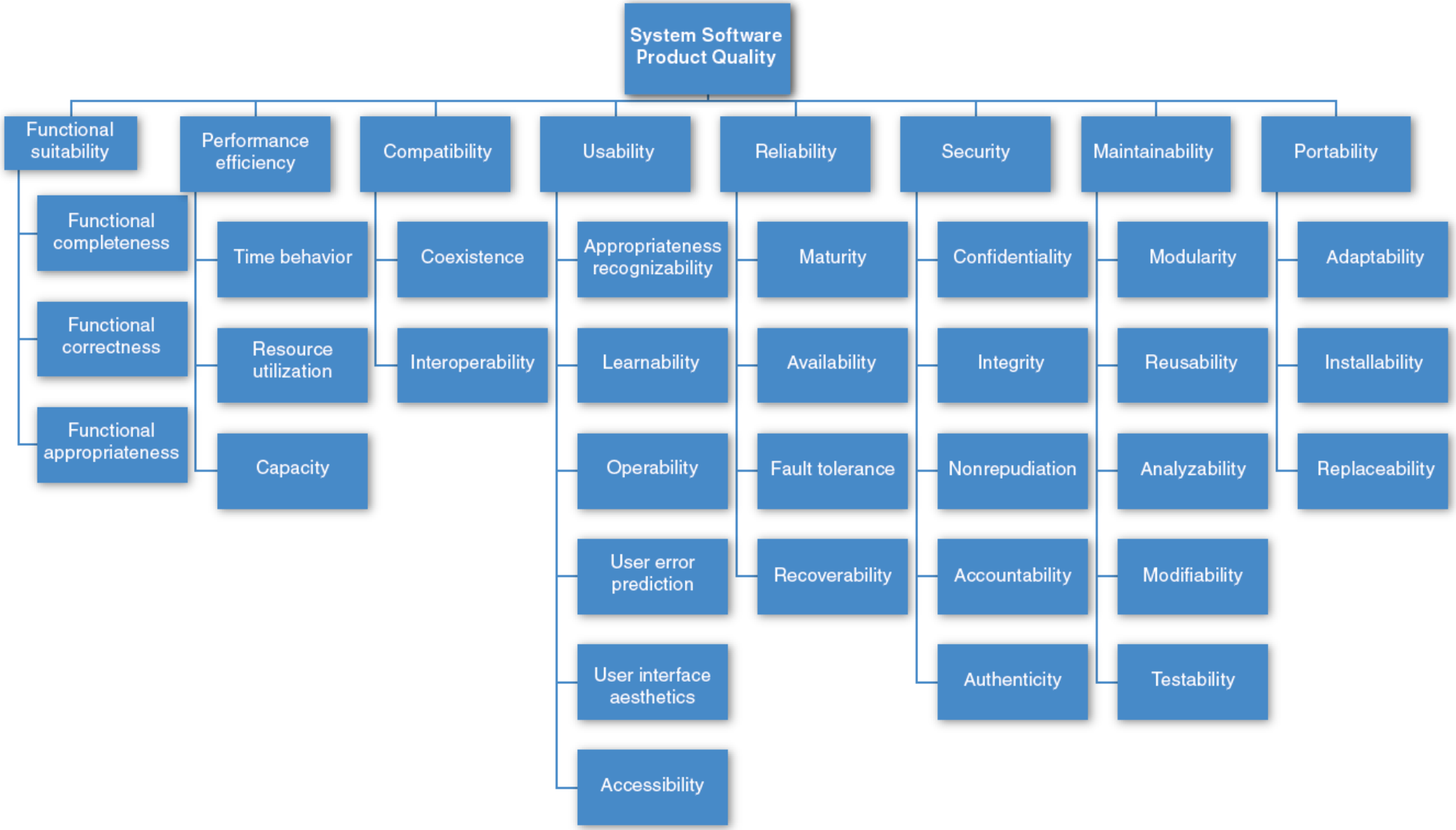
Contents

https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=146280205

Tactics

- Architectural techniques to achieve qualities
  - More tied to specific context and quality
- Smaller scope than architectural patterns
  - Problem solved by patterns: “How do I structure my (sub)system?”
  - Problem solved by tactics: “How do I get better at quality X?”
- Collection of common strategies and known solutions
  - Resemble OO design patterns

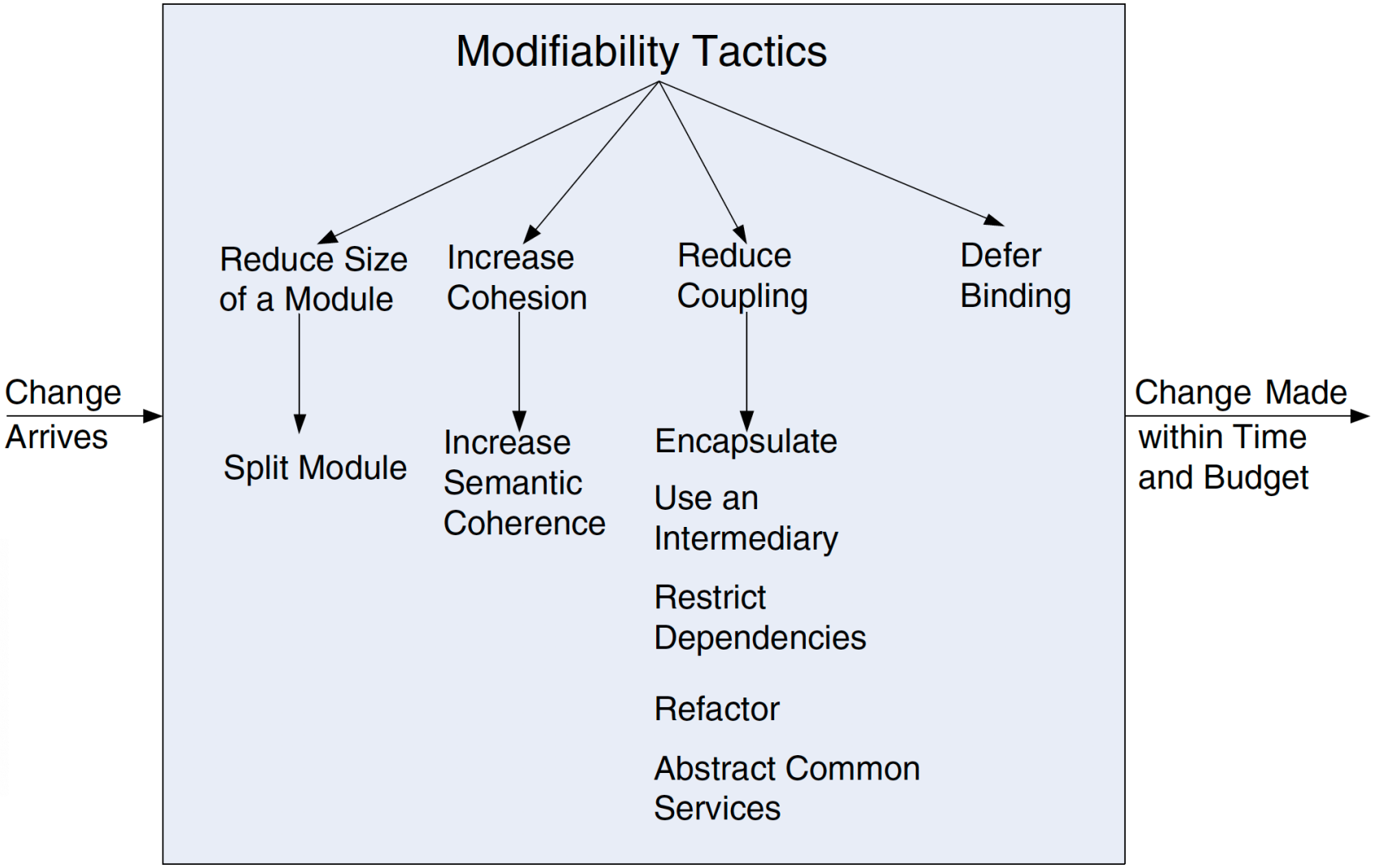
Achieving Quality Attributes through Tactics



Modifiability

Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, . . .
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: ▪ Make modification ▪ Test modification ▪ Deploy modification
Response Measure	Cost in terms of the following: ▪ Number, size, complexity of affected artifacts ▪ Effort ▪ Calendar time ▪ Money (direct outlay or opportunity cost) ▪ Extent to which this modification affects other functions or quality attributes ▪ New defects introduced

Modifiability



- coupling - probability that a modification to one module will propagate to the other
- cohesion - how strongly the responsibilities of a module are related

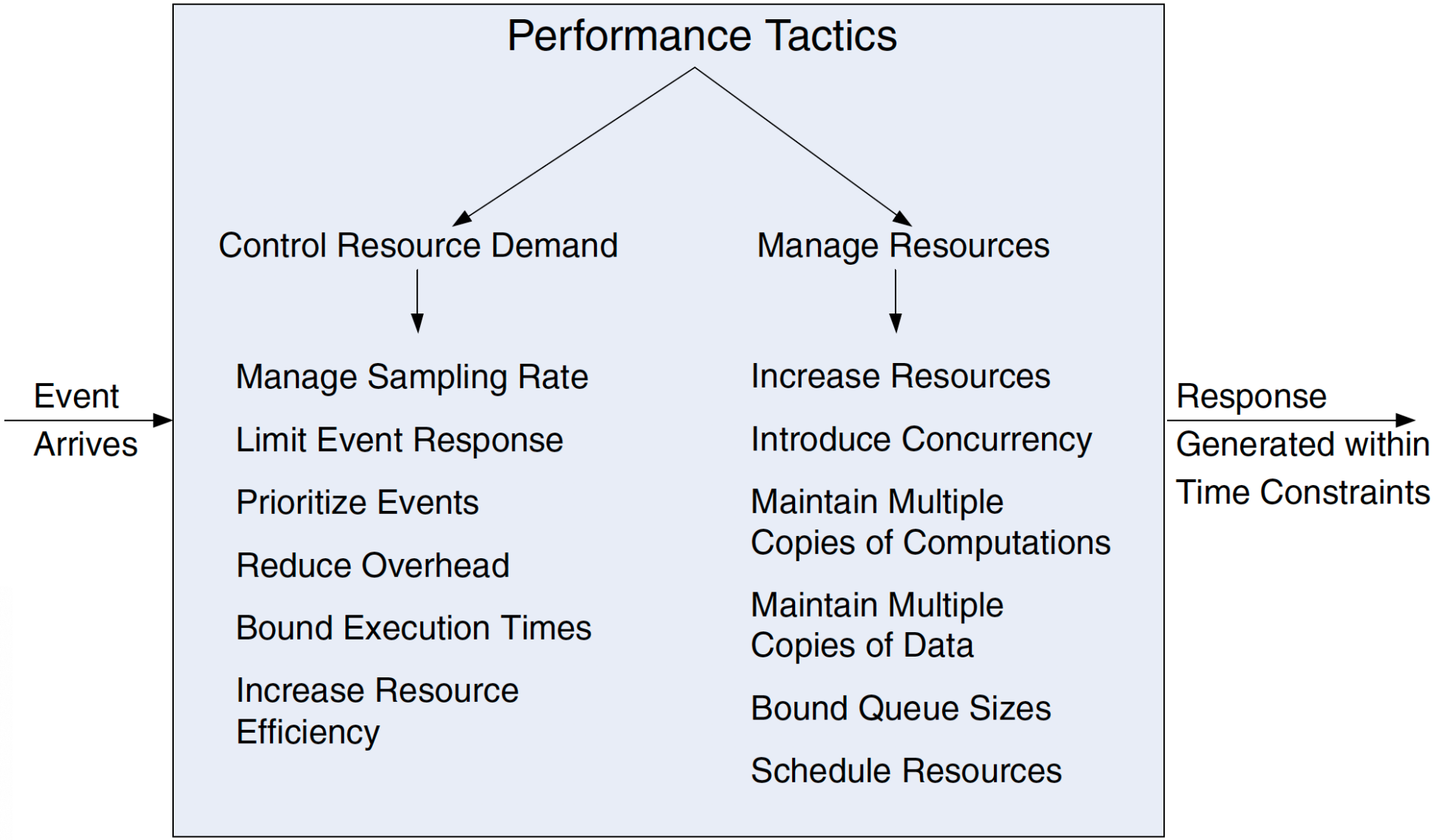
Low coupling, high cohesion, better modifiability

Performance

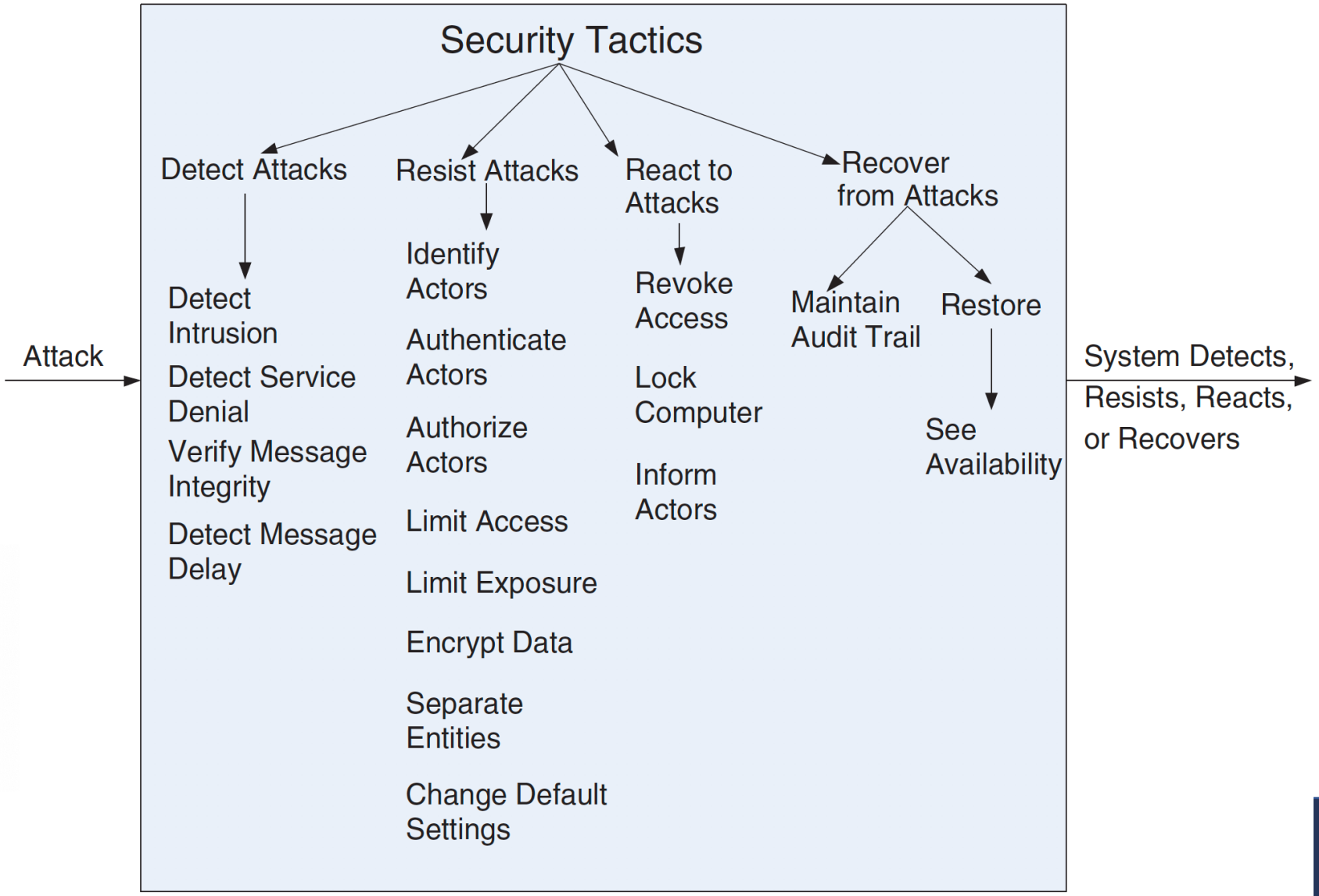
- about time and the software system's ability to meet timing requirements
- Event arrival patterns: Periodic, Stochastic, Sporadic
- Measurements:
  - Latency
  - Deadlines in processing
  - Throughput
  - jitter of the respsonse
  - number of events not processed

Performance

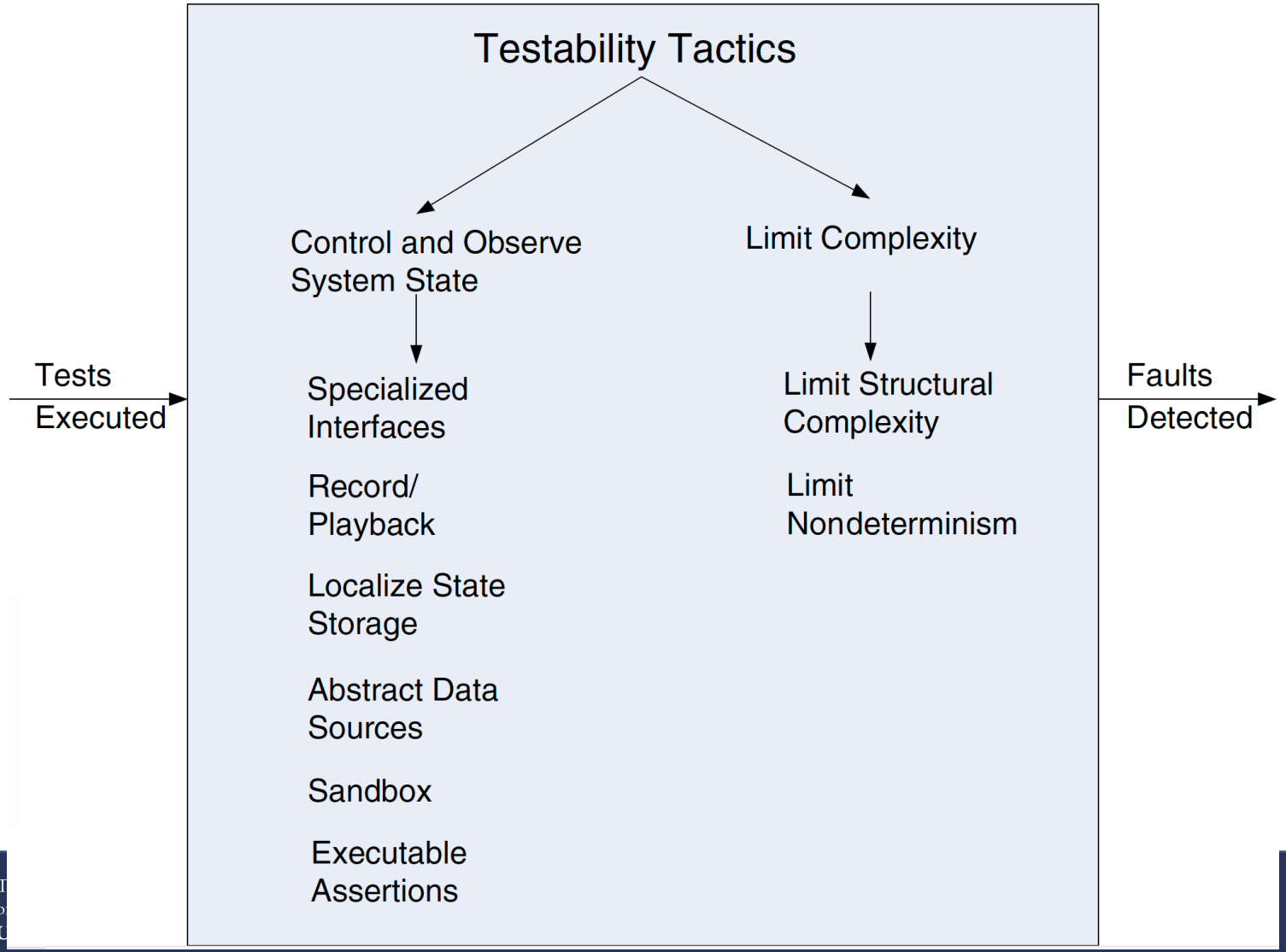
response time = processing time + blocked time



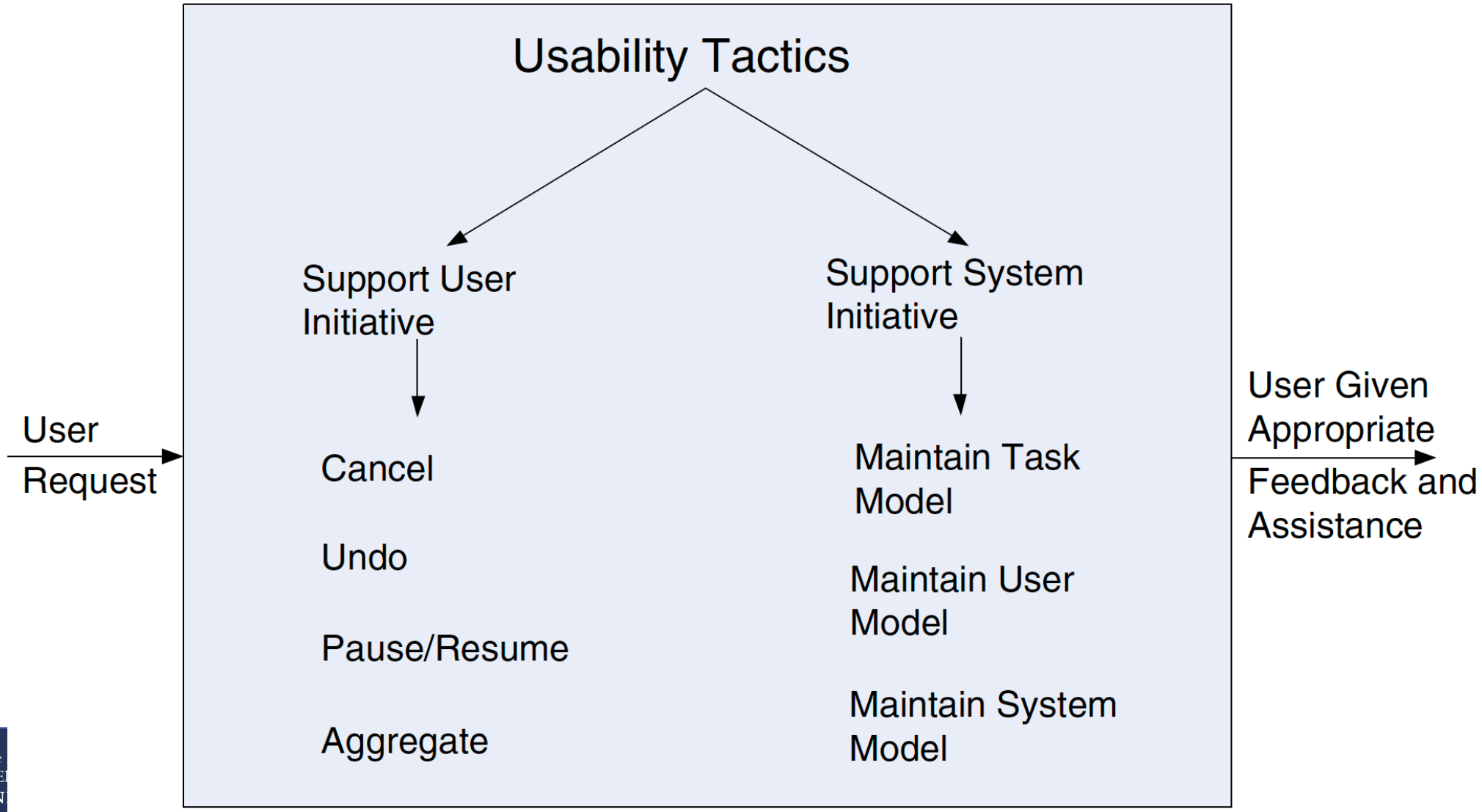
Security


Security Tactics
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
Attack
System Detects, Resists, Reacts, or Recovers

Testability

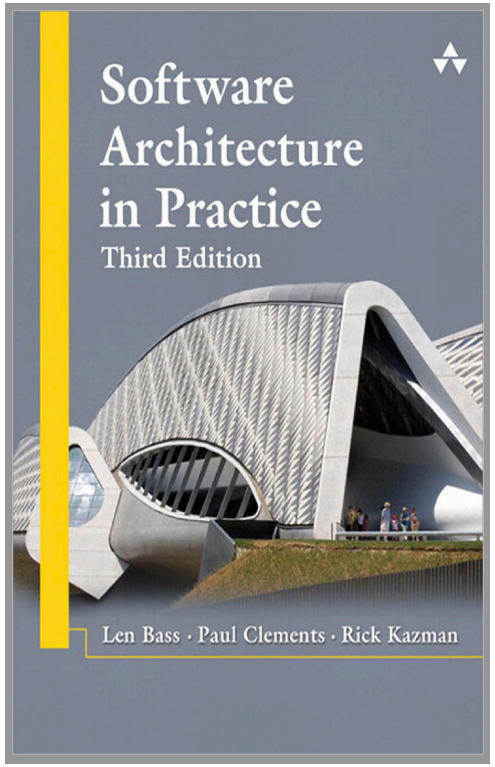


Usability


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

Summary of Tactics and Patterns

Tactics are the “building blocks” of design, from which architectural patterns are created. Tactics are atoms and patterns are molecules. Most patterns consist of several different tactics.



Many tactics described in Chapter 4-10

- Brief high-level descriptions (about 1 paragraph per tactic)
- Checklist available

Summary of Architecture

Architecture as
structures and relations

- Patterns
- Tactics

Architecture as
documentation

- Views
- Rationale

Architecture as process

- Decisions
- Evaluation
- Reconstruction
- Agile

What they don’t tell you

- Good architecture requires experience
- There is more to being an architect than picking the architecture
  - “chief builder”
  - create conceptual integrity

Future Readings

- Bass, Clements, and Kazman. Software Architecture in Practice. Addison-Wesley, 2013.
- Boehm and Turner. Balancing Agility and Discipline: A Guide for the Perplexed, 2003.
- Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford. Documenting Software Architectures: Views and Beyond, 2010.
- Fairbanks. Just Enough Software Architecture. Marshall & Brainerd, 2010.
- Jansen and Bosch. Software Architecture as a Set of Architectural Design Decisions, WICSA 2005.
- Lattanze. Architecting Software Intensive Systems: a Practitioner's Guide, 2009.
- Sommerville. Software Engineering. Edition 7/8, Chapters 11-13
- Taylor, Medvidovic, and Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley, 2009.