Software Architecture Fundamentals

Gruia-Catalin Roman

October 2014

Department of Computer Science
University of New Mexico

Course Overview

1. Software Architecture Fundamentals
2. Software Architecture Specification
3. Robust Design Strategies

Software Architecture Fundamentals

Chapter Overview

[1.1] The concept of architecture

[1.2] Software development lifecycle

[1.3] Architecture design phase

[1.4] Software architecture as artifact

[1.5] Technical implications

1.1 Architecture as a Concept

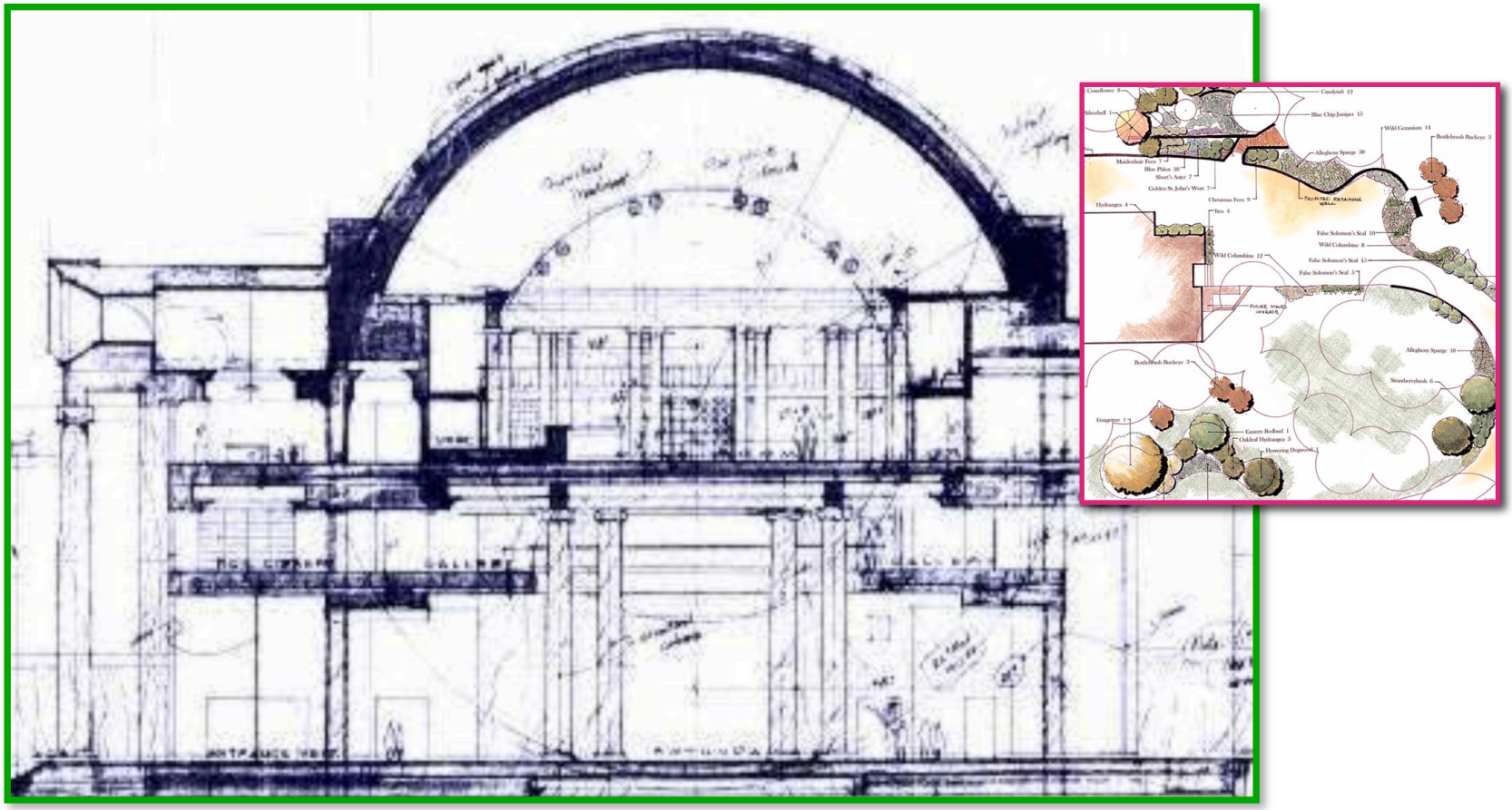
How general is the notion of architecture

- Architectural design
- Landscape architecture
- Architect of a policy of détente
- System architecture

Common Threads

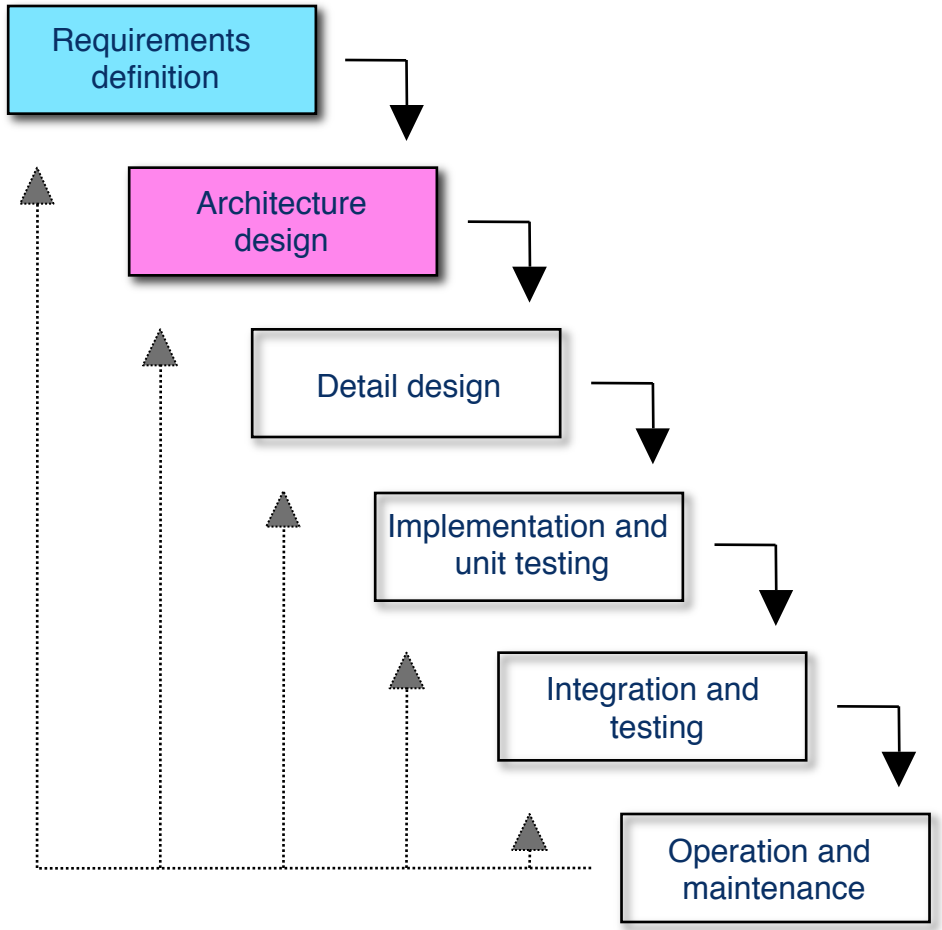
- Key organizational principle
  - integrative role
- Structure
  - explicit and visible
- Responsive to requirements
- Related to or shaping of function

Architectural Drawing



1.2 Software Development Lifecycle

- Two critical phases in software development
  - software requirements
  - software architecture

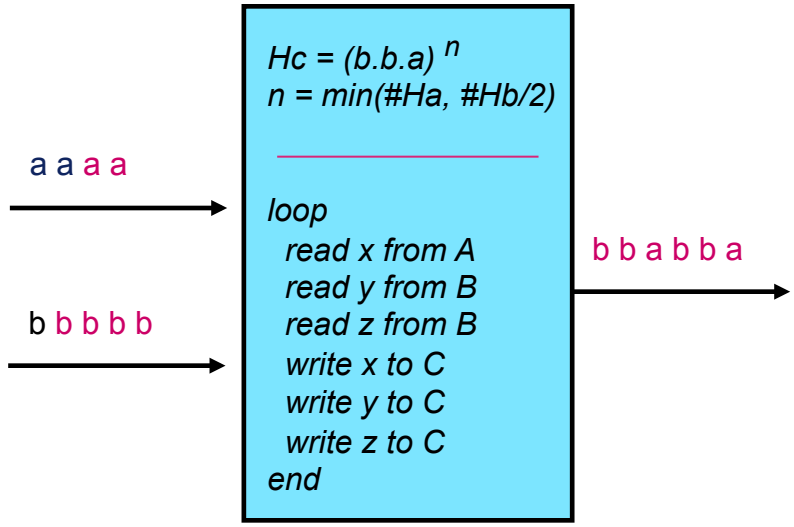


Architecture Design in Context

- Requirements definition
  - functional requirements — free of design biases
  - non-functional requirements — design constraints
  - analyzable from a functional perspective only
- Architecture design
  - overall organization of the system — technology specific
  - specifications for the individual components
    - interface, behavior, and performance parameters
  - analyzable at the functional and non-functional levels
- Detail design
  - internal mechanics of architectural components

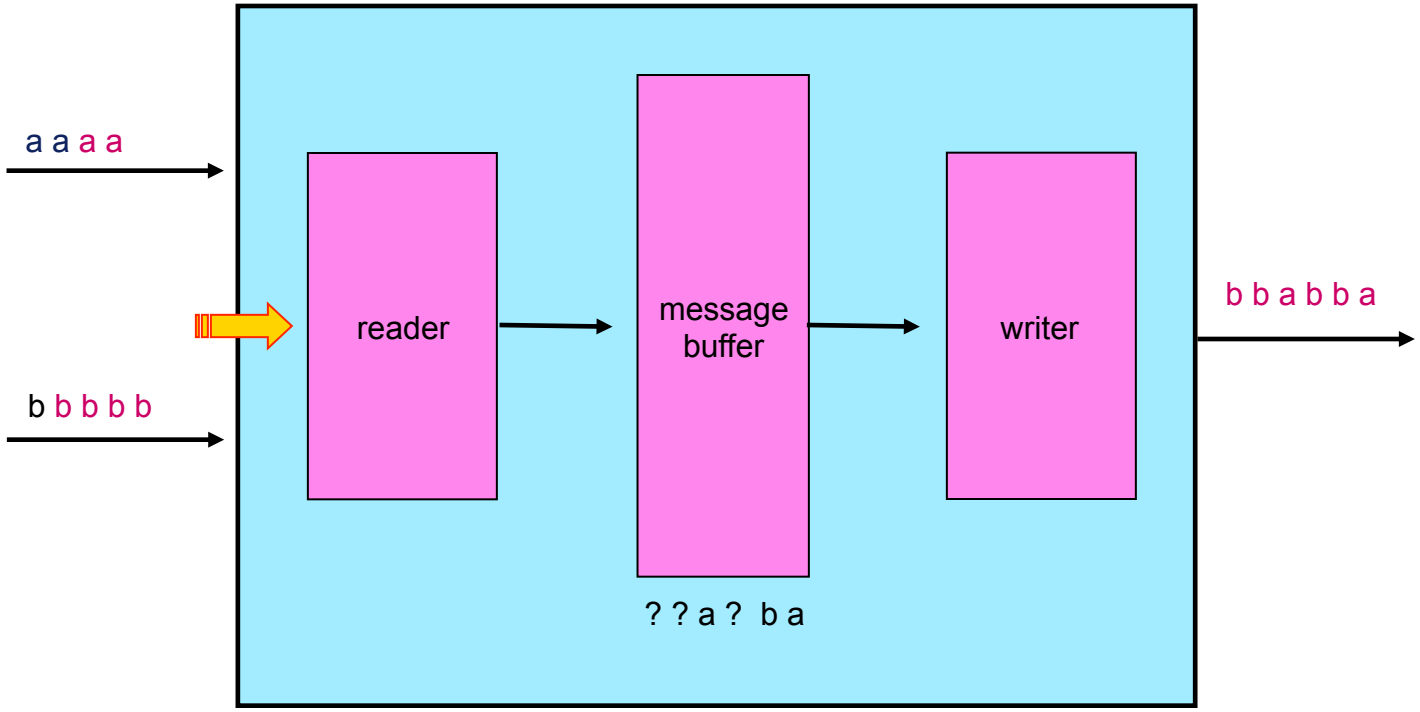
Message Merge

- Requirements definition
  - Build a message merger with inputs A and B and with output C
  - Functional requirements:
    - Release messages only in groups of three
    - A message from A must be followed by two from B
    - Preserve ordering among messages from the same input
  - Non-functional requirements
    - Messages have the following format ...
    - Message arrival generates an interrupt
    - Message arrival patterns obey …



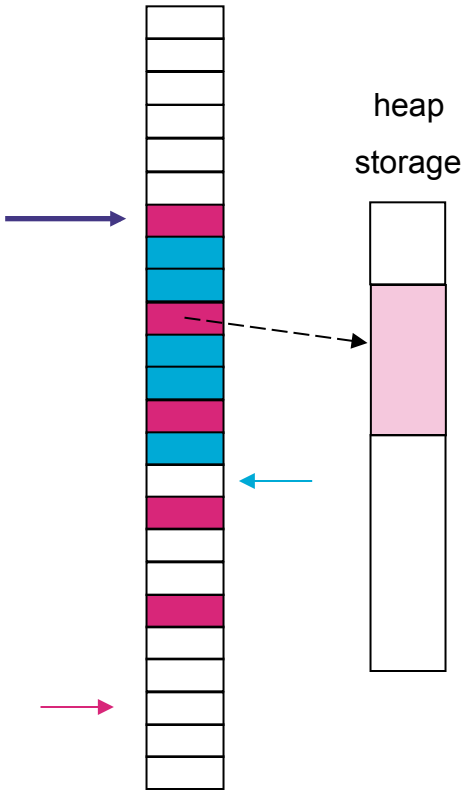
Message Merge

- Architecture design



Message Merge

- Detail design — message buffer
  - Interface
    - add message A or B
    - next cell
    - next message
  - Message pool
    - fixed size circular buffer
    - fixed size cells
    - three messages per cell
  - Message storage
    - heap storage



1.3 Architecture Design Phase

Technical goals

- Identify the principal components that make up the system and their (internal and external) interfaces
- Ensure compliance with respect to expectations captured in the requirements specification
- Understand, assess, and control risks
- Ensure predictability of the development process through accurate planning

Deliverables

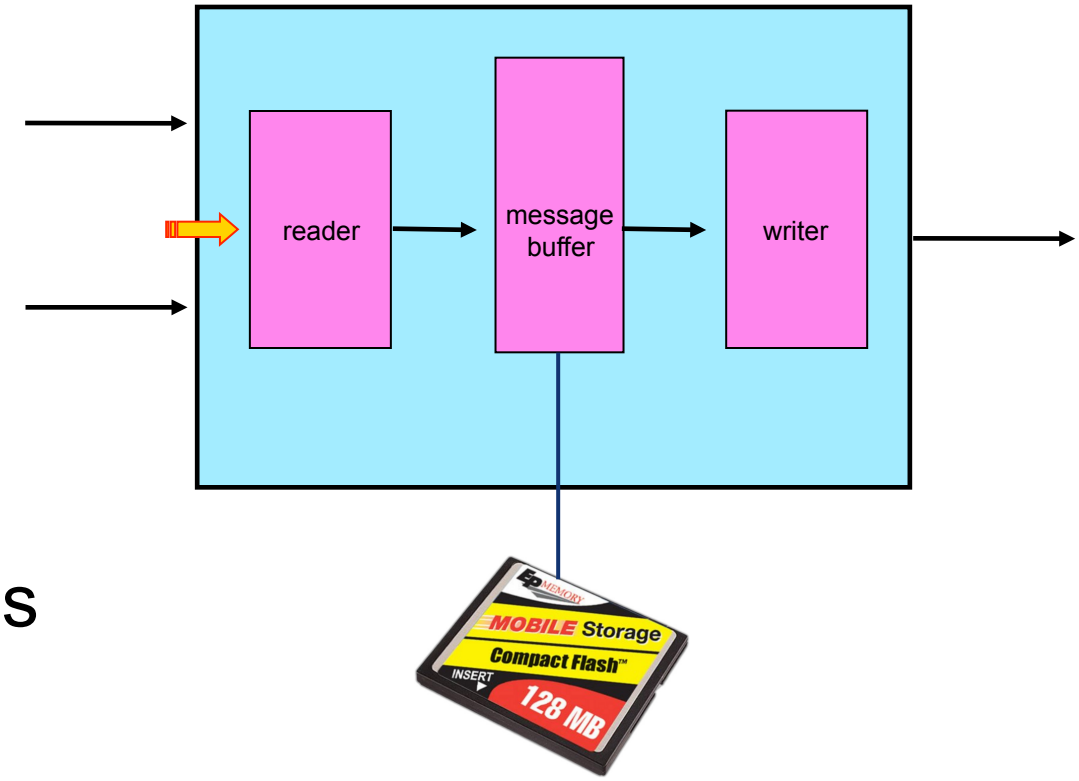
- Design diagram
  - including software to hardware allocation
- Component specifications
  - including binding specifications (COTS)
- Interface specifications
- Design analysis and evaluation
  - hazard analysis and modeling
  - risk assessment and containment
  - constraint evaluation and validation

Formative Concerns

- Design space
  - non-functional requirements limit the range of options
  - hardware selection introduces new constraints
  - purchasing decisions can reshape the design
- Requirements/design trade offs may alter the design space
- Sufficient detail is achieved when all critical design considerations can be addressed

Message Merge Revisited

- Storage analysis
  - extra storage
    - needed?
    - available?
    - fast enough?
- Failure analysis
- Timing analysis
- Code size analysis



Ancillary Concerns

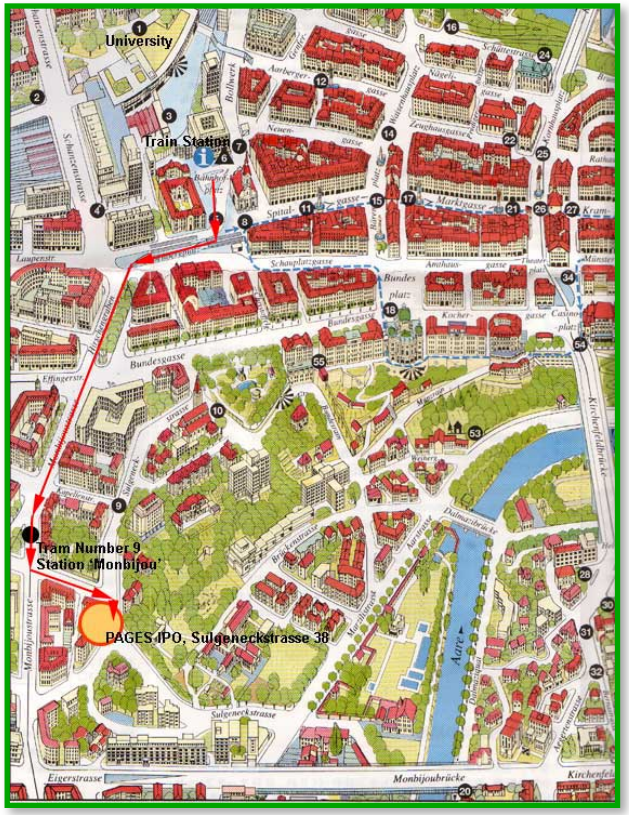
- Accurate planning requires the availability of a relatively complete architectural specification
- Idiomatic patterns of software organization are usually made evident by the design diagrams
- This facilitates design reuse and maintenance

1.4 Software Architecture as Artifact

- The architecture of a system is an abstract description of its structure and behavior
- The level of abstraction is chosen such that all critical design decisions are apparent and meaningful analysis is feasible
- A complete software architecture is usually specified by a combination of
  - design diagrams
  - component specifications

Design Diagram

- A design diagram is an abstraction of the system structure
- It is a map of the system organization at a particular level of abstraction
- It names of the principal system components and relations among them
- It hints at the operational behavior of the system (constrains what the system can do)
- It may provide implementation directives



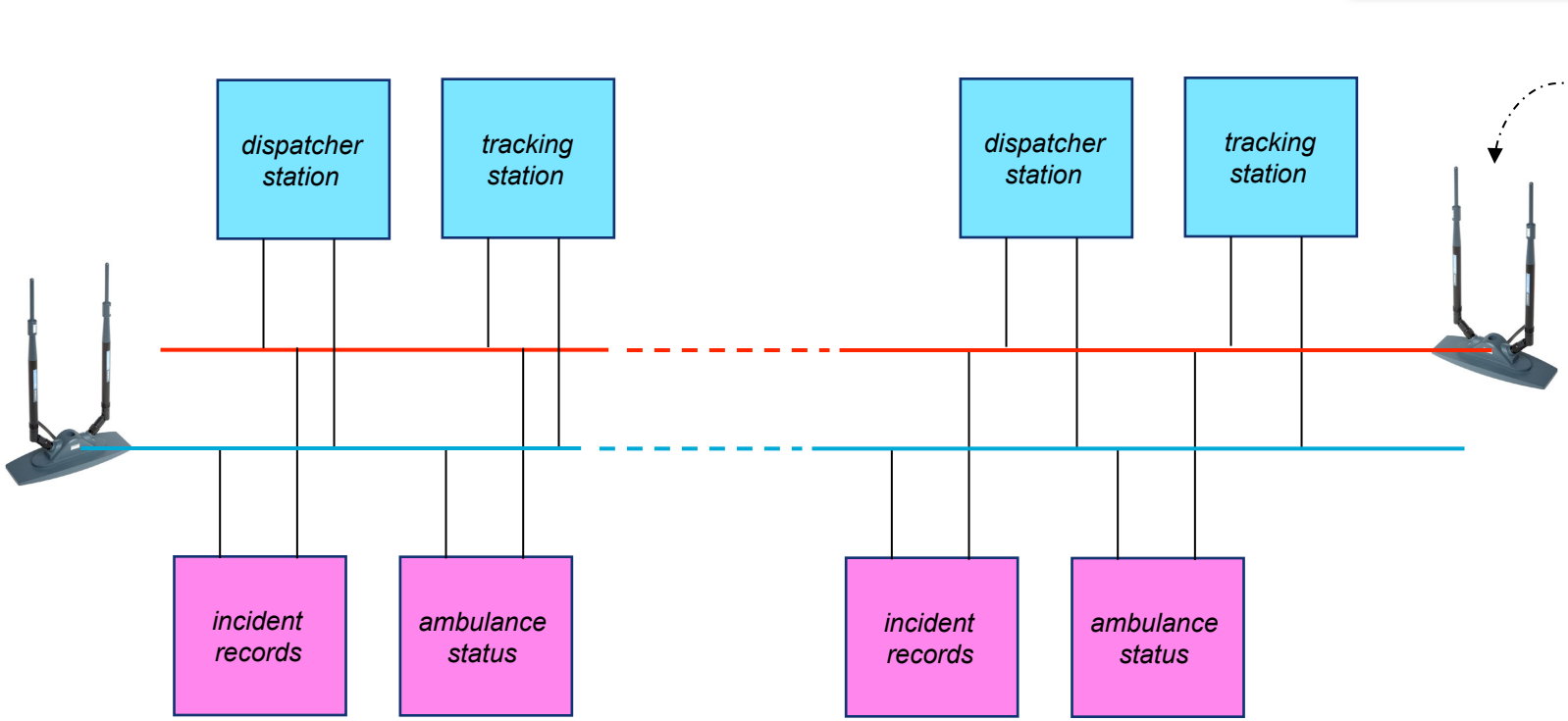
Design Diagram Caveats

- It is not an exact reflection of the code organization
- It is (by and large) independent of the syntax of any particular implementation language
- It is defined in terms of a set of constructs, selected by the designer, and believed to be implementable in the target language
- It has limited ability to express behavior
- It must be complemented by other specifications

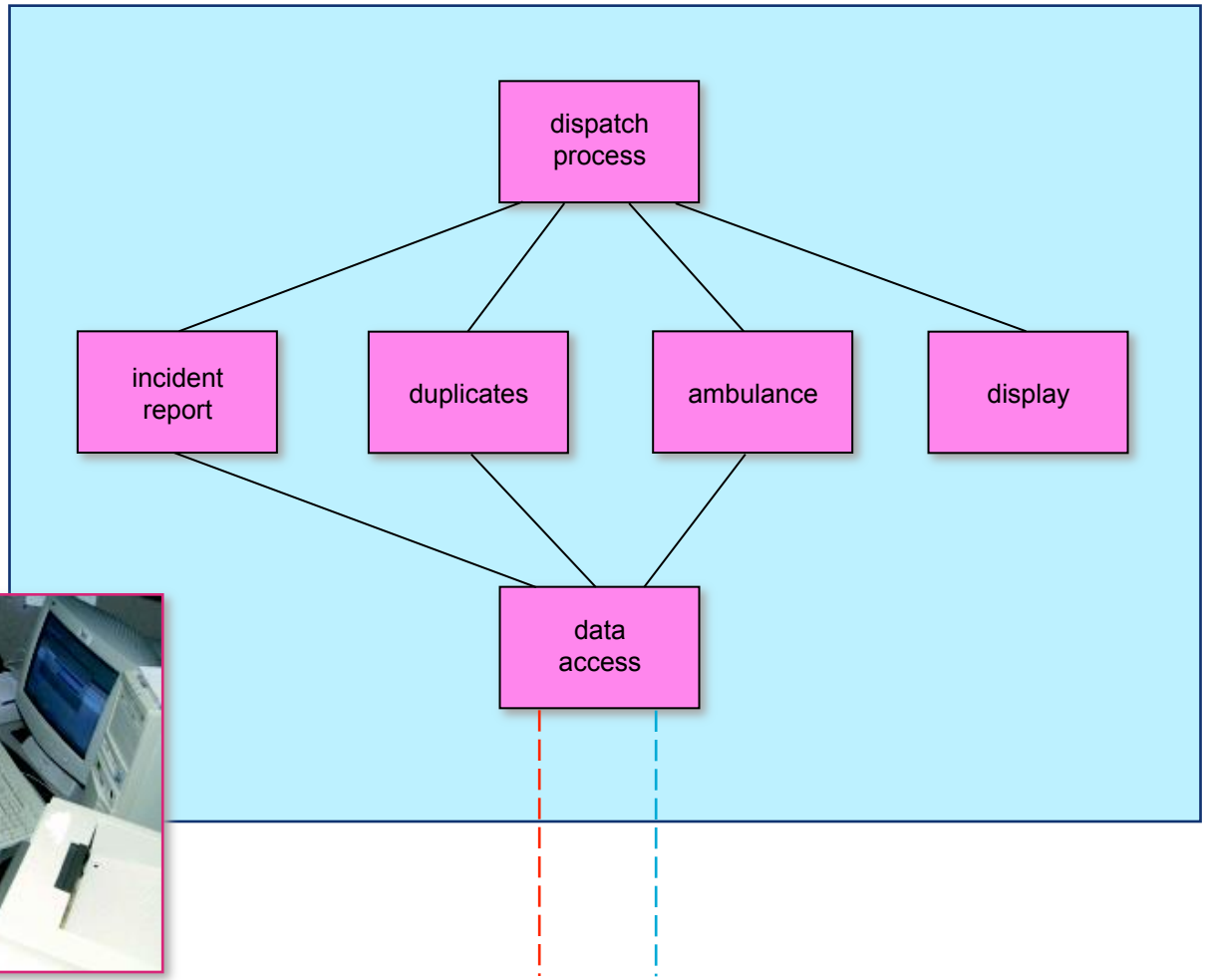
Design Diagram Role

- Important design instrument
  - captures key design decisions
    - components and their relations
    - system and underlying support
    - system and environment interactions
  - serves as an effective design communication vehicle
  - requires a low cost maintenance effort
  - is useful even after the fact
- Guides analysis and evaluation
- Facilitates project planning

Ambulance Dispatch



Dispatcher Station



Dispatch Process Analysis

- Dependability
- Survivability
- Response time
- Storage requirements
- etc.
- Timeliness
- Duplicate dispatching
- Auditing capacity
- Ambulance status accuracy

Component Specifications

- Component specifications supply the details needed to provide a complete system description at this particular level of abstraction
  - interface syntax
  - interface semantics
- The structural and behavioral properties captured by the component specifications must be consistent with the design diagram

Incident Report

- State
  - dispatcher name and id
  - date and time
  - location
  - nature of the incident
  - caller id
- Methods
  - define location
  - …
- Semantics
  - …

incident report

Managing Complexity

- Multiple architectural views
  - different levels of abstraction
  - different perspectives
- Capturing assumptions
  - structure and interfaces (design diagram)
  - semantic invariants
- Behavior patterns
  - system and/or environment
  - specialized diagrams and other notations
    - overlaid specifications

1.5 Technical Implications

- Analysis and evaluation
  - functional requirements verification
  - constraints evaluation
    - direct assessment
    - investigative targets
- Project management
  - hazard and risk analysis
  - time, effort, and code size estimation
  - activity scheduling

Traceability

- Traceability is a property of the software development process
  - refers to the ability to relate elements of a specification to those design components that ensure their satisfiability
  - relations that are precise and narrow in scope simplify analysis
  - relations that are diffused often entail complex analysis
- Specifications may be
  - functional or non-functional
  - part of the system requirements or the byproduct of design
- Traceability is a useful tool, but not a substitute for verification
- Most traceability chains run through the software architecture
  - requirements to design
  - design to code

Traceability to Requirements

- Most functional requirements must be traceable to the architecture
- All non-functional system-level requirements must be traceable to the architecture
  - quality of service, real-time constraints, dependability, allocation to physical resources, etc.
- Most non-functional requirements that relate to physical components, coding processes, protocols, installation procedures, etc. bypass the architecture
- Even with tool support, documenting traceability is costly
- Requirements affecting large areas of the design
  - render traceability efforts meaningless
  - demand systematic analysis and verification

Traceability to Code

- The software architecture is an abstraction of the code organization
  - most traceability between code and architecture requires no additional documentation
  - explicit traceability is required in some cases
    - abstract communication and coordination mechanisms
    - implementation directives
    - code elements and design elements that do not have one to one correspondence
    - non-functional requirements imposed by the design
- Requirements/code traceability is derived from the composition of two traceability mappings
  - requirements/architecture
  - architecture/code

Review Process

- The software architecture must be analyzed and evaluated by the design team
  - functional correctness
  - constraints satisfiability
  - system life-cycle issues
- Reviews
  - perform quality assurance
  - acquire measurement and evaluation data
  - exercise control
  - ensure cost effectiveness of the software development process
    - by cutting down on rework
    - by preventing error propagation to successive phases

Secondary Objectives

- Education
- Expertise development

- Project continuity
- Project visibility

- Motivation
- Standards enforcement
- Exposure of the design rationale

Review Styles

Reviews are a product adoption process

- Working session
- Walkthrough
- Inspection
- Project review
- Audit

Project Review Strategy

- Introduce the context
  - build expectations and a mental map for the audience
  - build a bridge to the system's mission
  - fully explain the interfaces
- Explain the structure
  - introduce the major system components
  - when subsystems are defined, follow the structure top-down and depth-first
- Explain the behavior
  - use a fixed repertoire of processing scenarios and predefined variations

Project Review Strategy

- Review the individual component specifications
  - relate the explanations to the selected repertoire of processing scenarios
- Justify the design
  - discuss properties critical to correct system execution
  - identify hazards and demonstrate their impossibility
- Perform other narrowly focused reviews
  - critical issues often warrant separate in depth reviews

Conclusions

- Software architecture
  - is key to any software development effort
  - is critical for ensuring the success of the undertaking
    - technically
    - managerially
  - ties to requirements in one direction and to implementation and testing in the other
  - is important to develop and maintain even after the fact