System Design and Architecture

Lecture 5

Software Engineering
TDDC88/TDDC93
autumn 2013

Kristian Sandahl (slides by David Broman)
Department of Computer and Information Science
Linköping University, Sweden
kristian.sandahl@liu.se
**Knowledge Areas**

- Week 36 – Requirements
- Week 37 – Planning and Processes
- **Week 38 – Design and Architecture**
- Week 39 – Testing and SCM
- Week 40 – Software Quality
A Software Life-cycle Model
Which part will we talk about today?

Validate Requirements, Verify Specification

Acceptance Test (Release testing)

System Testing (Integration testing of modules)

Module Testing (Integration testing of units)

Unit testing

Verify Implementation

Verify Module Design

Verify System Design

System Design (Architecture, High-level Design)

Module Design (Program Design, Detailed Design)

Requirements

Implementation of Units (classes, procedures, functions)

Project Management, Software Quality Assurance (SQA), Supporting Tools, Education

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part III
Architecture Styles and Patterns
Agenda - What will you learn today?

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part IV
Architecture Styles and Patterns
Part I
Basic Concepts
Why should we design a system?

Why not?

Carol
the customer

Implementation

Harry
the hacker

Requirements
Constructing a building...

I need a tower, with a big clock...

The king's requirements

Construction
Constructing a building...

The king's requirements

Architecture

Ulla

Construction

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part III
Architecture Styles and Patterns

Ulla
Constructing software...

Software is different
- No physical natural order of construction (e.g. start with the foundation of the house)
- Software is not tangible

Carol the customer ➔ Architecture ➔ Implementaiton ➔ Harry the hacker

Requirements ➔ Architecture ➔ Implementation

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part III
Architecture Styles and Patterns
Constructing software...

Abstraction

System Design
(Architecture, High-level Design)

Module Design
(Program Design, Detailed Design)

Fuzzy distinction
- Sometimes several levels
- Sometimes only one level

Requirements

Implementation

Carol
the customer

Harry
the hacker

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part III
Architecture Styles and Patterns
Why design and document software architectures?

**Communication between stakeholders**
A high-level presentation of the system.
Use for understanding, negotiation and communication.

**Early design decisions**
Profound effect on the systems quality attributes, e.g. performance, availability, maintainability etc.

**Large-scale reuse**
If similar system have common requirements, modules can be identified and reused.

(Bass et al., 2003)
Analyze and Synthesis a system (decompose and compose)

**Part I**
Basic Concepts

**Part II**
Quality Factors

**Part III**
Documenting the Architecture

**Part III**
Architecture Styles and Patterns

---

**Requirements**

- Imagine a "virtual" System
  - Divide into "virtual" modules
  - Design each module

**System Design**
(Architecture, High-level Design)

**Module Design**
(Program Design, Detailed Design)

**Implementation of Units** (classes, procedures, functions)

**System Testing**
(Integration testing of modules)

**Module Testing**
(Integration testing of units)

**Acceptance Test**
(Release testing)

**Unit testing**
Analyze and Synthesis a system (decompose and compose)

Imagine a "virtual" System

- System Design (Architecture, High-level Design)
- Module Design (Program Design, Detailed Design)
- Implementation of Units (classes, procedures, functions)

a "concrete" System

- Acceptance Test (Release testing)
- Module Testing (Integration testing of units)
- Unit testing

Design is an iterative process!
- Throw away Prototyping
- Evolutionary Prototyping

Requirements

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part III
Architecture Styles and Patterns
Box-and-line diagrams...

Encryption / Decryption

Logging

Packet Handler Session Handler

Interface

Identification & Authentication

User Database

Module, Subsystem, Element, Entity, Component... (many names)

Relationship

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part III
Architecture Styles and Patterns
Architectural views

**Implementation view**
- Cryptographic Module
  - Modules, packages, classes, functions, source files

**Execution view**
- Server
- Client
  - Components, connectors, sub-systems (box-and-line)

**Deployment view**
- On different machines?
- One machine? Different CPUs?
A developer’s perspective:
1. What are we going to develop?
2. Where is the code?

Package
- Organize work
- Compile together
- Name space

Physical code, file, or library

dependency
Coupling - dependency between modules

Uncoupled - no dependencies

Loosely coupled - few dependencies

Highly coupled - many dependencies

What do we want?

Low coupling. Why?
- Replaceable
- Enable changes
- Testable - isolate faults
- Understandable
Cohesion - relation between internal parts of the module

Low cohesion - the parts e.g. functions have less or nothing in common.

Medium cohesion - some logically related function. E.g. IO related functions.

High cohesion - does only what it is designed for

What do we want?
High cohesion. Why?
- More understandable
- Easier to maintain
Part II
Quality Factors
Several factors - sometimes overlap

Non-functional requirements...

- Portability
- Maintainability
- Modifiability
- Scalability
- Availability
- Performance
- Usability
- Testability
- Reliability
- Safety
- Non-functional requirements...
Performance - timing

Timing
- Throughput
- Response time (interactive system)

Can our architecture be parallelized?

Scale up...

Scale out...
Security

Confidentiality
- Only authorized users can read the information
- E.g. Military

CIA

Integrity
- Only authorized users can modify, edit or delete data.
- E.g. bank systems

Availability
- Right information is available at the right time
- Important for everyone

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part III
Architecture Styles and Patterns
How can we validate that a safety critical system is correct?

- Formal validation?
- Testing?

Design so that all safety critical operations are located in one or few modules / subsystems.
Modifiability - cost of change

**What can change?**
- Platform?
- Function?
- Protocols?
- Environment?

**When can change?**
- Source code?
- Compiler option?
- Library?
- Setup config?
- At runtime?

**Portability**
- OS (UNIX, Windows, Real-time OS, Mac,...)
- Memory consumption
- CPU (Big-endian?)
- Computation power

**Maintainability**
- Low coupling
- Consistent with code
- Enough details
Usability - How easy is it and what support exists to perform a task

- Easy to learning system features
  - E.g. a web-browser

- Using the system efficiently
  - E.g. Latex, or UNIX shells and pipes
At least 40% of the cost of well-engineered system is due to testing (Bass et. al., 2003)

What about cohesion and coupling?
Some Business Qualities

- **Time-to-market**
  - Reuse component and use commercial-off-the-self (COTS) products

- **Cost-and-benefits**
  - Use technology that the organization knows
Part III

Documenting the Architecture
### Not only one way to do it... Write from the point of view of the readers...

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Use of the architect document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements engineers</td>
<td>Negotiate and make tradeoffs among requirements</td>
</tr>
<tr>
<td>Architects/Designers</td>
<td>Resolve quality issues (e.g. performance, maintainability etc.)</td>
</tr>
<tr>
<td>Architects/Designers</td>
<td>A tool to structure and analyze the system</td>
</tr>
<tr>
<td>Designers</td>
<td>Design modules according to interfaces</td>
</tr>
<tr>
<td>Developers</td>
<td>Get better understanding of the general product</td>
</tr>
<tr>
<td>Testers and Integrators</td>
<td>Specify black-box behavior for system testing</td>
</tr>
<tr>
<td>Managers</td>
<td>Create teams that can work in parallel with e.g. different modules. Plan and allocate resources.</td>
</tr>
<tr>
<td>New software engineers</td>
<td>To get a quick view of what the system is doing</td>
</tr>
<tr>
<td>Quality assurance team</td>
<td>Make sure that implementation corresponds to architecture.</td>
</tr>
</tbody>
</table>
**When to document?**

- **Time**
  - Requirements
  - Initial design
  - Design iterations
  - Implementation
  - After implementation (consistent with code?)

**Part I**
- Basic Concepts

**Part II**
- Quality Factors

**Part III**
- Documenting the Architecture
- Architecture Styles and Patterns
Case: Vote counting system

“It’s not who votes that counts, It’s who counts the votes”

Josef Stalin

Correctness criterion
No of voters marked as voted = No of votes in ballot box

Voting register
(identification of voters)

Problem: More ballots than marked voters

Comparing a voted voter against with a new identity.

voter.getIdentification() == identification()
voter.getIdentification().equals(identification())

Requirements
- Mimicked traditional voting
- Anonymity

Ballot box
(store the actual votes)

Comparing references, not values
What to document?

System Overview
A brief description / overview of the system, stating

- Who the users are
- The main requirements / constraints including important quality factors
- Any important background information

Give the reader a "mental" model of the system!
What to document?

- **System Overview**
- **Different structural Views**
- **Mapping between views**
- **Behavior**
- **Rationale**

**Implementation view**
- Cryptographic Module
- Modules, packages, classes, functions

**Execution view**
- Server
- Components, connectors, sub-systems
  (box-and-line)

**Deployment view**
- Client
- On different machines?
- One machine?
  Different CPUs?
What to document?

For each view
- Give an overview of the view
- Create a view diagram
- Describe each element
- Describe each relation between elements
- Describe interfaces
What to document?

Mapping between views

- Describe how the view relates (all views describe the same system – what is common)
- Central to decrease confusion and increase understanding
- Especially important if not one-to-one mapping
What to document?

Behavior
- Views state structural information
- Behavior for elements or between elements can be described using e.g., texts, sequence diagrams and state machines.

![Behavior Diagram]

- calcPrice
- getQuantity
- getProduct
- aProduct
- getPriceDetails
- calculateBestPrice
- getDiscountInformation
- calculateBestPrice
What to document?

Why the architecture is the way it is

- Rationales for views, interfaces, etc.
- Architecture implication due to certain requirements
- Expected effect when changing requirements or adding new ones
- Constraints for the developer when implementing the solution
- Design alternatives that were rejected and the rational for doing so.

In general

- Why a decision was made
- What the implication is to change it
Part IV
Architecture Styles and Patterns
Example of styles and patterns

- Client-Server
- Layering
- Pipes-and-filters
- Model-View-Control (MVC)
- Repository
- Peer-to-Peer

Discussed today
1. Client-Server

The clients need to be aware of the server.

Clients initiate communication
The clients need to be aware of the server.

Clients initiate communication
### 1. Client-Server

#### Two-Tier, Thin-client
- **Client**
  - Presentation layer
- **Server**
  - Business Layer
  - Data management

#### Two-Tier, Fat-client
- **Client**
  - Presentation layer
- **Middle-ware**
  - Business Layer
- **Server**
  - Data management

#### Three-Tier
- **Client**
  - Presentation layer
- **Middle-ware**
  - Business Layer
- **Server**
  - Data management

- **Pros**
  - + Distribute workload on clients
  - + Map each layer on separate hardware
  - + Possibility for load-balancing
- **Cons**
  - - Heavy load on server
  - - Significant network traffic
  - - System management problem, update software on clients
  - - System management problem, update software on clients
  - - Possibility for load-balancing
2. Layers

In a “pure” layered model, only the immediate below layer can be accessed.

Layer bridging – can access lower than the closest one.
2. Layers

Pros

- Easy reuse of layers
- Support for standardization
- Dependencies are kept local - modification local to a layer
- Supports incremental development and testing

Cons

- Could give performance penalties
- Layer bridging loses modularity
3. Pipes and Filters

Example: UNIX Shell

```
ls -R | grep "html$" | sort
```

Example: A Compiler

```
lexer -> parser -> semantic analysis -> Intermediate Code Generation
```

```
Intermediate Code Generation -> Optimization -> Code Generation
```

```
ls -> grep -> sort
```

```
ls -R | grep "html$" | sort
```

```
ls -> grep -> sort
```

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part III
Architecture Styles and Patterns
Case: SOA and Amazon

Before 2001...

Two-tier architecture

- Customer web clients
- Webserver + DBMS

Problems
- Scaling the DBMS
- Too complex software to maintain and develop

After 2001...

- Customer web clients
- Web servers
- Services
- Partner Companies (1 million)

Key Success Factors
- Data encapsulated with business logic.
- No data sharing between services
- Independent dev teams for each service
- Developers have operational responsibility (you build, you run)

CTO Werner Vogels blog
www.allthingsdistributed.com
Summary- What have we learned today?

Part I
Basic Concepts

Part II
Quality Factors

Part III
Documenting the Architecture

Part IV
Architecture Styles and Patterns