Software Architecture

15-313:
Foundations of Software Engineering

Jonathan Aldrich

Sources

[BCK03] Bass, Clements, and Kazman. *Software Architecture in Practice.* Addison-Wesley, 2003.

[TMD08] Taylor, Medvidovic, and Dashofy. *Software Architecture: Foundations, Theory, and Practice.* Wiley, forthcoming.

Software Architecture

- A software system's architecture is the set of **principal design decisions** made about the system [TMD08]
- Architecture typically focuses on views of a system's structure: how it decomposes into elements, the externally visible properties of those elements, and the relationships among them
  – adapted from [BCK03]
  - BUT architecture includes all principal design decisions, not just structure
- Architectural questions
  - What are the entities (modules, servers, threads, objects) in a system and connections between them?
    - Does this change over time? If so, how?
  - How are entity interactions constrained?
    - Communication: types, protocols, synchronization
  - System composition and evolution rules
    - Extension rules (e.g. for framework plug-ins)
    - Architectural patterns/styles (e.g. pipe-and-filter, repository, 3-tier)

Design vs. Architecture

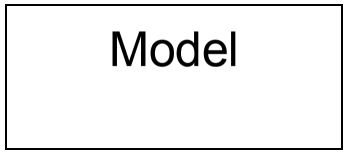
Design Questions

- How to I add a menu item in Eclipse?
- What lock protects this data?
- How does Google rank pages?
- What encoder should I use for secure communication?
- What is the interface between objects?

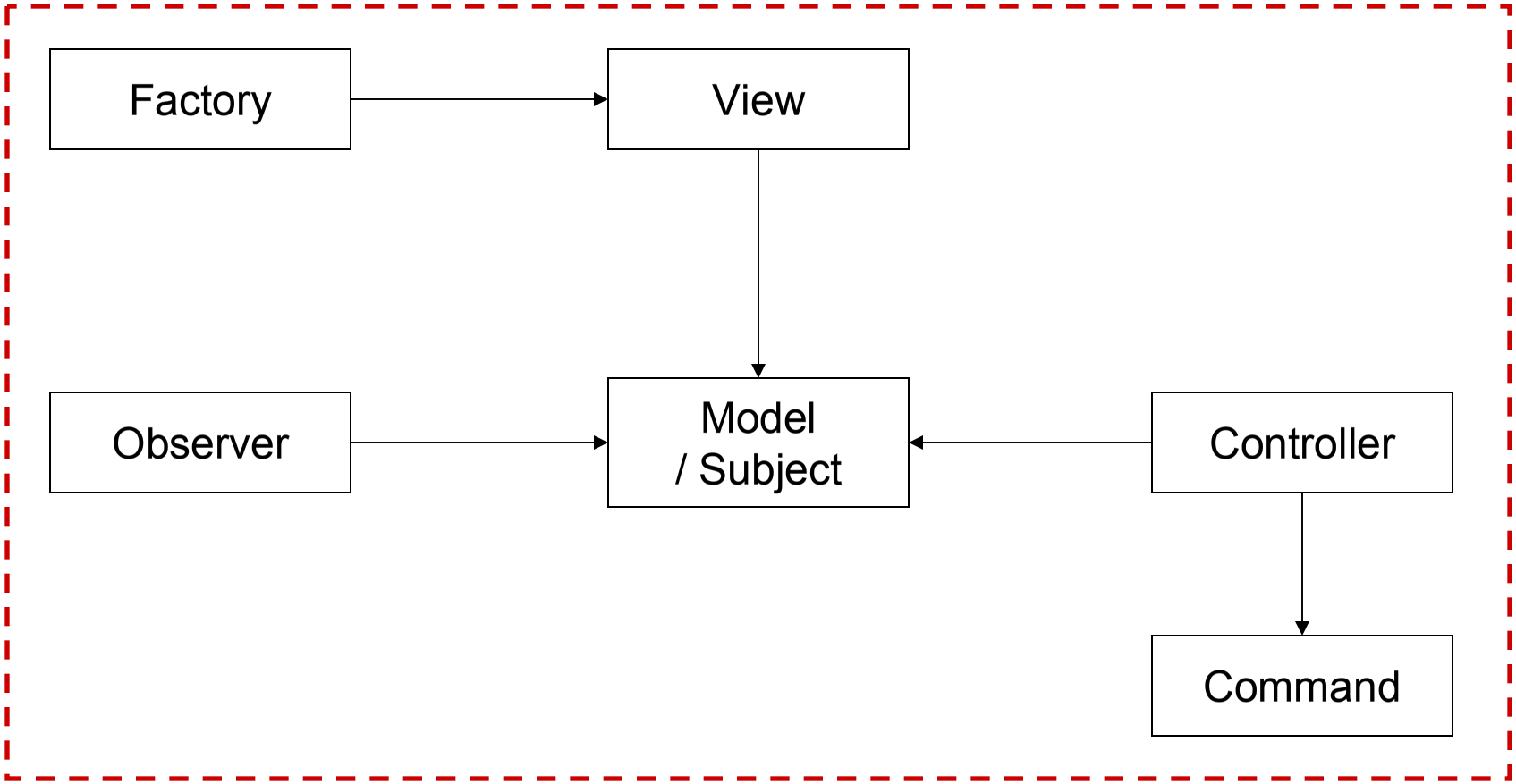
Architectural Questions

- How do I extend Eclipse with a plugin?
- What threads exist and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

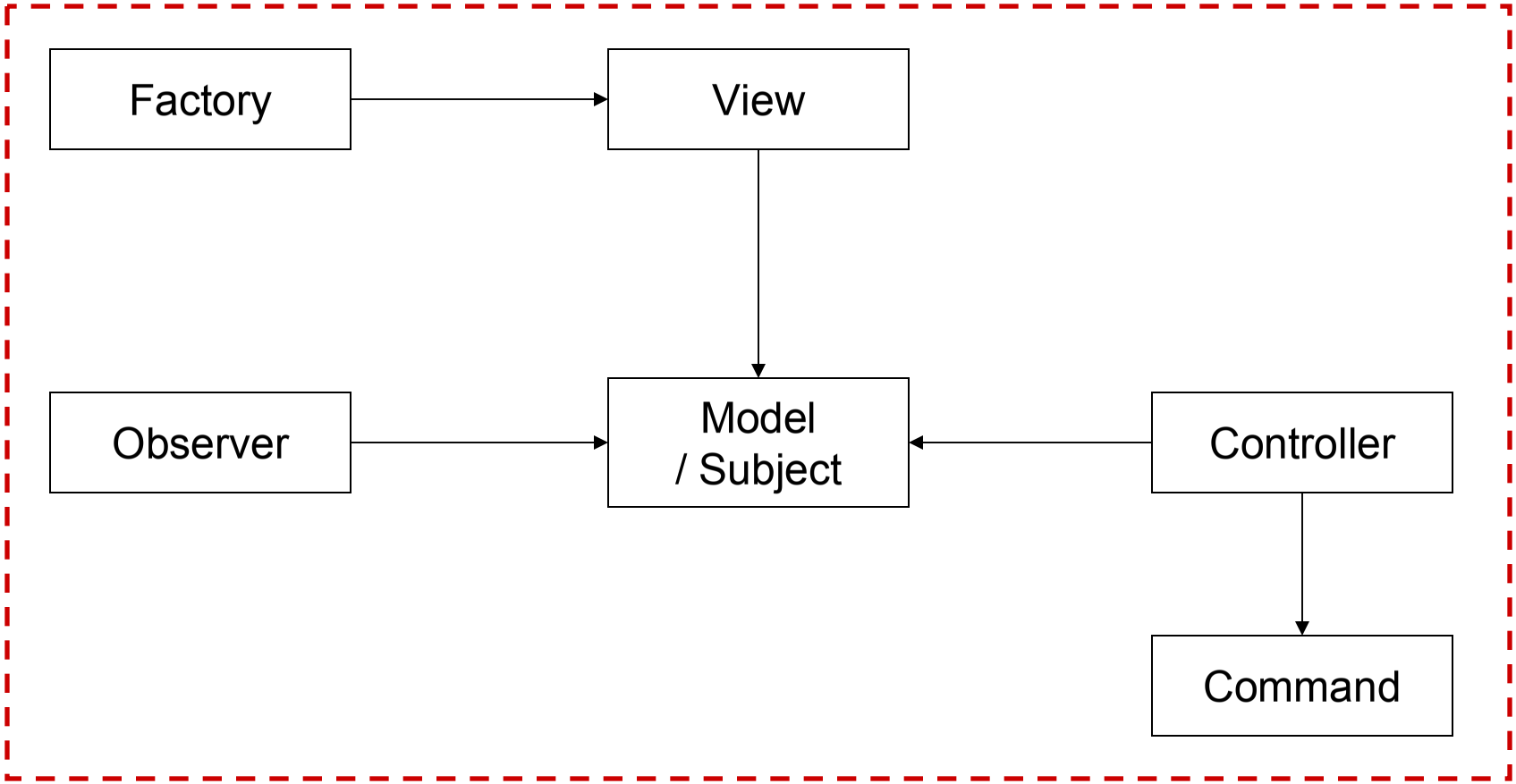
Objects [Dahl and Nygaard '67]



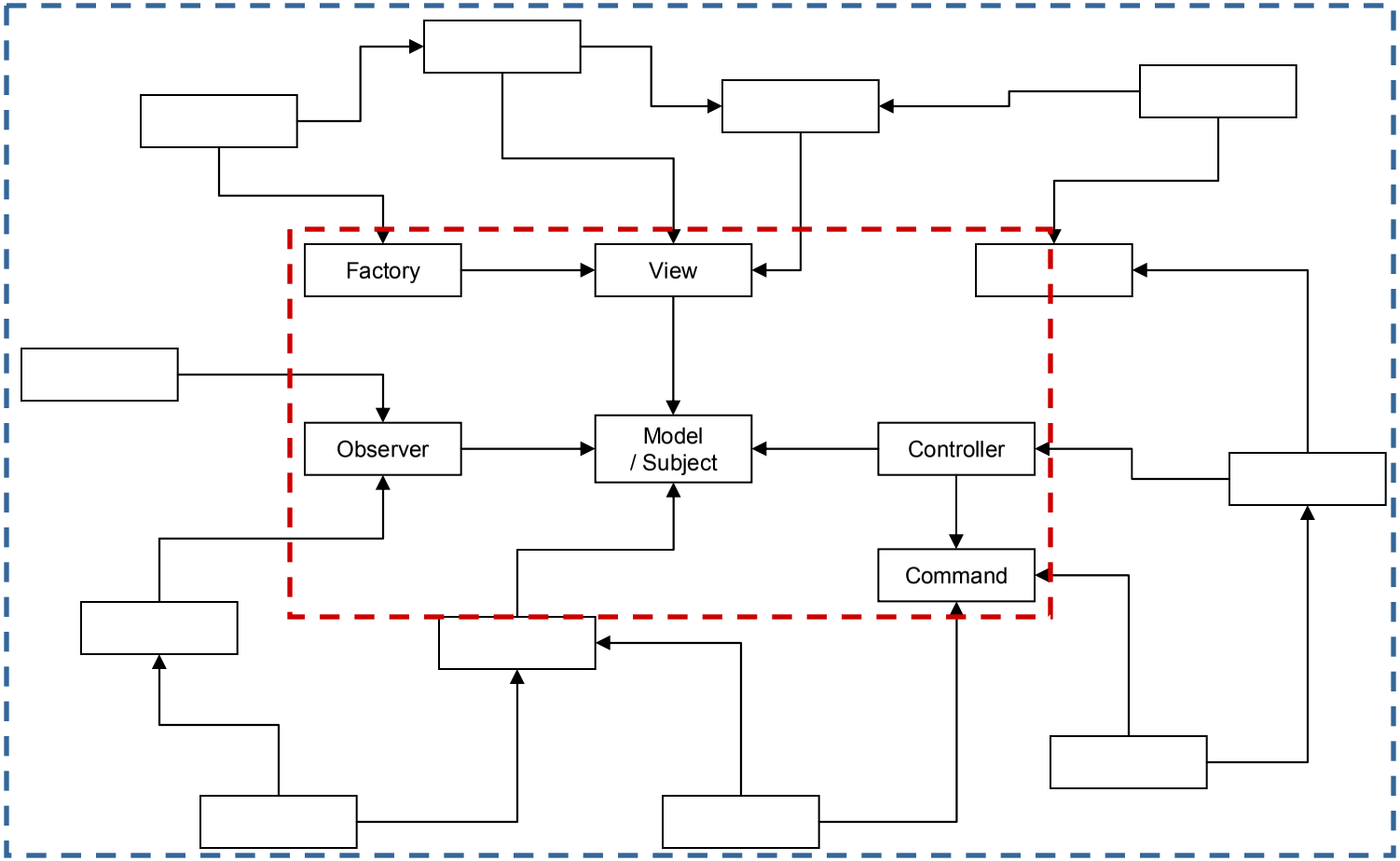
Patterns [Gamma, Helm, Johnson, and Vlissides '95]



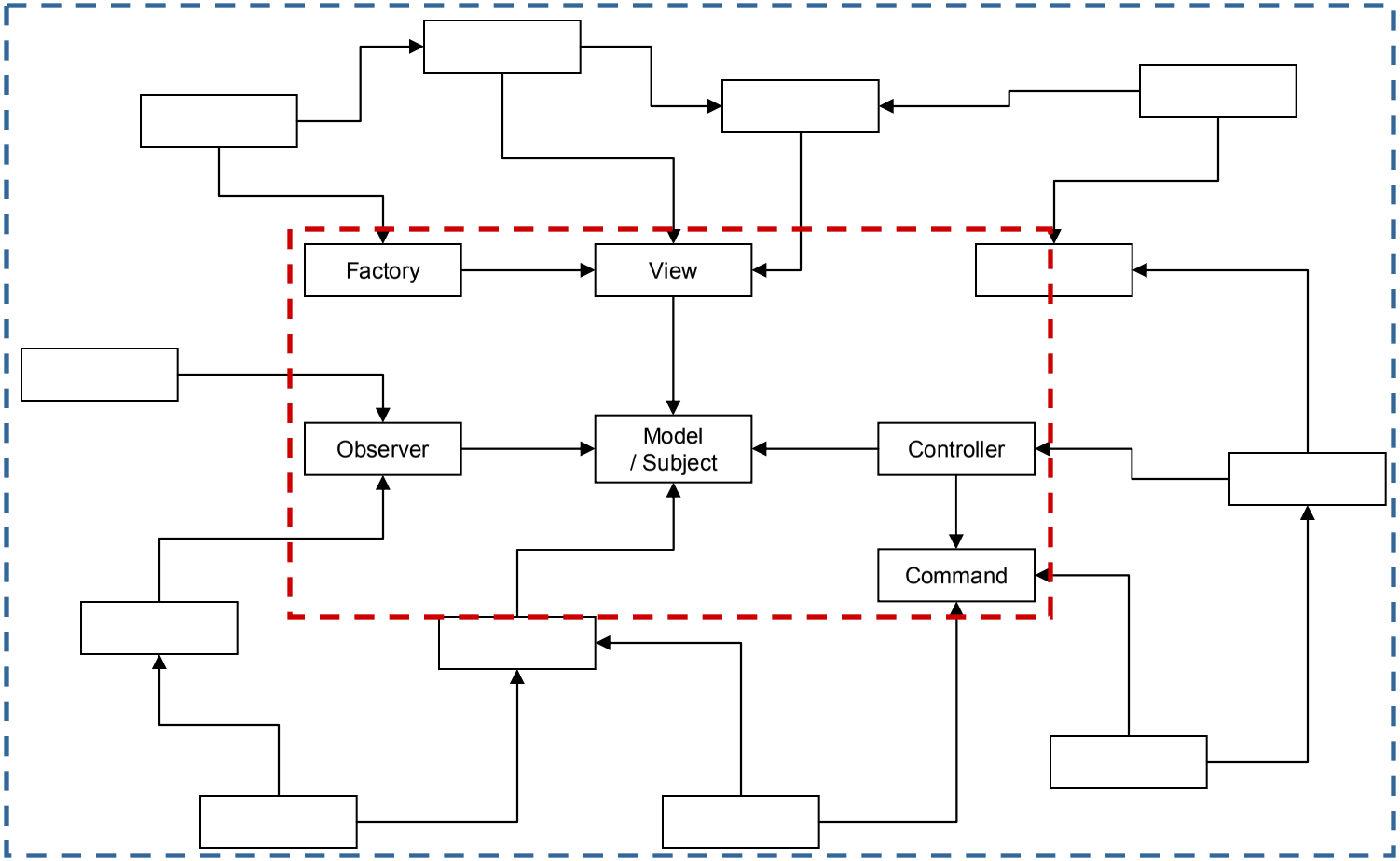
Patterns [Gamma, Helm, Johnson, and Vlissides '95]



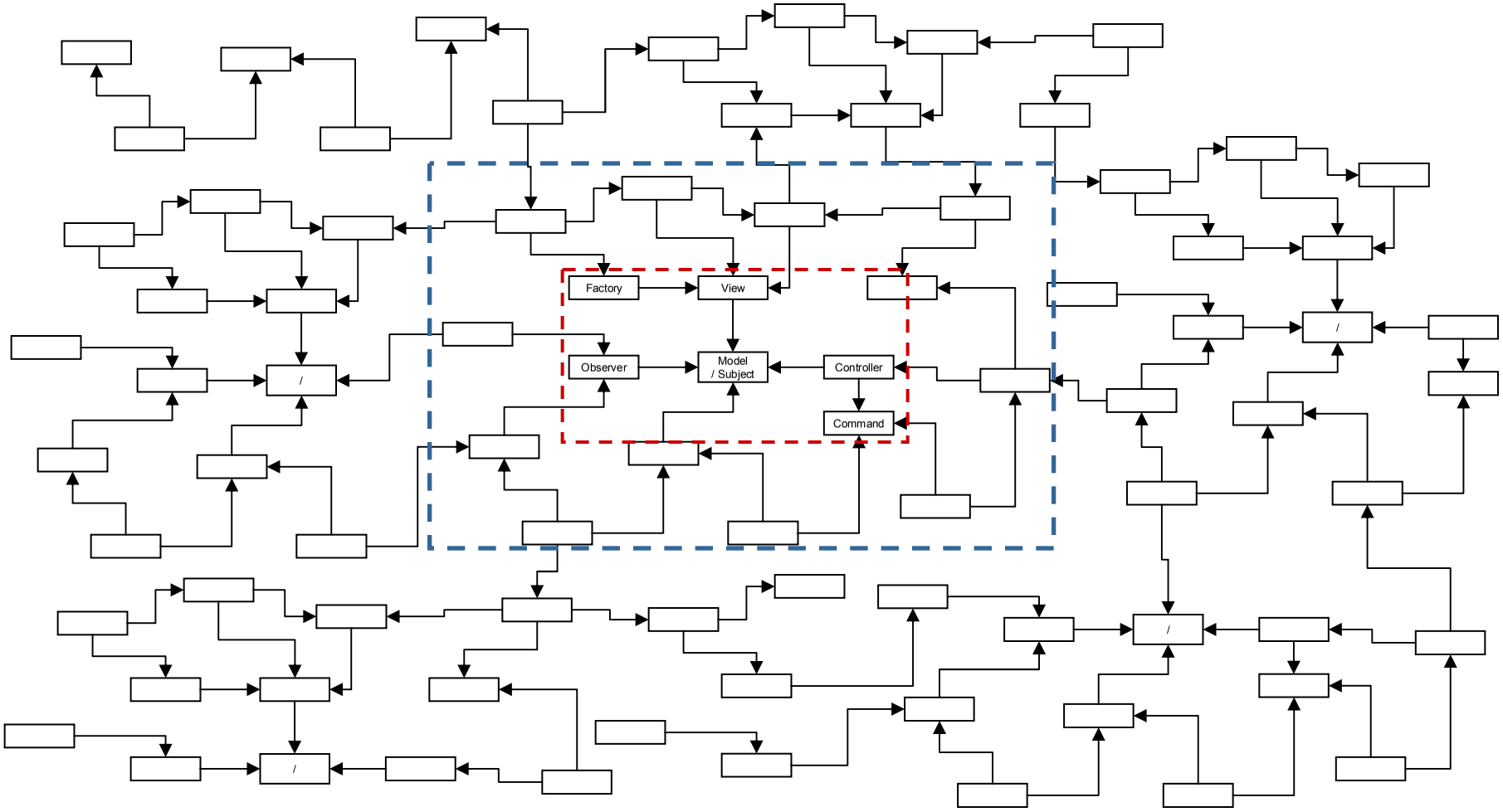
Patterns [Gamma, Helm, Johnson, and Vlissides '95]



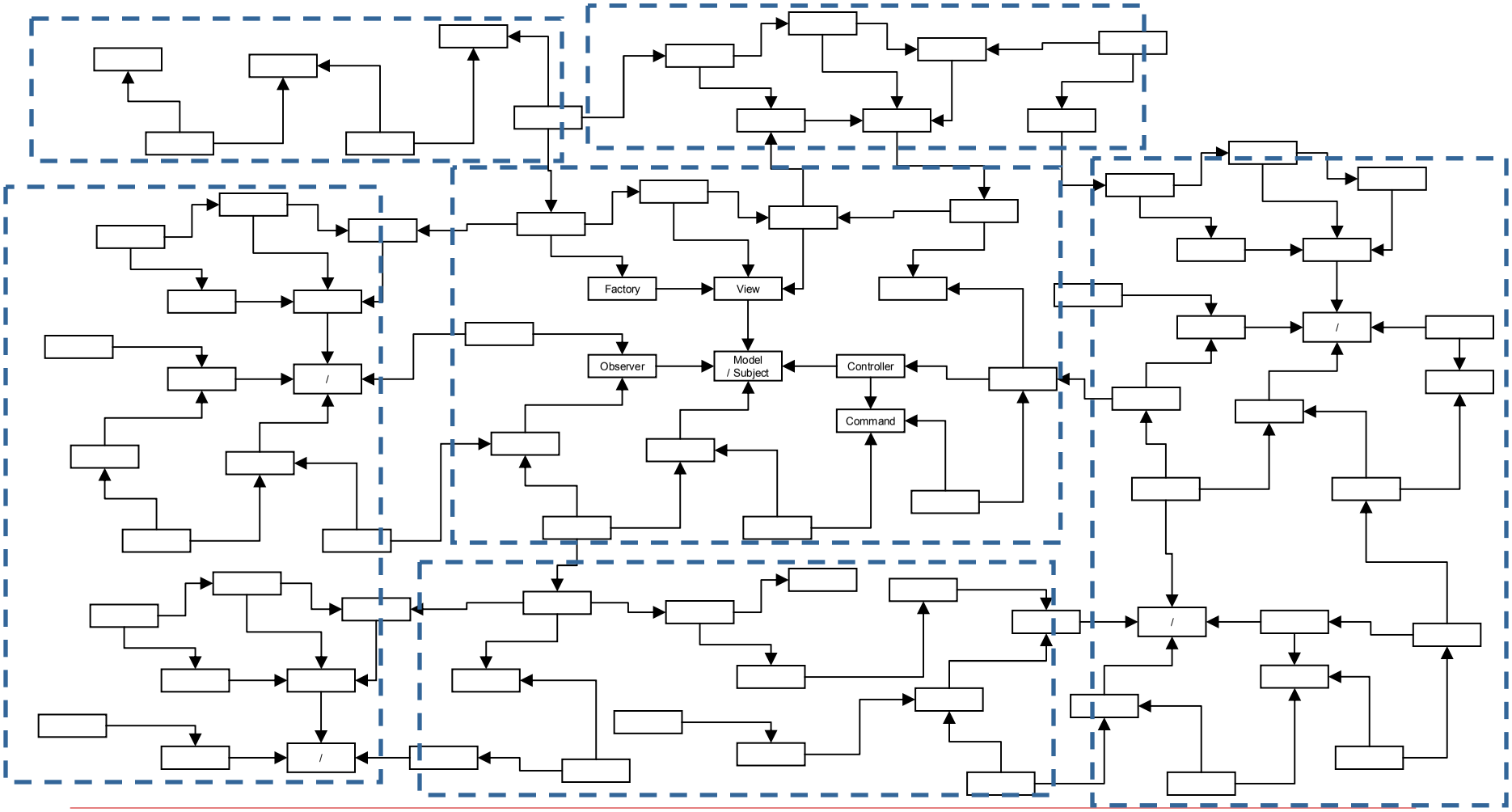
Patterns [Gamma, Helm, Johnson, and Vlissides '95]



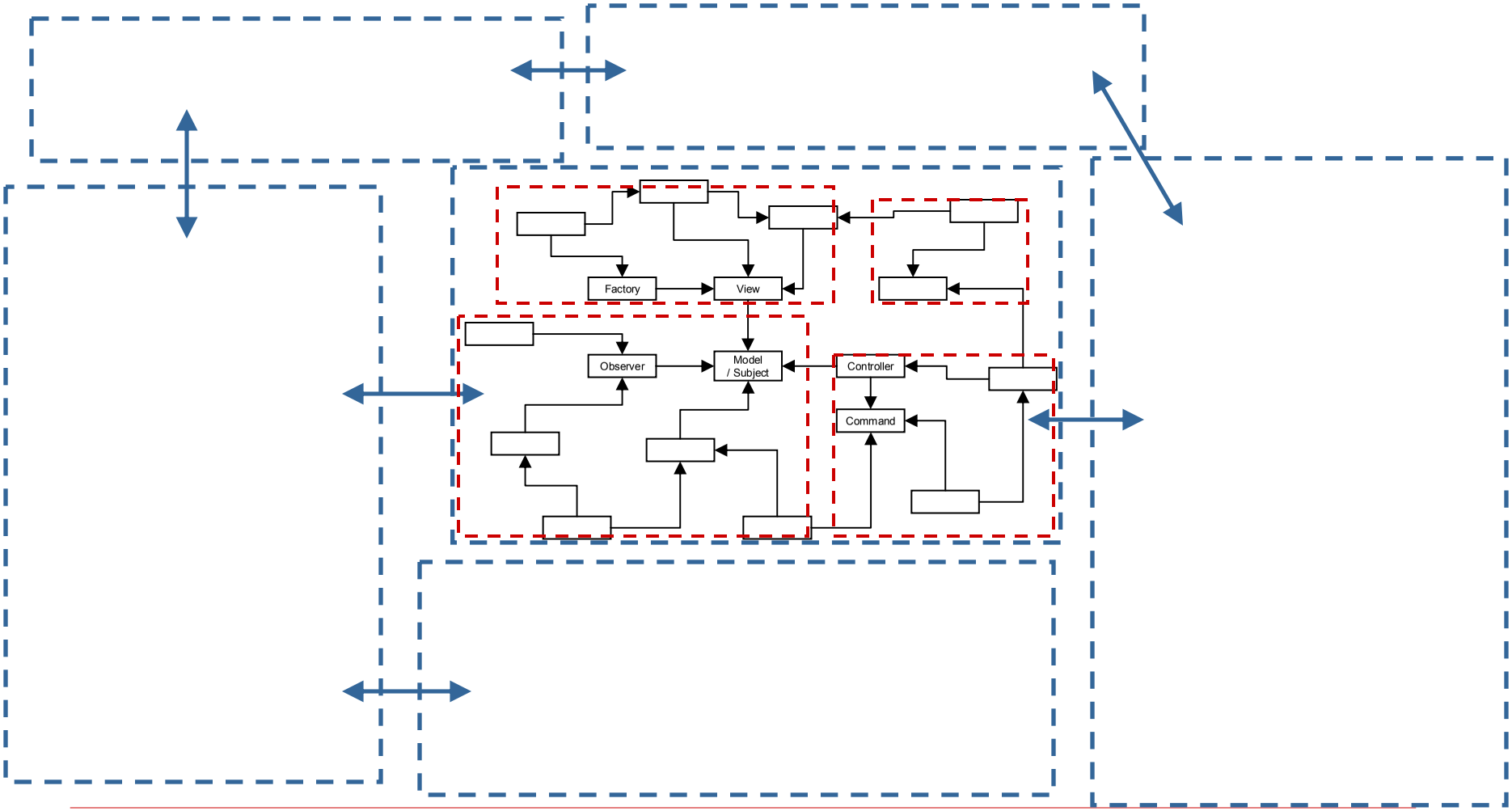
Patterns [Gamma, Helm, Johnson, and Vlissides '95]



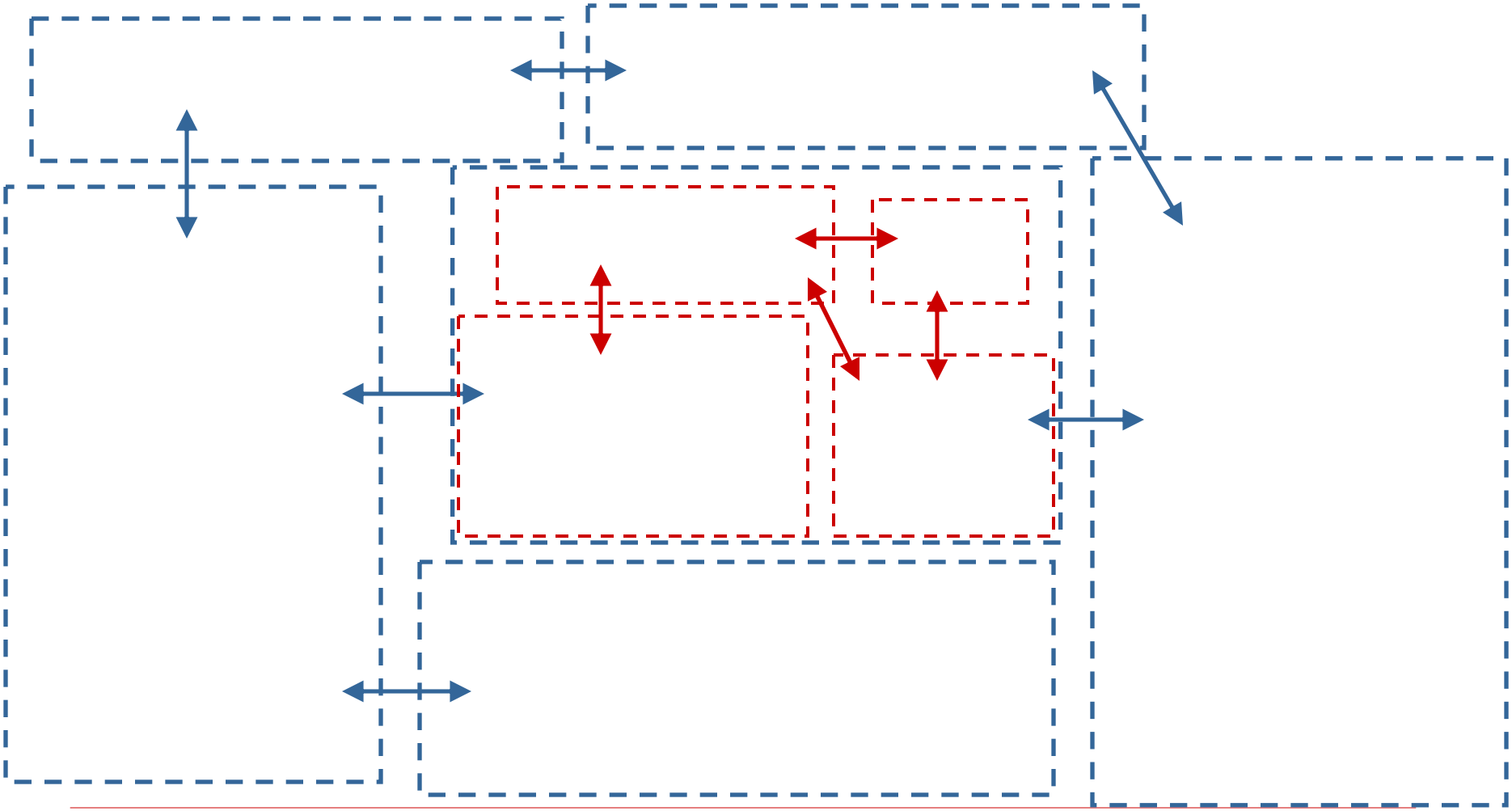
Architecture [Perry and Wolf 1992] [Garlan and Shaw 1993]



Architecture [Perry and Wolf 1992] [Garlan and Shaw 1993]



Architecture [Perry and Wolf 1992] [Garlan and Shaw 1993]



Rationale for Architecture

- Need for an additional level of abstraction
  - System-wide decomposition
    - Allow teams to work independently on subsystems
    - Specify interfaces before full detailed design is known
  - System-wide design constraints
    - Avoid different conventions for each subsystem
      - and therefore avoid interface mismatch
  - Promote system-wide quality attributes
    - Design pattern are often at too low a level of abstraction to affect system-level quality attributes
    - e.g. reliability requires end-to-end reasoning

Architecture is an Abstraction

- Focus on **principal** design decisions
  - **Structure** – components and connections
  - **Behavior** – responsibilities of each component, high level algorithms
  - **Interaction** – rules governing how components communicate
  - **Quality attributes** – strategy for achieving
  - **Implementation** – language, platform, libraries, etc.
  - ***Any decision that impacts key stakeholder concerns or has global impact on the program***
- Elide unimportant details
  - Decisions that are **internal to a component**
    - i.e. which other components cannot depend on
    - e.g. internal algorithms, data structures, local design patterns
  - AND do not impact **key stakeholder concerns**

Architecture is design, but not all design is architectural

Levels of Abstraction

- Requirements
  - high-level “what”
- Architecture
  - high-level “how”, mid-level “what”
- Detailed design (e.g. design patterns)
  - mid-level “how”, low-level “what”
- Code
  - low-level “how”

Architecture as Communication

- What are major system goals, and how are they achieved?
  - Stakeholders ↔ Developers
  - Case study:
    After two days of requirements presentation, a client sees an architecture diagram, realizes he doesn't understand something, and begins the first productive conversation about what the system is supposed to do

    – Software Architecture in Practice, pp 27-28

- What guidelines must be followed for independently developed components to work together?
  - Developers ↔ Developers

Benefits of Architecture [BCK03]

- Aids in communication with stakeholders
  - Shows them "how" at a level they can understand, raising questions about whether it meets their needs
- Defines constraints on implementation
  - Design decisions form "load-bearing walls" of application
- Dictates organizational structure
  - Teams work on different components
- Inhibits or enables quality attributes
  - Similar to design patterns
- Supports predicting cost, quality, and schedule
  - Typically by predicting information for each component
- Aids in software evolution
  - Reason about cost, design, and effect of changes
- Aids in prototyping
  - Can implement architectural skeleton early

Business Case: Cell Phones [M. Bass]

- Market is driven by killer products
  - e.g. Razr, iPhone
- Most profit is made at initial release
  - Premium charged on initial sales
  - Drops rapidly when copycats arrive
- Business model
  - Be first to market with new features
- Software quality attributes
  - Ability to change rapidly and at low cost
- True story: effect of architecture
  - Leading cell phone manufacturer
    - not enough new products
    - starts to lose market share, decides to release faster
    - leads to trouble: e.g. tone so loud it damages hearing ➔ recalls
  - Analysis
    - software structure did not enable rapid change
    - too costly to rewrite software from scratch
    - eventually left cell phone business entirely

More is Not Necessarily Better [M. Bass]

- Domain: mobile infotainment
  - Key quality attribute was Modifiability
- Architecture
  - Chose very general, heavyweight framework that promoted modifiability
- Results
  - Too much resource use
  - Unstable, overly complex system
  - Full generality of architecture was not needed
- Lessons
  - Every architectural decision has a cost
  - Ensure decisions are traceable to a business goal
    - Make quality attributes concrete with scenarios
    - Prioritize quality attribute scenarios

Architectural Drivers

- Functional requirements
  - What the system is supposed to do
- Quality attributes
  - How the system does what it does
- Technical constraints
  - Design decisions you have to work around
    - e.g. Google has 4 approved programming languages
    - e.g. Must use platform X to interoperate with legacy code
- Business constraints
  - e.g. must deliver the product in time for Christmas

- Quality attributes typically drive the architecture
  - But allocation of functionality, technical and business constraints cannot be ignored

Architectural Drivers: Amazon.com

- Functional requirements?
- Technical constraints?
- Quality attributes?
- Business constraints?

Amazon.com

- Which problem is most likely to drive you away?
  - The site was unavailable
  - It was unable to give you recommendations
  - Prices were 10% higher than at the competition
  - Your stored credit card information was stolen
  - You had to wait for 30 seconds after each click

Amazon.com

- Which problem is most likely to drive you away?
  - The site was unavailable
    - Reliability / Scalability
  - It was unable to give you recommendations
    - Functionality
  - Prices were 10% higher than at the competition
    - Maintainability
  - Your stored credit card information was stolen
    - Security
  - You had to wait for 30 seconds after each click
    - Performance / Scalability

- What's more important: functionality or quality attributes?

Architectural Drivers: Functional Requirements

- Typically specified as use cases

- Architectural role
  - Assign functionality to components
  - Ensure no functionality is forgotten
  - Ensure adequate connections
    - Communication necessary to perform tasks

Architectural Drivers: Business Constraints

- Imposed by the organization, the marketplace, or business issues

- Examples
  - Time to market
  - Cost
  - Available personnel and expertise
  - Required standards & certifications

Architectural Drivers: Technical Constraints

- Externally imposed design decisions

- Examples
  - Language
  - Platform (EJB, ASP.NET)
  - Standards
  - Interoperability with legacy systems

Documenting Constraints [Lattanze/M. Bass]

- Record
  - Rationale for constraint
  - Stakeholder originating constraint
  - Flexibility of constraint
  - Alternatives considered

- Typically no notion of priority, as there is no choice in the matter!

Quality Attributes

- The system shall be…
  - Modifiable — For what changes? At what cost?
  - Efficient — How much performance is enough?
  - Reliable — What percentage? In what scenarios?
  - Scalable — To what level? At what cost?
  - Usable — To whom? For what? Is this architectural?
  - Maintainable — In what scenarios? At what cost?

    And which of these are most important? e.g. Reliability probably imposes efficiency costs.
- Great, now let's design!
  - Or are we really ready for that?