



# Architectural Decomposition & Representations

Reid Holmes



# What is SW architecture?

- ▶ Definition:

“The set of principal design decisions about the system”

- ▶ Blueprint for construction and evolution.
- ▶ Encompasses:
  - ▶ Structure
  - ▶ Behaviour
  - ▶ Non-functional properties

# Components

- ▶ Elements that encapsulate processing and data at an architectural level.
- ▶ Definition:
  - ▶ Architectural entity that:
    - ▶ encapsulates a subset of functionality.
    - ▶ restricts access via explicit interface.
    - ▶ has explicit environmental dependencies.

# Connectors

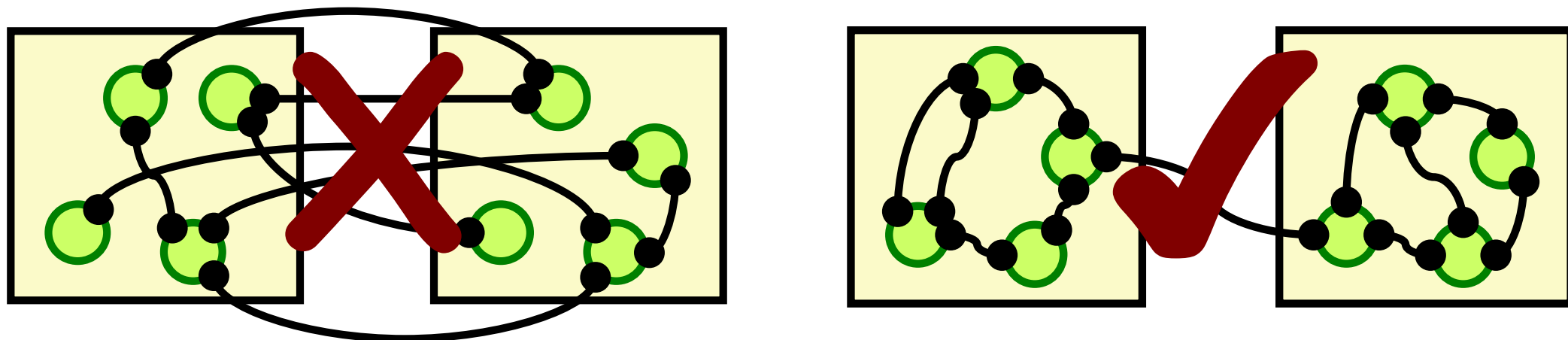
- ▶ Definition:
  - ▶ An architectural entity tasked with effecting and regulating interactions between components.
- ▶ Connectors are often more challenging than components in large heterogenous systems.
- ▶ Often consists of method calls, but be much more.
- ▶ Frequently provide application-independent interaction mechanisms.

# Configurations

- ▶ Bind components and connectors together in a specific way.
- ▶ Definition:
  - ▶ An architectural configuration, or topology, is a set of specific associations between the components and the connectors of the system's architecture.
- ▶ Differentiates a bag of components and connectors from an implementable system.

# Topological Goals

- ▶ Minimize **coupling** between components
  - ▶ The less components know about each other, the better (also known as information hiding).
- ▶ Maximize **cohesion** within each component
  - ▶ Components should be responsible for a logical service; extraneous functionality should not be present.



[Steve Easterbrook: <http://www.cs.toronto.edu/~sme/CSC302/notes/04-package-diagrams.pdf>]

# Coupling

**WORSE**

**BETTER**



Content

No  
interaction

Global

Message

Control

Data (params)

Useful definitions:

- [https://en.wikipedia.org/wiki/Coupling\\_\(computer\\_programming\)](https://en.wikipedia.org/wiki/Coupling_(computer_programming))

# Cohesion

**WORSE**

**BETTER**



Coincidental

Functional

Logical

Sequential

Temporal

Communication

Useful definitions:

- [http://en.wikipedia.org/wiki/Cohesion\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Cohesion_(computer_science))



# Abstraction

- ▶ Complex problems can be approached by abstracting away unnecessary detail
- ▶ Focus on the key issues while eliding extraneous detail (some of these details will be pertinent during more detailed design activities)
- ▶ In software two classes of abstraction dominate:
  - ▶ Control abstraction
    - ▶ (e.g., structured programming)
  - ▶ Data abstraction
    - ▶ (e.g., abstract data types)

# Decomposition

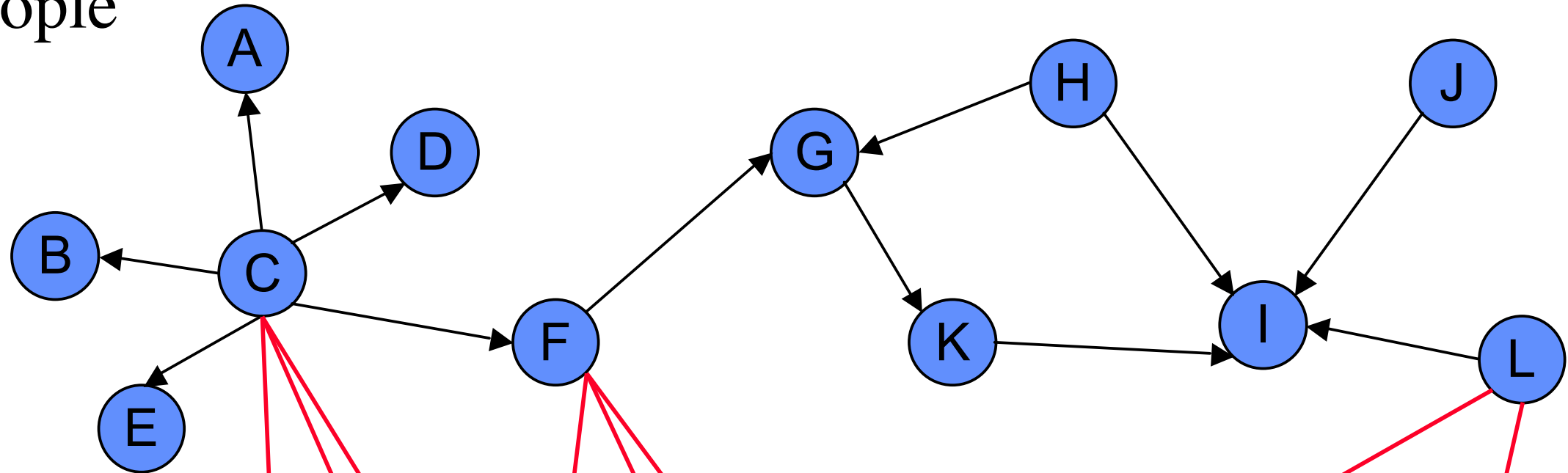
- ▶ Top-down abstraction is also called decomposition
  - ▶ Break problem into independent components
  - ▶ Describe each component
- ▶ Criteria for decomposition can include:
  - ▶ Implementing teams
  - ▶ Application domains (aka obvious partitions)
  - ▶ Parallelization
- ▶ Make typical cases simple, and exceptional cases possible

# Conway's Law

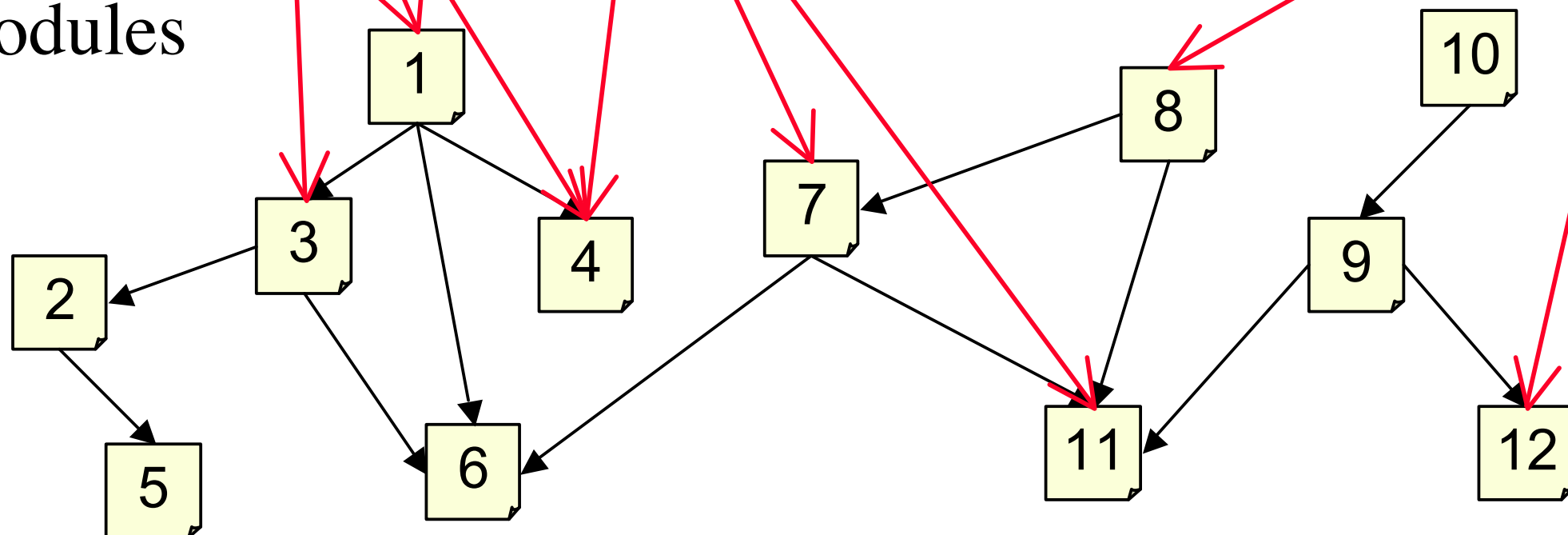
“The structure of a software system reflects the structure of the organization that built it”

# Conway's Law

People



Modules



[Steve Easterbrook: <http://www.cs.toronto.edu/~sme/CSC302/notes/04-package-diagrams.pdf>]



# Architectural representations

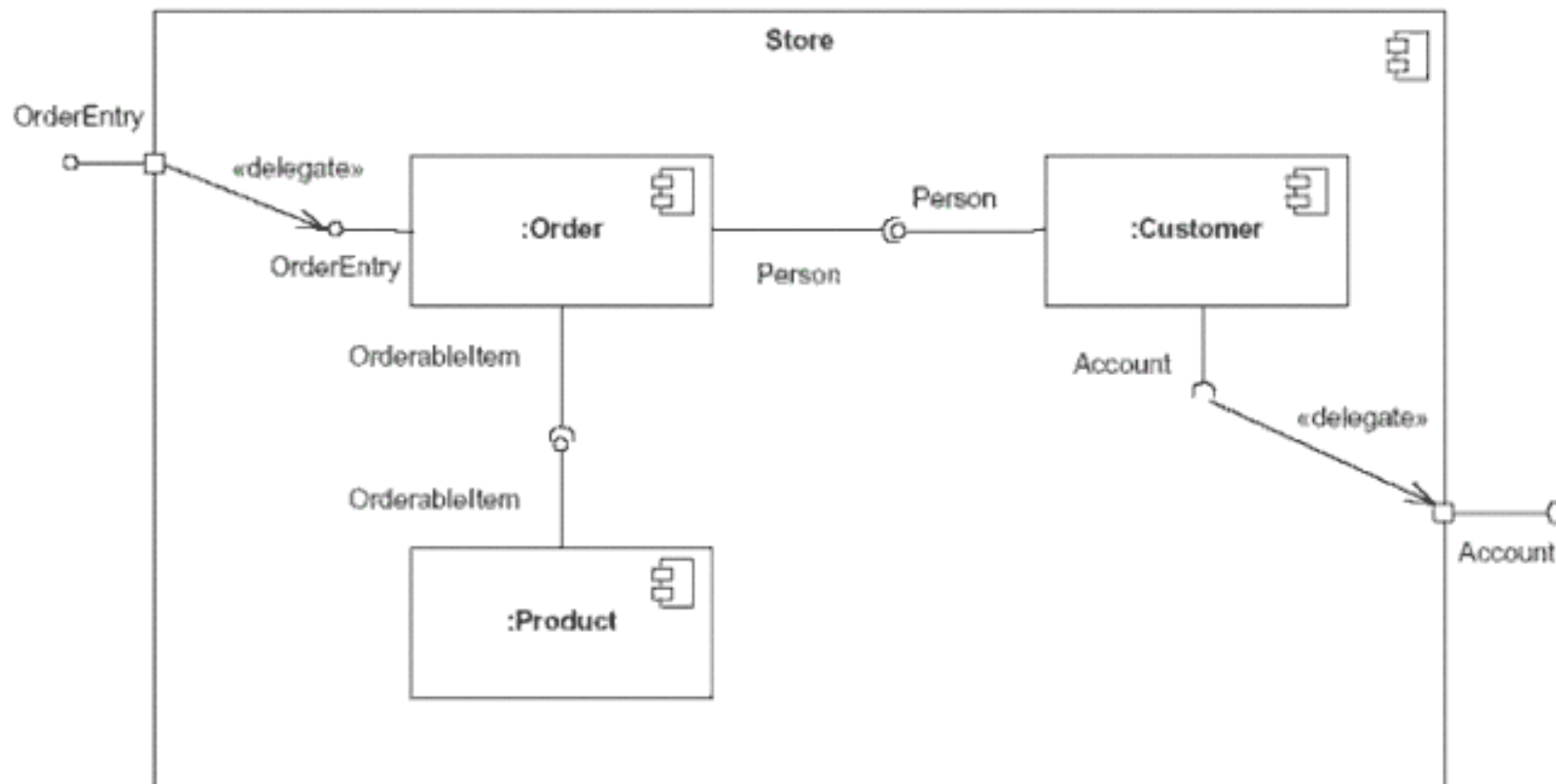
- ▶ Software architecture is fundamentally about facilitating technical communication between project stakeholders
- ▶ An opaque architecture has no value as it will not be adequately understood
- ▶ Properties of representations:
  - ▶ Ambiguity: Open to more than one interpretation?
  - ▶ Accuracy: Correct within tolerances
  - ▶ Precision: Consistent but not necessarily correct

# Architectural views

- ▶ Architectural models can be overwhelming
  - ▶ Different views focus on specific subsets of elements or subsets of relationships
  - ▶ Views often focus on specific concerns or scenarios within a system
- ▶ Views overlap; maintaining consistency between views is challenging

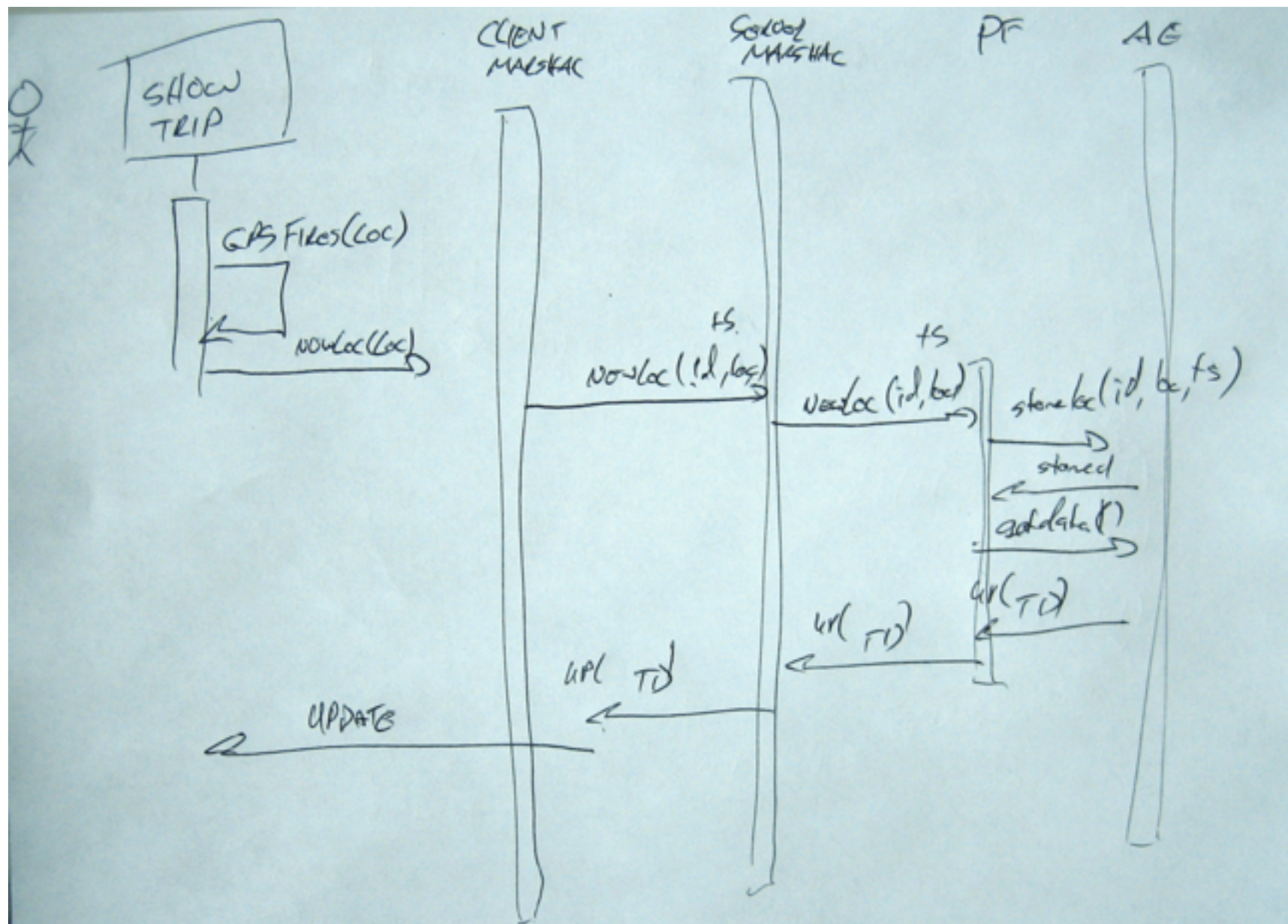
# Component diagram

- ▶ Captures components and relationships.
- ▶ Required and provided APIs explicitly recorded.



# Sequence diagram

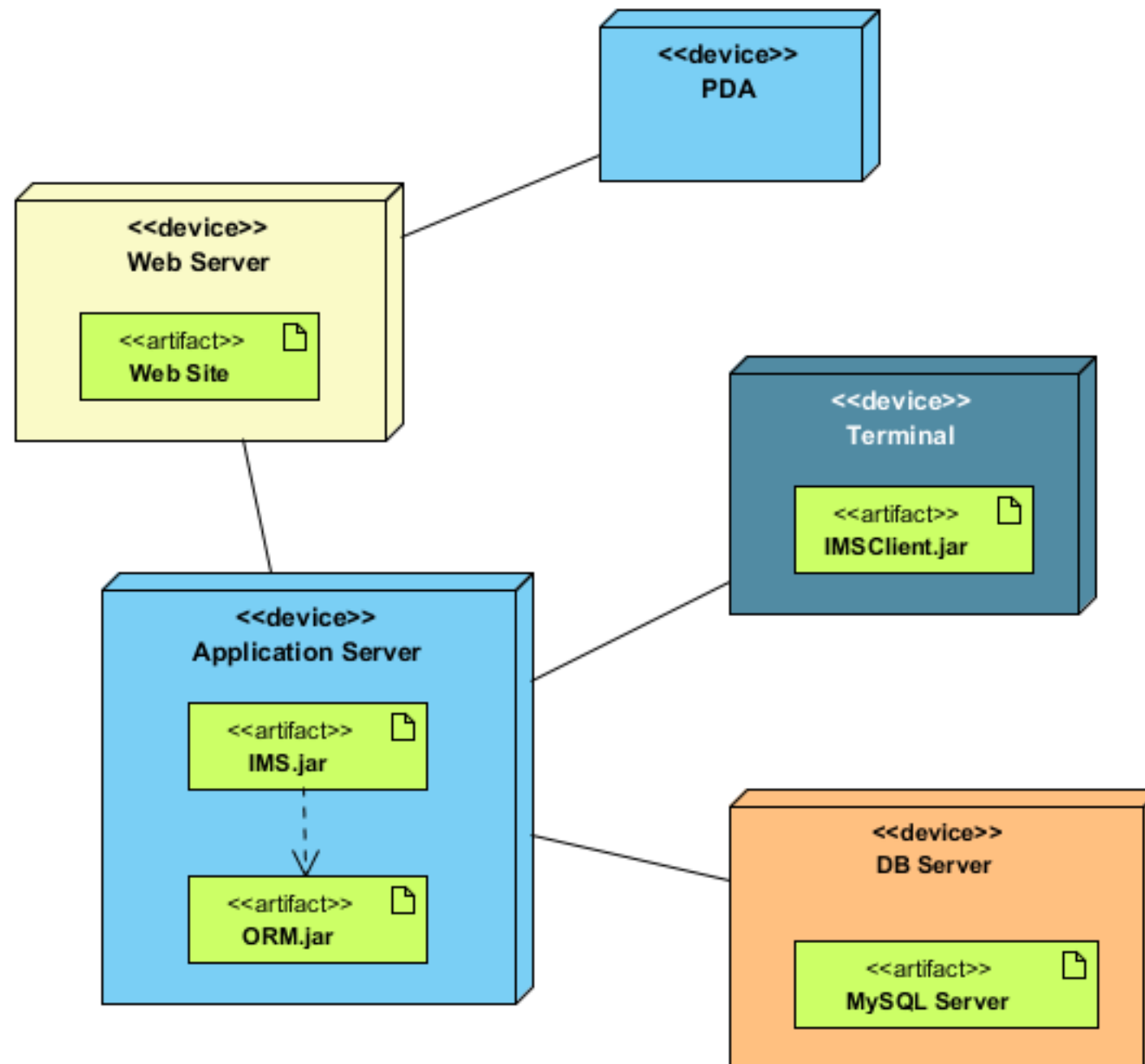
- ▶ Focus on inter-component collaboration.
- ▶ Capture behaviour for specific runtime scenarios.





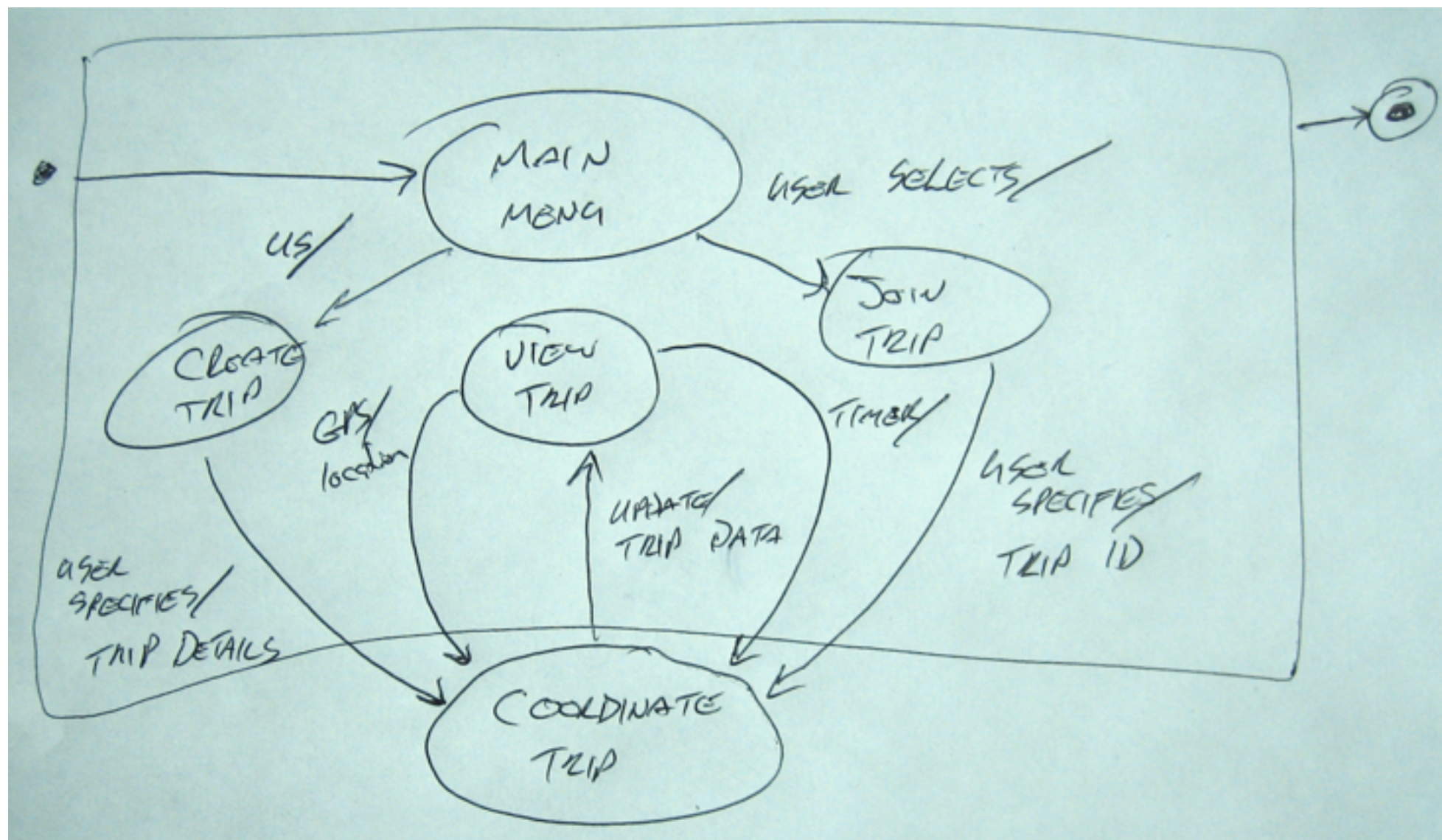
# Deployment diagram

- ▶ Provide mapping between physical devices



# Statechart diagram

- ▶ More formal description of system behaviour.
- ▶ Poor mapping between states and components.



# Prescriptive vs descriptive

- ▶ Prescriptive architecture dictates how the system will be built *a priori*.
  - ▶ (as-conceived)
- ▶ Descriptive architecture captures how the system was actually built after the fact.
  - ▶ (as-implemented)

# Architectural degradation

- ▶ Drift
  - ▶ Introduction of changes that are not captured in the current architecture but do not violate it.
- ▶ Erosion
  - ▶ Introduction of changes that violate the current architecture.



# Architectural recovery

- ▶ [ICSE 1999: Bowman, Holt, and Brewster]
- ▶ Conceptual architecture
  - ▶ How developers think about the system.
  - ▶ Focuses on meaningful relationships.
- ▶ Concrete architecture
  - ▶ How the system was actually built.
  - ▶ Necessary: the devil is in the details.