CSE 210: The Role of Software Architecture in the Development Process

Ingolf H. Krueger
ikrueger@ucsd.edu, http://sosa.ucsd.edu

with contributions from Bernhard Rumpe

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA

California Institute for Telecommunications and Information Technologies
La Jolla, CA 92093-0405, USA
Overview

- Background and Motivation
- The Notion and Role of Architecture
- Architecture vs. System Properties
- Architectural Aspects
- Modeling and Documenting Architectures
- From Domain Models to Architectures
- Architecture Evaluation
- Summary and Outlook
“»Buy, don’t build« is the anthem of the software community today. But buying means less control over every aspect of a system’s development. How can this loss of control be reconciled with our desire for quality? Part of the answer lies in our assertion that, for large systems, quality lives primarily in the architecture.”

[BCK98]
Architecture? What Architecture?

- Hardware
- Hardware Abstraction
- Middleware
- Application Server
- Legacy
- GUI-Coupling
- GUI
Architecture Improvement

- GUI
- GUI-Coupling
- Application Server
- Middleware
- Hardware Abstraction
- Hardware

...
Overview

• Background and Motivation

• The Notion and Role of Architecture

• Architecture vs. System Properties

• Architectural Aspects

• Modeling and Documenting Architectures

• From Domain Models to Architectures

• Architecture Evaluation

• Summary and Outlook
A **Software-Architecture** describes the decomposition of a system into

*units / components / subsystems*,

and their

*connections / interactions / relationships*

observing

*quality requirements / design guidelines / constraints*

More precisely: interfaces (offered/used)
- Syntax (signals/methods, typing)
- Behavior (protocols)

Often: (arbitrary) distinction between “functional” und “non-functional” properties
The Role of SW-Architecture

• Bounds leeway for implementation

• Fosters (or impedes!) system quality

• Supports the critical system services

• Defines starting point for
  - Change management
  - Product families
  - Division of work
The Role of SW-Architecture

The Classic: ISO/OSI-Layered Architecture

- Clear assignment of responsibilities to layers
- What functionality does the component perform and where is it located?
- Who develops what part of the system?
- Positioning of business models
- Clear interfaces between layers
- Call constraints (no “skipping” of layers)
The Role of SW-Architecture

“CORBA” (simplified)

• Clear interfaces, responsibilities, decoupling
• Infrastructure for communication, distribution
• Implementation strategy

Component → Component → Component → Component → ... → ORB
Example: “Pipes & Filters”
Example: Compiler

Source Code -> Scanner -> Parser -> Code-Generator

Token

Abstract Syntax Tree

Symbol-Table

Error Trace

Error Output

Target Code

Error Messages
Example: “Interpreter”

[Diagram showing the components of an interpreter system with labels: Program, Virtual Machine, Base System, and "User"]
The Role of SW-Architecture

- The selection of an adequate architecture is a key success factor in system design

- Transparently structured architecture is basis for:
  - Project organization
  - Complexity management
  - Reuse
  - Component- and service-oriented development
  - System comprehension beyond team boundaries
  - Manageable system evolution

⇒ Make architecture description part of every software project!
The Role of SW-Architecture

A good SW-Architecture

• Brings stability to the system
  ⇒ small modifications to the requirements induce small modifications to the system

• Has “conceptual integrity”
  ⇒ Balance
  ⇒ Simplicity
  ⇒ Elegance, Clarity
  ⇒ Practicality

Employ
• Dedicated SW-Architect(s)
• Architecture reviews
Architecture vs. Process

- Business Processes, Use Cases, User Stories, Requirements, Risks
- Domain Model, Architecture
- Architecture Document
- Implementation
Overview

- Background and Motivation
- The Notion and Role of Architecture
- Architecture vs. System Properties
- Architectural Aspects
- Modeling and Documenting Architectures
- From Domain Models to Architectures
- Architecture Evaluation
- Summary and Outlook
Architecture Influences System Properties

see also: [DW98]

- reusable
- changeable
- System
- portable
- testable
- implementable
Architecture Influences System Properties

see also: [DW98]

- functional
- extensible
- user friendly
- performant
- economical
- available
- safe/secure
- dependable
- scalable
Overview

- Background and Motivation
- The Notion and Role of Architecture
- Architecture vs. System Properties
- Architectural Aspects
- Modeling and Documenting Architectures
- From Domain Models to Architectures
- Architecture Evaluation
- Summary and Outlook
Who is involved? Stakeholders!

Manager
Customer
End User
Administrator
Maintainer
Operator
Evaluation Team
Implementer
Tester
Marketing

...
Forces Influencing Architectures

Requirements

Technology Base

Problem Domain Architecture

adapted from [H03]
Architectural Aspects

Functional requirements
Models of structure and behavior

Source code organization
File structure
Configuration information

Aspects of distribution and concurrency
Response times
Throughput

Mapping of executables to processors
System platform
Installation

see also: [RUP98]
# Architectural Aspects and Models

<table>
<thead>
<tr>
<th>Model View</th>
<th>Elements Considered</th>
<th>Description Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data Structures</td>
<td>Entity Relationship-/Class-Diagrams</td>
</tr>
<tr>
<td>Process</td>
<td>Processes, Communication</td>
<td>Dataflow Diagram</td>
</tr>
<tr>
<td>System Behavior</td>
<td>Events, States, Control Flow, Message Histories</td>
<td>Statecharts, Message Sequence Charts</td>
</tr>
<tr>
<td>Structure</td>
<td>Objects, Modules, Classes, Components</td>
<td>Object-/Class-/Component-Diagrams</td>
</tr>
<tr>
<td>Component Behavior</td>
<td>Interfaces, Event Traces, Message Histories, Control Flow</td>
<td>Statecharts, Message Sequence Charts</td>
</tr>
</tbody>
</table>

Overview

• Background and Motivation
• The Notion and Role of Architecture
• Architecture vs. System Properties
• Architectural Aspects
• Modeling and Documenting Architectures
• From Domain Models to Architectures
• Architecture Evaluation
• Summary and Outlook
Modeling and Documenting Architectures

How to model and implement

- Units / components / subsystems,
- Connections / interactions / relationships,
- Quality attributes / development guidelines / constraints

adequately?

⇒ Approaches:

- Architecture Description Languages (ADLs)
- Architectural styles and patterns
- Domain specific architectures
- Infrastructures
How to model and implement:

• **Units / components / subsystems**, 
• **Connections / interactions / relationships**, 
• **Quality attributes / development guidelines / constraints**

adequately?

⇒ Approaches:

• *Architecture Description Languages (ADLs)*
• *Architectural styles and patterns*
• *Domain specific architectures*
• *Infrastructures*

Modeling and Documenting Architectures

- Unicon/Wright (Garlan et al., CMU)
- Darwin (Kramer et al., Imperial College)
- Rapide (Luckham et al., Stanford)
- UML-RT (Rational)?!
- Service-ADL (Krueger et al., UCSD)
- ...

Implementation
Architecture vs. Process

- Domain Model, Architecture
- Business Processes, Use Cases, User Stories, Requirements, Risks
- Architecture Document
- Implementation
Methodological Treatment

class diagram:

- build model of application domain
- structural system view
- conceptualization as starting point for detailed design

Transition to behavior analysis and specification
An Architecture Document Outline

1. Introduction and Background Information
2. Glossary of Terms Used in the Document
3. Logical View
4. Deployment View
5. Process View
6. Implementation View
7. Supplementary Information
8. [Requirements Tracing]
9. [Risk Tracing]
10. Index
11. [Appendix]
1.1 High-level Overview of the Context, Goals and Major Constraints of the Architecture

Highly modular system for online video rental supporting high volumes of renting transactions for new releases. System needs to be extensible towards a general online rental system...

1.2 Customer/approval authority

Griswold Industries, Inc.

1.3 Organization and individuals developing the architecture

CSE210

1.4 Version of the Architecture

4.0

1.5 Scheduled Date of Completion

April 14, 2005
2. Glossary of Terms Used in the Document

**Account:** Information about customers, required for maintaining business relationship with the online store

... 

**Customer:** Person or organization accessing the online store

... 

**Order:** Sequence of line items describing items purchased or rented through the online store

... 

**SSL:** Secure Socket Layer; standardized protocol for secure information exchange over the Internet

...
3. Logical View

3.1 High-Level Operational View

3.1.1 High-level Component Diagram Showing a Logical Decomposition of the System into Functional Areas.

3.1.2 Explanatory Text

3.2 Use Cases/ User stories/ Services (on the logical level)

3.3 Business Processes

3.4 Domain Model

3.5 Component View

3.5.1 Component Diagram

3.5.2 Rationale
4.1 System Configuration

Web clients (IE, Firefox), Appserver farm, Database (DB2), Message Bus (WebSphere MQ) for communication between Appservers and Database, Linux for Appservers and Database, …

4.2 Deployment Diagram

4.3 [Event Traces]

4.4 [State Machines]

4.5 Database Layout
5. Process View

5.1 Parallel Processes

5.2 Performance Constraints per Use Case/Scenario

5.3 Security Constraints per Use Case/Scenario
6. Implementation View

6.1 Choice of Programming Language(s) + Rationale

6.2 Coding Standard

6.3 File Structure

6.4 Configuration Management
7.1 Standards Used

7.2 Technology Forecast

7.3 Findings

7.4 [Migration Strategies]
What are Patterns?

A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate.

[POSA96]
What are Patterns?

- Describes one proven solution for a recurrent design problem
- Defines the context for the solution’s applicability

Diagram:
- Pattern
  - Architectural Pattern
  - Design Pattern
  - Idiom

Granularity:
- Coarse
- Granularity
- Fine
What are Patterns?

- Describes one proven solution for a recurrent design problem
- Defines the context for the solution’s applicability

Pattern

Architectural Pattern

Design Pattern

Idiom

coarse  granularity  fine
Architectural Patterns

“An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.”

[POSA96]
Architectural Patterns

Generalization yields pattern “layered architecture”
Architectural Patterns – Schema

• Name:
  – Concise identifier for the pattern

• Context:
  – Under what circumstances is the pattern applicable?

• Problem:
  – What problem does the pattern solve?
  – What are the tradeoffs?

• Solution:
  – Structure
  – Behavior
  – Implementation

• Application examples, variants, consequences
Example: Subsystem Controller

• Context & Problem:
  - Complex system with large number of communication links between individual components
  - Tight coupling of components
  - Coherent subsystems can be identified

• Solution:
  - Decompose hierarchically into subsystems
  - Introduce controller components to decouple subsystems
  - Every subsystem consists of one controller and a set of further subsystems
  - Controllers manage communication
    - of subsystems on the same level of abstraction
    - of subsystems on lower levels of abstraction
  - Subsystems interact exclusively via “their” controller
Example: Subsystem Controller

• Structure:

Communication proceeds exclusively via the controllers’ interfaces
Example: Subsystem Controller

• Behavior:

Use efficient communication mechanisms!
Example: Subsystem Controller

- **Behavior:**

  ![Diagram of Subsystem Controllers](image)

  

  Communication across subsystem boundaries, may have to use less efficient communication mechanisms.
Example: Subsystem Controller

- **Example Applications:**
  - Telecommunication switches

- **Consequences:**
  - Clear communication structure
  - High degree of decoupling
  - Possibly less efficient
    - due to communication across multiple layers of hierarchy
    - in case of frequent communication across subsystem boundaries on the same layer of hierarchy
  - Starting point for performance analysis regarding communication paths
  - Starting point for separation between “hard” and “soft” real-time constraints:
    - **⇒** Within subsystems: “fast” communication
    - **⇒** between controllers: “slow” communication
Example: Subsystem Controller

- Related Patterns:
  - Facade (see [GOF95])

The facade shields components within a subsystem from the environment. Components within the subsystem can interact directly (no detour via the facade).
Example: Subsystem Controller

- Related Patterns:
  - Mediator (see [GOF95])

The mediator coordinates the communication among a set of components. The components interact exclusively via the mediator.
Overview

- Background and Motivation
- The Notion and Role of Architecture
- Architecture vs. System Properties
- Architectural Aspects
- Modeling and Documenting Architectures
- From Domain Models to Architectures
- Architecture Evaluation
- Summary and Outlook
What’s in a Component?

Component is unit of*

- Abstraction
- Accounting
- Analysis
- Compilation
- Delivery
- Deployment
- Dispute
- Extension
- Fault Containment
- Instantiation
- Installation
- Loading
- Locality
- Maintenance
- System Management

Systematic Component Development

• Observe constraints:
  
  - Technical Infrastructure
  
  - Legacy systems

• Iterative, hierarchy-oriented process (“divide and conquer”):

  1.) Starting point: Domain- und use case-model

  2.) Derivation of components from entities of the domain model

  3.) Definition of interfaces
Systematic Component Development

- Observe constraints:
  - Technical Infrastructure
  - Legacy systems

- Iterative, hierarchy-oriented process ("divide and conquer"):
  1.) Starting point: Domain and use case model
  2.) Derivation of components from entities of the domain model
  3.) Definition of interfaces

  - Who/what manages whom/what?
  - What is logically/functionally related (same use case/service etc.)?
  - What entities provide services to multiple others?
  - What entities depend on each other strongly/weakly?
  - What entities should be exchangeable independently from others?
  - What is going to run on the same processor/network node?
  - What parts are constrained by similar performance requirements?

2.) Derivation of components from entities of the domain model

3.) Definition of interfaces
Typical building blocks of an OO component:

- **Interface Classes**
- **Control Classes**
- **Domain Classes**

- Coordinate complex tasks within component
- Control communication with the environment ("Services")
- Conceptual entities of the application domain
Systematic Component/Service Development
Refactoring Techniques

• Given:
  - A design model of the system under consideration, or
  - An implementation of the system

• Problem:
  - The system requirements change (frequently)
  - Mandatory/desirable system qualities are missing:
    • Clear structure
    • Reusability
    • Clarity
    • Changeability
    • ...
  - The adaptation to changing requirements is difficult
  - The investment into the existing design should pay off
Refactoring Techniques

Refactoring:

- “minimally invasive” modifications to system structure

  ⇒ Strategy of small steps

- Set up adequate test suite before changing the system

- Carry out tests during and after performing the change

  ⇒ Increases confidence in correctness;

  Goal: no change of observable behavior
Refactoring Techniques

Example: Move method to superclass

```
Motor

<table>
<thead>
<tr>
<th>Left_Door_Motor</th>
<th>Right_Door_Motor</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock()</td>
<td>lock()</td>
</tr>
</tbody>
</table>

“Pull Up Method” (see [F99])

Motor

<table>
<thead>
<tr>
<th>Left_Door_Motor</th>
<th>Right_Door_Motor</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock()</td>
<td>lock()</td>
</tr>
</tbody>
</table>
```
Refactoring Techniques

• Refactoring proceeds in extremely small steps

• Each individual change is manageable

• Refactoring fits particularly well with design patterns and architectural patterns:

  1.) Evaluate current state

  2.) Select target pattern

  3.) Identify sequence of refactoring steps leading to target pattern

  4.) Perform refactoring until target pattern is reached
Overview

• Background and Motivation
• The Notion and Role of Architecture
• Architecture vs. System Properties
• Architectural Aspects
• Modeling and Documenting Architectures
• From Domain Models to Architectures
• Architecture Evaluation
• Summary and Outlook
Why Evaluate an Architecture?

- Stakeholders have different views/opinions on
  - what the system does
  - how the system should be structured
  - what documentation should look like
  - how the company/suppliers should conduct their business
  - ...

- Architecture Evaluation
  - brings different stakeholders together
  - forum for voicing concerns (can, but need not, be related to the software architecture under consideration)
  - forum for establishing common understanding of important system aspects across development/business teams
  - means for risk identification and analysis
Why Evaluate an Architecture?

• Find errors early: 55% of all errors are *made*, but less than 10% are *detected* during requirements capture and analysis.

• Find out if architecture is adequate wrt.
  - desirable/mandatory requirements
  - critical use cases
  - anticipated changes/extensions
  - budget constraints

• Find out if state of the art development and documentation approaches were used.

• Find interfaces for coupling existing architecture with legacy or new neighboring systems.

• Decide among several competing architectural alternatives.

• Determine migration path towards target architecture.
Goal-Oriented System Development

- First Step of Architecture Evaluation: determine the **specific goals** for the system under consideration
- Prioritize goals!
- What are the critical properties/requirements the architecture **must support**?
- Is the architecture **suitable** wrt. these goals/properties/requirements?
- Determine the points in the architecture that influence the critical goals
Key Elements of Architecture Evaluation

- Determine goals for the evaluation
  - why does it happen?
  - who has an interest in it?
- Build understanding of the application domain
  - where are the difficulties in building this and similar applications?
  - are standard solutions available/used?
- Build domain model if it doesn’t exist already
- Determine and prioritize goals for the system/organization
- Identify and prioritize scenarios (“critical”, “medium”, “low”)
- Play through scenarios, record adequacy of the architectural decisions
- Discuss alternative architectures and migration strategies
Evaluation Methods: Example

ATAM: Architecture Tradeoff Analysis Method*

• Presentation
  1. Present ATAM
  2. Present business drivers
  3. Present architecture

• Investigation and Analysis
  4. Identify architectural approaches
  5. Generate quality-attribute utility tree
  6. Analyze architectural approaches

• Testing
  7. Brainstorm and prioritize scenarios
  8. Analyze architectural approaches

• Reporting
  9. Present the results

Overview

- Background and Motivation
- The Notion and Role of Architecture
- Architecture vs. System Properties
- Architectural Aspects
- Modeling and Documenting Architectures
- From Domain Models to Architectures
- Architecture Evaluation
- Summary and Outlook
Summary and Outlook

• Software Architecture key success factor in Software and Systems Engineering efforts

• Dedicated architecture (document) and review indispensable for complex systems

• Domain model is central point in the architecture development process

• Several standards and notations exist for documenting architectures

• Architecture evaluation excellent starting point for architecture improvement, team collaboration, information sharing

• CSE 218 next Winter Quarter!