MTAT.03.094
Software Engineering

Lecture 07: Architecture and Design I

Dietmar Pfahl
email: dietmar.pfahl@ut.ee

Fall 2021
Schedule of Lectures

Week 01: Introduction to SE
Week 02: Requirements Engineering I
Week 03: Requirements Engineering II
Week 04: Analysis
Week 05: Development Infrastructure
Week 06: Continuous Development and Integration
Week 07: Project Estimation / Architecture and Design I
Week 08: Architecture and Design II
Week 09: Verification and Validation I
Week 10: Verification and Validation II
Week 11: Refactoring (and TDD)
Week 12: Agile/Lean Methods
Week 13: Industry Guest Lecture
Week 14: Course wrap-up, review and exam preparation
Week 15: Reserve time slot (no lecture scheduled as of today)
Acknowledgements

Textbooks/Slides:

- Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren (in German), 2013
- Hans van Vliet: Software Architecture, Free University of Amsterdam, Lecture 2008
- Richard Taylor et al.: Software Architecture, University of California at Irvine, Lecture 2011
- George Fairbanks: Just Enough Software Architecture, 2012 (Video: https://www.youtube.com/watch?v=x30DcBfCJRI)
- Tutorials by Derek Banas (on YouTube)
Structure of Lecture 07

• Why Architecture?
• Terminology: Architect, Architecting, Architecture
• Viewpoints and View Models
• Notation
• Architecture & Design Patterns
Two Telephone Systems

Plain Old Telephone System
- Feature:
  - Call subscriber
- Architecture:
  - Centralized switchboard
- Good qualities

Same feature

Skype
- Feature:
  - Call subscriber
- Architecture:
  - Peer-to-peer software
- Good qualities

Architects pay more attention to qualities that arise from architecture choices.
Two Telephone Systems

**Plain Old Telephone System**
- Feature:
  - Call subscriber
- Architecture:
  - Centralized switchboard
- Good qualities
  - Works during power outages
  - Reliable
  - Emergency calls get location information

**Skype**
- Feature:
  - Call subscriber
- Architecture:
  - Peer-to-peer software
- Good qualities
  - Scales without central hardware changes
  - Easy to add new features (e.g., video calling)

Architects pay more attention to qualities that arise from architecture choices.
## Two Telephone Systems – Pro’s and Con’s

Which one is better?

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Landline Phone</th>
<th>VoIP (Skype)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Outage Tolerant</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Reliable</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Scalable</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Extendable (new features)</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Two Telephone Systems – Pro’s and Con’s

Which one is better?

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Landline Phone</th>
<th>VoIP (Skype)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Outage Tolerant</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Reliable</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Scalable</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Extendable (new features)</td>
<td>-</td>
<td>++</td>
</tr>
</tbody>
</table>
Trade-Offs and Decision-Making

Telephone system for a Fire Brigade Station:

Which one is better?

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Landline Phone</th>
<th>VoIP (Skype)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Outage Tolerant</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Reliable</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Scalable</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Extendable (new features)</td>
<td>-</td>
<td>++</td>
</tr>
</tbody>
</table>
Trade-Offs and Decision-Making – Template

Telephone system for a Fire Brigade Station:

- Because *Power Outage Tolerance* is more important than *Scalability* for this system, we choose a *Landline Phone*, accepting *a higher cost for adding new subscribers*.

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Landline Phone</th>
<th>VoIP (Skype)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Outage Tolerant</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Reliable</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Scalable</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Extendable (new features)</td>
<td>-</td>
<td>++</td>
</tr>
</tbody>
</table>
Trade-Offs and Decision-Making –
Template for capturing design rationales:

Telephone system for a **Fire Brigade Station**:

- Because *Quality Attribute 1* is more important than *Quality Attribute 2* for this system, we choose *technical (design/architecture) option*, accepting *drawback*

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Landline Phone</th>
<th>VoIP (Skype)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Outage Tolerant</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Reliable</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Scalable</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Extendable (new features)</td>
<td>-</td>
<td>++</td>
</tr>
</tbody>
</table>
Trade-Offs and Decision-Making –
Template for capturing design rationales:

Telephone system for a University:

- Because `<Quality Attribute 1>` is more important than `<Quality Attribute 2>` for this system, we choose `<technical (design/architecture) option>`, accepting `<drawback>`

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Landline Phone</th>
<th>VoIP (Skype)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Outage Tolerant</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Reliable</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Scalable</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Extendable (new features)</td>
<td>-</td>
<td>++</td>
</tr>
</tbody>
</table>
Mind Map for Capturing Architecture/Design Rational

Capture Design Rationale

(1) Question
- What exactly is the problem?
  - Why is it relevant for the architecture?
  - What effects does the decision have?

(2) Influencing factors
- What fixed constraints do we have to consider?
- What quality goals do we have to consider?
- What risks do we have to consider?

(3) Assumptions
- What assumptions have we made?
  - What assumptions can be checked upfront?
  - What new risk may emerge?

(4) Alternatives considered
- What are promising solution alternatives?
  - How do we assess each alternative?
  - Is there a solution option that we discard explicitly?

(5) Decision
- What alternative has been chosen?
  - How is the decision justified?
  - When and by whom was the decision made?

What problems?
- P1: What network topology should be used?
- P2: What transmission technology should be used
- P3: ...?

Why relevant?
- P1-R1: Topology choice defines skills needed by the development team
- P1-R2: Topology choice defines what suppliers for 3rd party components are available
- P2-R1: Transmission technology defines the transmission speed
- P2-...

What effects?
- P1-P2: Choice of topology (P1) may interfere with choice made regarding transmission technology (P2)
- ...
Mind Map for Capturing Architecture/Design Rational

Capture Design Rationale

(1) Question
What exactly is the problem?
Why is it relevant for the architecture?
What effects does the decision have?

(2) Influencing factors
What fixed constraints do we have to consider?
What quality goals do we have to consider?
What risks do we have to consider?

(4) Alternatives considered
What constraints?
- P1-C1: available skills in staff
- P1-C2: stability of power supply
- P2-C1: ...

What quality goals?
- P1-Q1: Power outage tolerance
- P1-Q2: Reliability
- P1-Q3: Scalability
- P1-Q4: Evolvability/Maintainability (-> adding new features)

(5) Decision
What risks?
- P1-R1: ...
- ...

What are promising solution alternatives?
- How do we assess each alternative?
- Is there a solution option that we discard explicitly?
- What alternative has been chosen?
- How is the decision justified?
- When and by whom was the decision made?
- What constraints?
  - P1-C1: available skills in staff
  - P1-C2: stability of power supply
  - P2-C1: ...
- What quality goals?
  - P1-Q1: Power outage tolerance
  - P1-Q2: Reliability
  - P1-Q3: Scalability
  - P1-Q4: Evolvability/Maintainability (-> adding new features)
- What risks?
  - P1-R1: ...
  - ...

What fixed constraints do we have to consider?
- P1-C1: available skills in staff
- P1-C2: stability of power supply
- P2-C1: ...

What quality goals do we have to consider?
- P1-Q1: Power outage tolerance
- P1-Q2: Reliability
- P1-Q3: Scalability
- P1-Q4: Evolvability/Maintainability (-> adding new features)

What risks do we have to consider?
- P1-R1: ...
- ...

Quelle: Stefan Zömer, Softwarearchitekturen dokumentieren und kommunizieren
Mind Map for Capturing Architecture/Design Rationale

What assumptions?
- P1-C1-A1: skills can be learned if needed
- P2- ...

What assumptions can be checked?
- P1-C2-AC1: stability of power supply
- P2- ...

What new risks may emerge?
- PX-NR1: subscribers are more and more mobile
- ...
Mind Map for Capturing Architecture/Design Rationale

(1) Question
What exactly is the problem?
What is the context of the problem?

(2) Influencing factors
What effects does the decision have?
What are the fixed constraints to consider?
What quality goals must be considered?
What risks do we have to consider?
What assumptions have we made?
What assumptions can be checked upfront?
What new risks may emerge?

(3) Assumptions
What are promising solution alternatives?
How do we assess each alternative?
Is there a solution option that we discard explicitly?

(4) Alternatives considered
What are promising solution alternatives?

(5) Decision
What alternative has been chosen?
How is the decision justified?
When and by whom was the decision made?

Quality Attribute | Landline Phone | VoIP (Skype)
--- | --- | ---
Power Outage Tolerant | ++ | –
Reliable | + | –
Scalable | – | ++
Extendable (new features) | – | ++

What promising solutions?
- P1-S1: Centralized switch
- P1-S2: Peer-to-peer software
- P2-

How assess alternative?
- P1-SX: cross-table against quality attributes
- P2-SX: …

Solution explicitly discarded?
- P1-EX1: Ring topology
- …

Telephone system for a Fire Brigade Station: Because <Power Outage Tolerance> is more important than <Scalability> for this system, we choose a <Landline Phone>, accepting <a higher cost for adding new subscribers>.

Quality Attribute | Landline Phone | VoIP (Skype)
--- | --- | ---
Power Outage Tolerant | ++ | --
Reliable | + | -
Scalable | - | ++
Extendable (new features) | - | ++

What choice?
- P1-Choice: Centralized switch
- P2- ...

How justified?
- P1-Choice-J: Cross table + Application domain
- P2- ...

Choice made by whom/when?
- P1-Choice-WW: ...

What alternative has been chosen?
How is the decision justified?
When and by whom was the decision made?
Another Example: Chess Engine

- What runtime environment?
- Where to keep the status of a running online match?
- What application framework to use?
- Where/how to store persistent data? (database type?)
- How to implement the web application? (JavaScript?)
- How to interact client and server during a match? (status updates via regular polling? Push? …)
- How to realize the chat functionality and advertisements? (make or buy?)
Why is documenting architecture/design decisions important?

• Explains why this is a good (= suitable) architecture
  • Emphasizes qualities and criticality with regards to requirements/goals
• Provides context and background
• Prevents repeating (expensive) past steps

Yet another design rationale example:
Since avoiding vendor lock-in is a high priority, we choose to use a standard industry framework with multiple vendor implementations, even though using vendor-specific extensions would give us greater performance.
What if you don’t think architecturally?

- Developers optimize locally, miss the big picture
  - Lousy choice of frameworks, languages, ...

- Project success depends on having virtuosos in the team
  - But how many James Goslings and Jeff Deans are there?

- Poor communication
  - Idiosyncratic notations, fuzzy semantics

- Shallow (or no) analysis of design options
  - Ad hoc; no use of best practices
  - From first principles, therefore high effort
  - Little attention to tradeoffs and rationale

- Architectural patterns ignored
  - ... or incorrectly chosen
  - Squandering known-good designs

Remember:
All programs have an architecture
...
But not every architecture suits the problem to be solved by the program!
Structure of Lecture 07

- Why Architecture?
- Terminology: Architect, Architecting, Architecture
- Viewpoints and View Models
- Notation
- Architecture & Design Patterns
Terminology

- Architect – Person
- Architecting – Process
- Architecture – Product
The Role of the Architect

- Client, users
- Requirements
- Architect
- Solutions
- Developers
- Creates
- Architectural design
- Visualises
- Prescribes
- Creates
- Assesses
- Requirements
- Solutions
The Role of the Architect

- **Client, Users**: Assess
- **Architect**: Creates, Visualises
- **Developers**: Assess
- **Requirements**: Solutions
- **Architectural Design**: Prescribes
- **Construction, Co-operation**:

**Visualisation**
- Appearance, Behaviour

**Prescription**
- Requirements

**Co-operation**
- Solutions

**Assessment**
- Architectural Design
- Construction, Co-operation
Now, what is ‘Architecting’?
Non-Architecture-Driven Life-Cycle

Characteristics:

- Iteration mainly on functional requirements
- Few stakeholders involved
- No balancing of functional and quality requirements

Architecture?
Adding Architecture – The “Easy” Way

stakeholders (few)  ↔  functionality  ←→  quality

architecture

- detailed design
- implementation

agreement

development
Adding Architecture – The “Easy” Way

- stakeholders (few)
- functionality
- quality
- agreement
- development
Architecture in the Life-Cycle

Characteristics:

• Iteration on both functional and quality requirements
• Many stakeholders involved
• Balancing of functional and quality requirements
Architecture Iterations Example

Top-level:

- **usability** ⇒ separate UI ⇒ 3-tier architecture

Iteration 1
Architecture Iterations Example

**Iteration 2**

**Top-level:**
- **usability** ⇒ separate UI ⇒ 3-tier architecture

**Lower-level, within user interface:**
- **security** ⇒ authenticate users

**Lower-level, within data layer:**
- **availability** ⇒ active redundancy
Attribute-Driven Design (ADD)

An architect faces many *design issues*

These are sub-problems of the overall design problem.

Each issue usually has several alternative solutions (or *design options*).

The designer makes a *design decision* to resolve each issue.

Process of ‘Architecting’ = choosing the best options from among the alternatives.

*ADD = focus on quality attributes and do ‘Architecting’ in iterations*
Example: The type and level of **Security** in a system

- Security can be decomposed into
  - authentication (user recognition),
  - authorization (user access to data),
  - privacy (encryption of data exchanged on a public network).
ADD: Design Issues, Options and Decisions

Example: The type and level of **Security** in a system

- Security can be decomposed into
  - authentication (user recognition),
  - authorization (user access to data),
  - privacy (encryption of data exchanged on a public network).

If the architecture is for a **medical system**, then all security sub-issues must be addressed.
Example: The type and level of **Security** in a system

- Security can be decomposed into
  - authentication (user recognition),
  - authorization (user access to data),
  - privacy (encryption of data exchanged on a public network).

If the architecture is for **gaming applications**, probably not all of them are important, and could be dropped in favor of, e.g., higher performance.
Now, what is ‘Architecture’?
Architecture in Construction of Buildings
Software Architecture

• Architecture is *conceptual*.

• Architecture is about *fundamental* things.

• Architecture exists in some *context*.

Architectural descriptions are concrete, but the architecture itself is inherently conceptual, and cannot be captured in any (set of) views – nor in the code.

Abstraction !!!

NB: We can only understand qualities in context.
Software Architecture – Definition (1)

The architecture of a software system defines that system in terms of computational components and interactions among those components.

The software architecture of a system is the structure or structures of the system, which comprise **software elements**, the externally visible properties of those elements, and the relationships among them.

Software Architecture – Definition (3)

Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.

NB: ‘Principles’ includes the explicit identification and mentioning of properties (-> behaviour)

Next Lecture

- Date/Time:
  - Friday, 22-Oct, 10:15-12:00
- Topic:
  - Architecture and Design II
- For you to do:
  - **Go to Labs next week!**
  - **Submit Homework 3**
  - **Start working on Homework 4**