Design of Software Systems

Fall 2017, BSc Software Engineering

Requirements and Intro to Software Architecture

Assist. Prof. Dr. Ronald Jabangwe

Overview

- Software Requirements
- User Stories
- Software Architecture

Recap: UML Pros and Cons

- **Pros**
  - Flexible and customizable
  - Generally known thus a go-to for communication
  - Can be used for communicating software system concerns, e.g., software structure, functionality and behavior
  - Can be used to assess nonfunctional aspects, e.g., security, performance
  - Availability of tools
- **Cons**
  - Diagrams can become quite complex and thus make it difficult to effectively communicate
  - Comprehensive knowledge of the notation may be needed (at times) to understand diagrams
  - The size of UML notation can deter users
  - The formality that comes with UML may also deter users

SOFTWARE REQUIREMENTS

Design of Software Systems

Requirements Engineering (Definition)

Requirements Engineering

- Iterative, systematic approach to specifying requirements [in cooperation with all stakeholders]
- Explicit and agreed requirements and specifications as a
  - Basis for the development activities
- Note: Quite often contains project management activities, e.g., planning, estimation, risk analysis, etc.

Requirements Engineering (Definition)

A Requirement…

1. Is a **condition/capability/characteristic** that a stakeholder demands for a product (software) or a process **to adequately solve a problem, or to reach a certain goal**

2. Is a **condition/capability/characteristic** that a system must realize **to comply with standards, contracts, or specifications**

3. Is **a documented representation of a condition/capability/ characteristic** as defined in 1. or 2.

Requirements Engineering (Definition)

- Classification of requirements:
  - Functional:
    - influence (usage) behavior
    - degree to which a software product provides functions that meet stated and implied needs [ISO 25010 Quality standard]
  - Non functional:
    - Often referred to as quality requirements
    - Address characteristics/properties
    - Define the operation of the software
    - E.g., performance, usability, security, etc.

Requirements Engineering (Definition)

- Requirements generally come from:
  - Users
  - Regulators or legal entities
  - Developers, testers, product/project owners

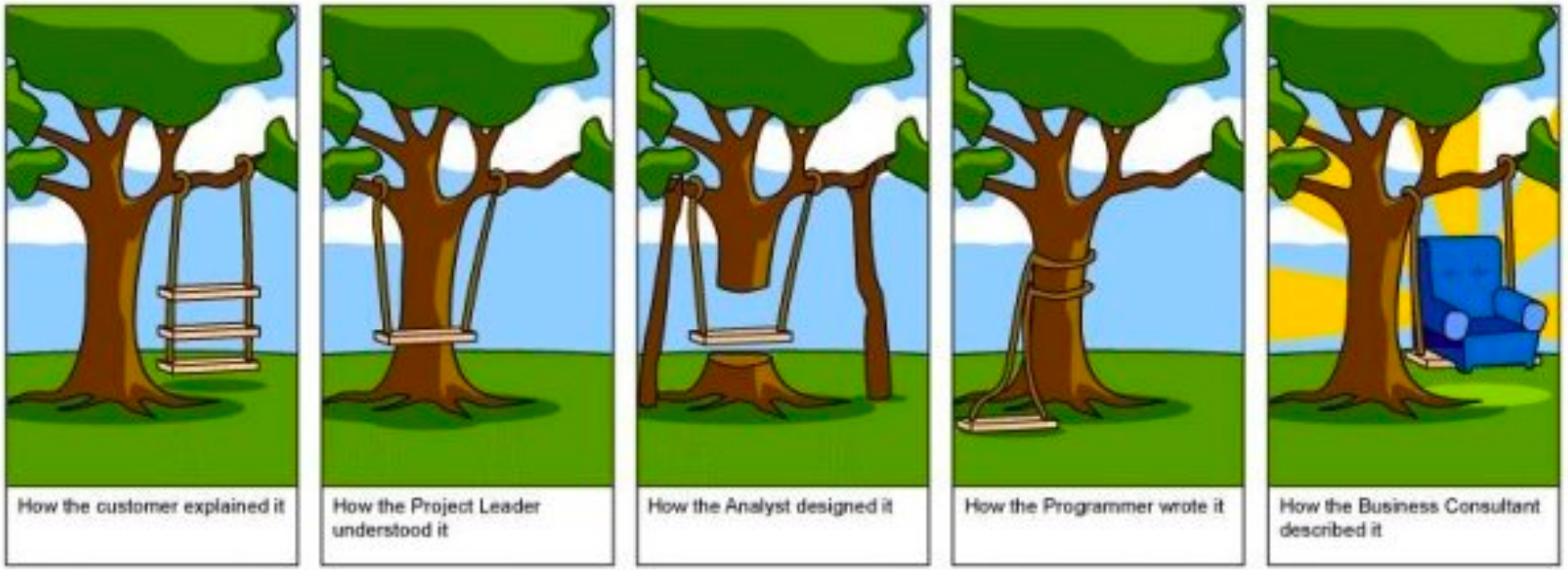
  - In essence they are all stakeholders (which is defined as: individuals/groups with a direct or indirect influence on the requirements)

Requirements Engineering (Terminology)

Typical activities in Requirements Engineering

Elicitation	• Gather requirements
Specification	• Documentation of requirements • Requirements modeling • Structuring requirements
Analysis	• Negotiation of requirements • Prioritization of requirements • Verify and validate
Management	• Managing changes • Tracking changes

“You built what I asked for but … it’s not what I want.”



How the customer explained it

How the project leader understood it

How the Analyst designed it

How the Programmer wrote it

How the Business Consultant described it

Requirements Approaches and Documentation

At best, the customer gets exactly what was written down.

But:

- Words are imprecise
- Words have multiple meanings
- Time-consuming
- Tedious to read

USER STORIES

Design of Software Systems

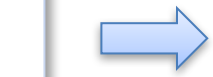
Introducing User Stories

Definition:
"A user story is a brief statement of intent that describes something the system needs to do for the user"

- Business rules
- Terms & Definitions
- Security requirements
- Error checking
- Validation

User stories template

As a <*role*>,
I can <*activity*>
so that <*business value*>.



As a <*student*>,
I can <*find my grades online*>
so that < *I don't have to wait until next day to know if I passed*>.

<role>

Who is going to perform an action or receiving value from this action

<activity>

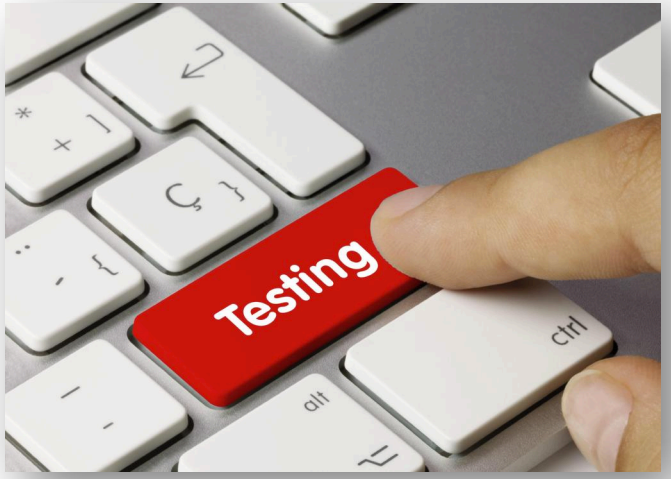
What is going to be performed by the system - which action.

<business value>

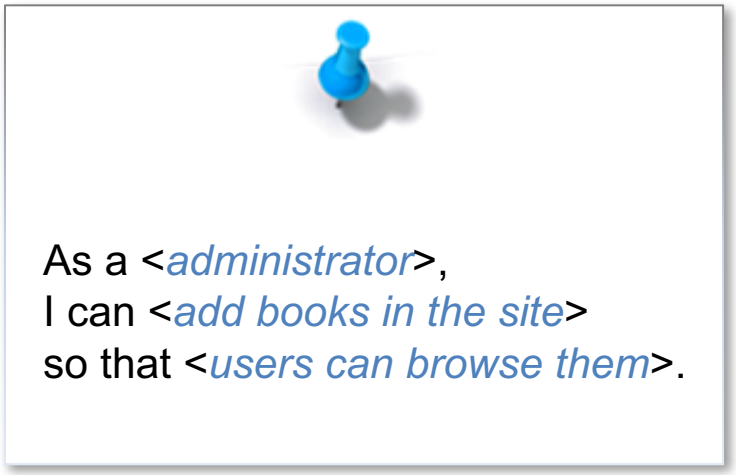
Why the action is needed.

User Stories and Acceptance Testing

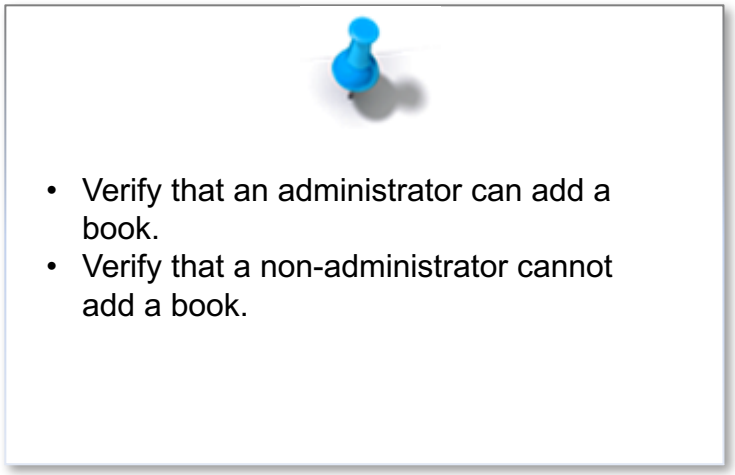
- In the back of the index card
- Written by the customer
- Verified by the customer
- Developers can write test cases



Test cases – An example



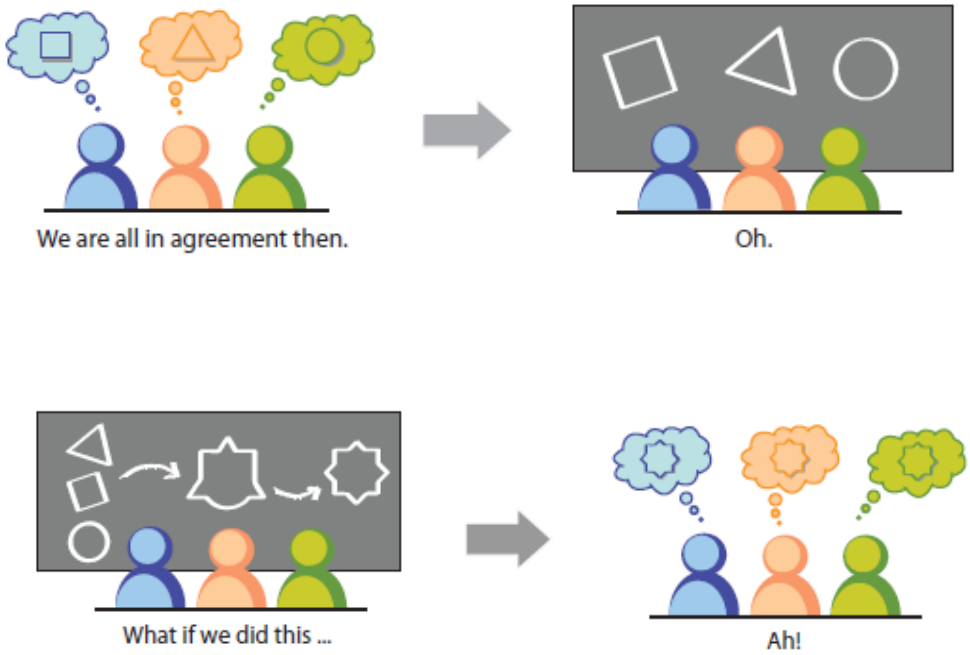
Front



...and in the back

Gathering User Stories - Techniques

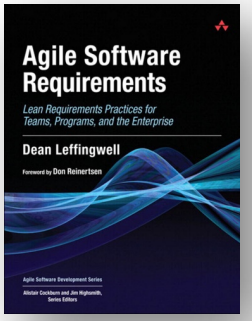
- User interviews
  - open ended
  - context free
- Observations
- Workshops
- Questionnaires

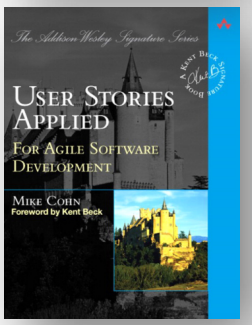


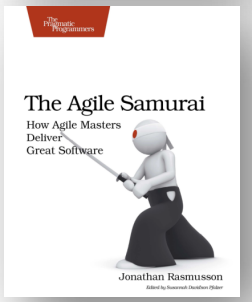
User stories drawbacks

- Lack of context
- No sense of completeness
- No sense of upcoming work
- Team can be very large

Use also Use Cases when the team is too large and the project is too complex!





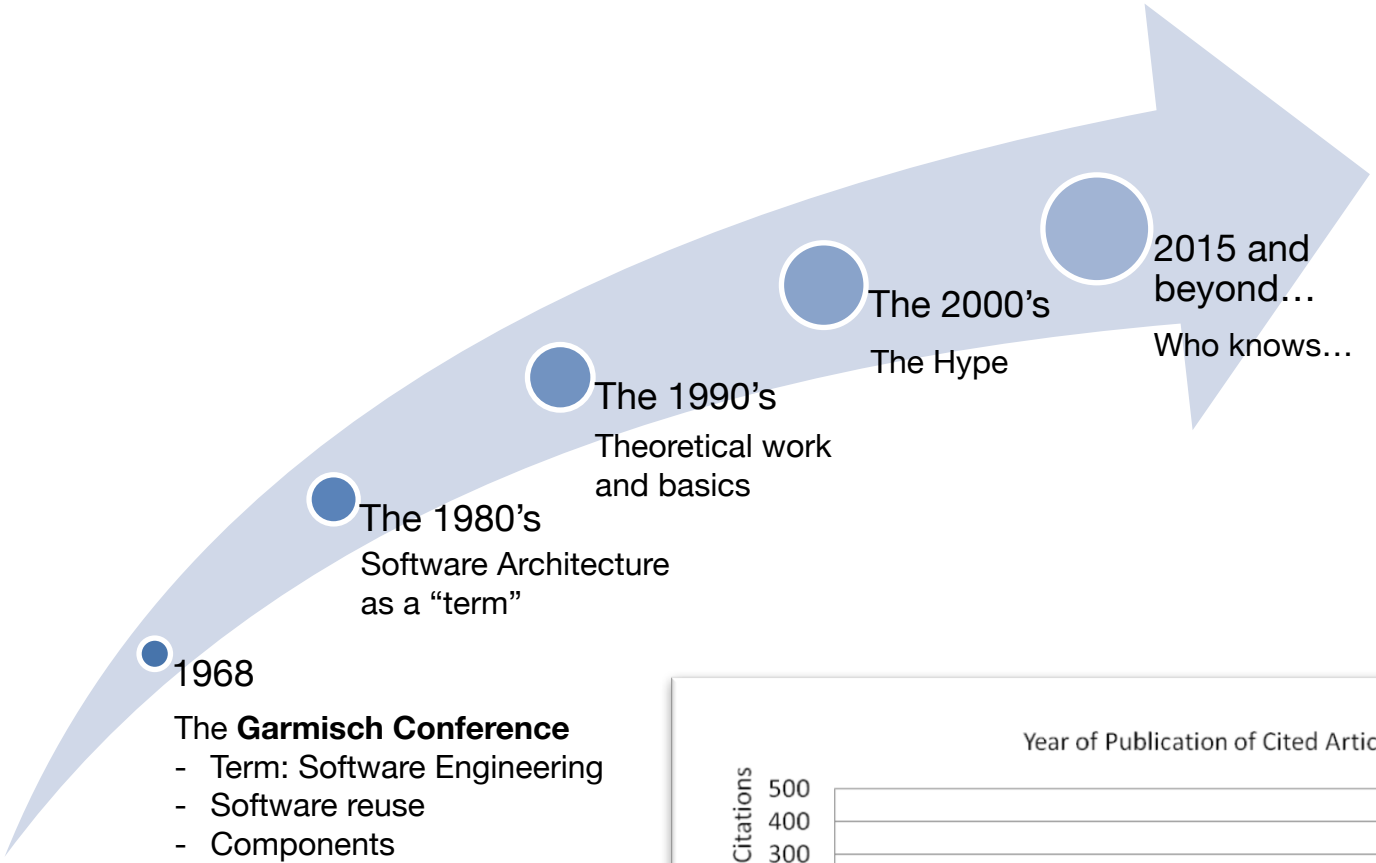


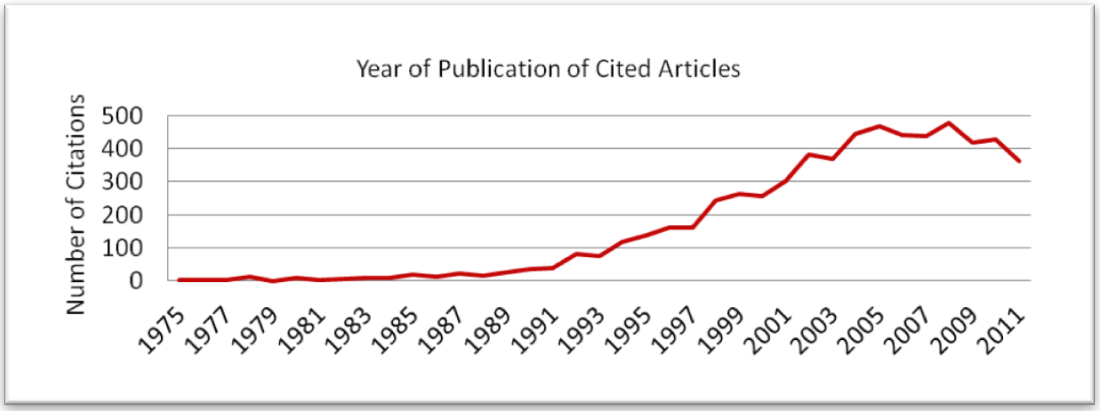
Note: this was just a short introduction. Have a look at the sources referred if you want to learn more…

SOFTWARE ARCHITECTURE

Design of Software Systems

Software Architecture





What is software architecture?

There are ~100 competing definitions of the term “Software Architecture” (cf. http://www.sei.cmu.edu)

Some examples (part 1):

- “Architecture is defined [...] as the **fundamental organization** of a system, embodied in its **components**, their **relationships** to each other and to the environment, and the **principles governing its design and evolution**.” (IEEE Std. 1471:2000)

- “The **structure** of the **components** of a program/system their **interrelationships**, and principles and **guidelines governing their design and evolution** over time.” (Garlan and Perry, 1995)

What is software architecture? – That's the problem…

Some examples (part 2):

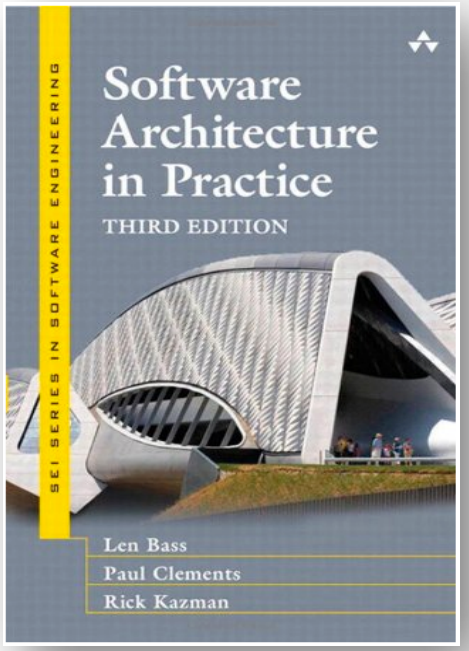
- "…the **highest level concept** of a system in its environment. The architecture of a software system is its **organization or structure** of significant **components** interacting through **interfaces**, those components being **composed** of successively smaller components and interfaces." (Rational Unified Process)

- "In most successful software projects, the expert developers working on that projects have a **shared understanding of the system design**. This shared understanding is called „architecture". This understanding includes how the system is divided into **components** and how the components interact through **interfaces**…" (Johnson, 2003)

- "Architecture is a **framework for change**." (De Marco)

What is Software Architecture? Lets adopt this definition for the course…

*The software architecture of a system is the **set of structures needed to reason about the system**, which comprise software elements, relations among them, and properties of both.*



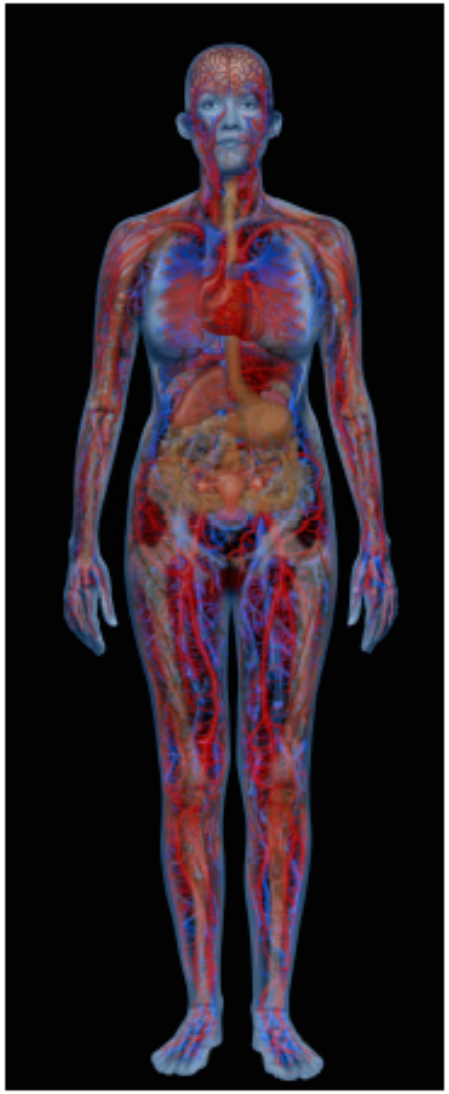
Which Structures are Architectural?

- A structure is architectural if it supports reasoning about the system and the system's properties.
- The reasoning should be about an attribute of the system that is important to some stakeholder.
- These include
  - functionality achieved by the system
  - the availability of the system in the face of faults
  - the difficulty of making specific changes to the system
  - the responsiveness of the system to user requests,
  - many others.

Physiological Structures

- The neurologist, the orthopedist, the hematologist, and the dermatologist all have different views of the structure of a human body.
- Together they describe the architecture of the human body.
- So it is with software!

Physiological Structures



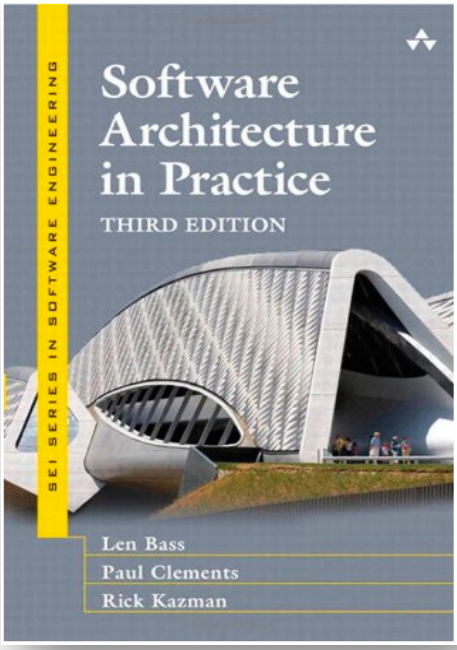
Architectural Structures and Views

- Structure and Views will be used to when discussing software architecture representations

- **View**: “ is a representation of a coherent set of architectural elements, as written by and read by system stakeholders. It consists of a representation of a set of elements and the relations among them.”

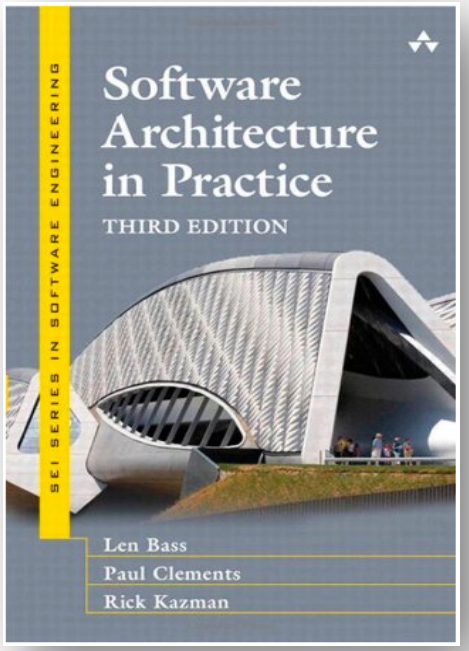
- **Structure**: “is the set of elements itself, as they exist in software...”

- **“Architects design structures. They document views of those structures.”**



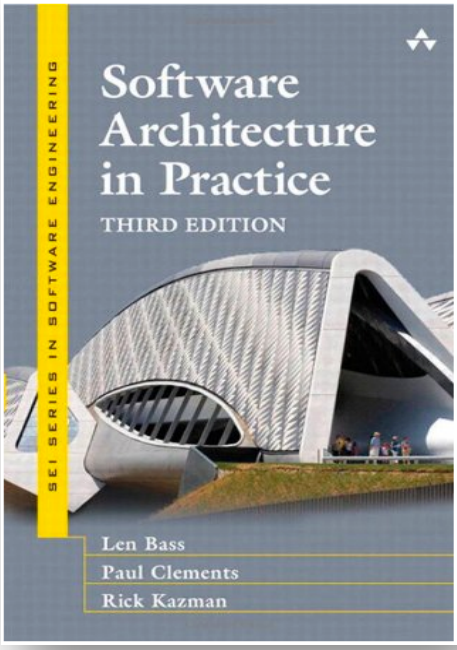
Architecture Is a Set of Software Structures

- Bass et al. define 3 categories of architectural structures.
  1. Module
  2. Component and Connector
  3. Allocation



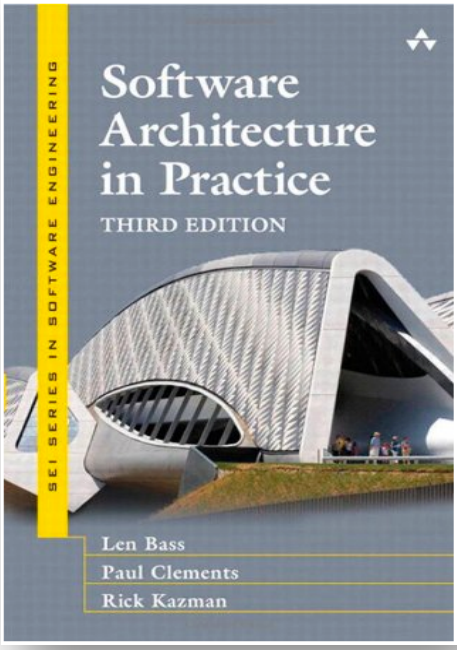
Kinds of Architectural Structures

- Module structure: “embody decisions as to how the system is to be structured as a set of code or data units...”
- Component and connector structure: “embody decisions as to how the system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors).”
- Allocation structures: “show the relationship between the software elements and elements in one or more external environments in which the software is created and executed.”



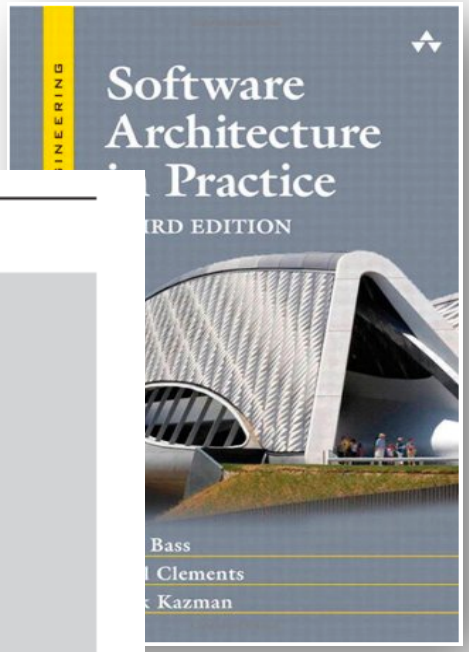
Kinds of Architectural Structures

- Each view of a structure gives a different perspective of the system
- Lets see a very simple example as an illustration



Kinds of Architectural Structures: Simple Example

Module decomposition	Component-and-connector



WHY?

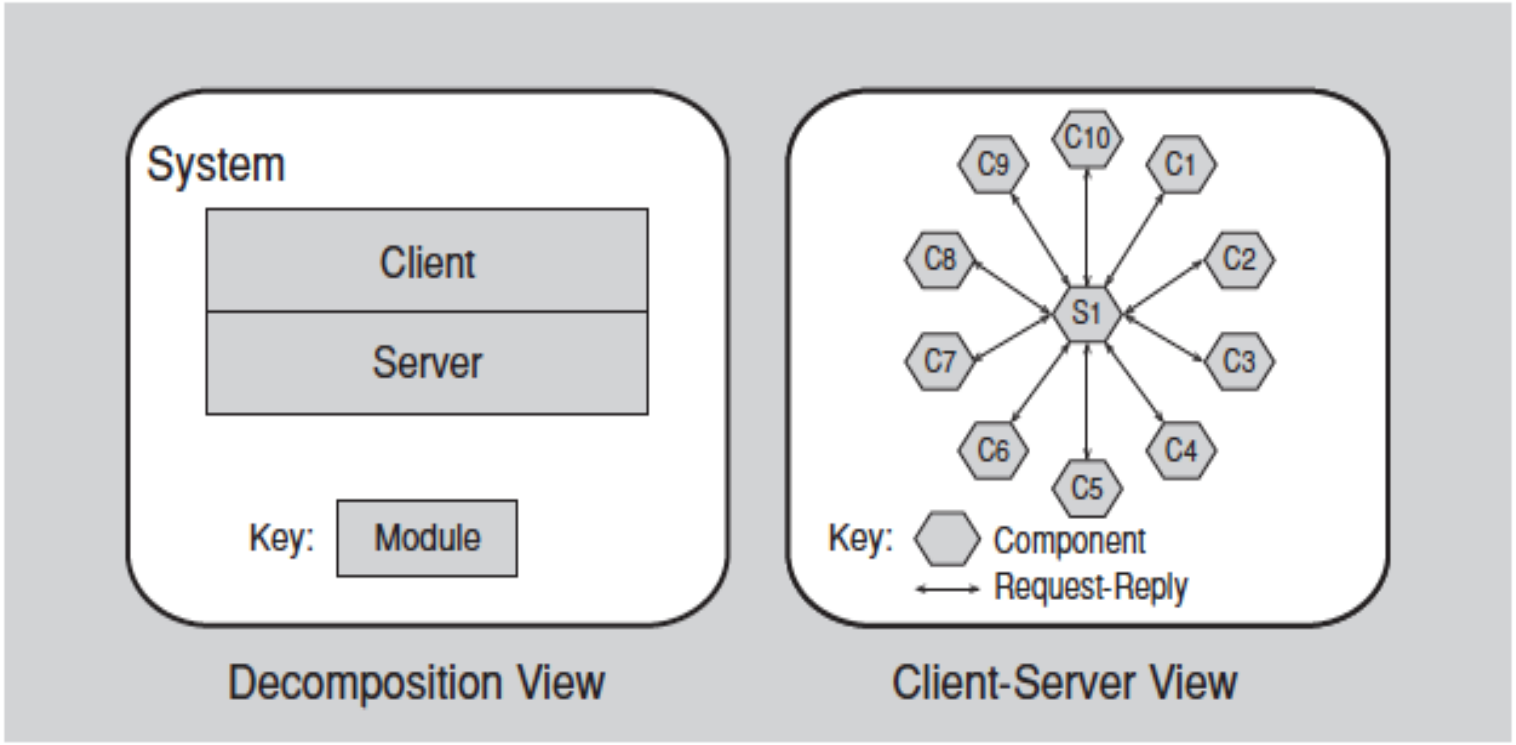


FIGURE 1.2 Two views of a client-server system

What Makes a “Good” Architecture?

- There is no such thing as an inherently good or bad architecture.
- Architectures are either more or less fit for some purpose
- Architectures can be evaluated but only in the context of specific stated goals.

- There are, however, good rules of thumb.

Process “Rules of Thumb”

- The architecture should be documented using views. The views should address the concerns of the most important stakeholders in support of the project timeline.

- The architecture should be evaluated for its ability to deliver the system’s important quality attributes.
  - This should occur early in the life cycle and repeated as appropriate.

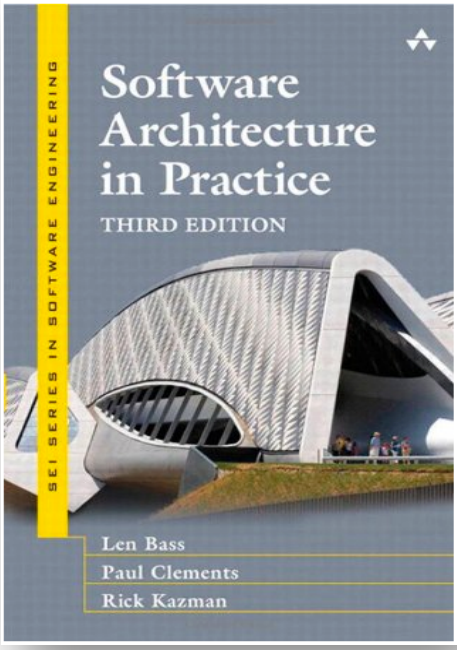
Structural “Rules of Thumb”

- Unless your requirements are unprecedented your quality attributes should be achieved using well-known architectural patterns and tactics specific to each attribute.

- Modules that produce data should be separate from modules that consume data.
  - This tends to increase modifiability
  - Changes are frequently confined to either the production or the consumption side of data.

Structural “Rules of Thumb”

- More in the this book ….



Why is software architecture that difficult?

- A particular architecture depends on the context
- One problem may have different solutions, as long as:
  - The solution architecture is appropriate
  - The solution architecture is reasonable
- Note: architectures are **not** “good” or “bad” per se…

- Example: “*Provide housing for a small group of people for protection against hostile environments.*”

Solution 1: Igloo

Solution 2: Castle

Solution 3: Space Station

Why is software architecture that difficult?

Administrators:
- Simple deployment
- Easy to maintain
- …

User:
- Function
- Performance
- Stability
- Safety
- Security
- …

Client:
- Fair price
- High quality
- In time

Companies:
- Service levels
- Regulation/laws
- Policies
- …

Software Architect

Developer:
- Implementation strategy
- Coding guides
- Frameworks, patter, reuse
- …

Management:
- Efficiency
- Effectiveness
- (eco.) growth
- …

Marketing/Sales:
- More features
- Better price
- Better quality

→ than competitors… ☺

The **Software Architect**

… has many faces (just a selection)…

- As a developer:
  - Evaluation of alternatives → prototypes
  - Communication in the team
  - Introduction of “new way”, and innovative technologies and concepts
  - Trainer and coach of the team
  - Coordinates the definition of interfaces

- As an analyst:
  - Contributes to the system vision by: defining goals, requirements…
  - Evaluation of requirements, goal achievements using, e.g., reviews, prototypes, etc.
  - Integration of technical and conceptual innovations into the shared system vision
  - Communication to foster acceptance among all stakeholders

- As a (technical) project manager:
  - Can make decisions regarding designs and changes
  - Participates in the team staffing
  - Participates in the selection and instantiation of training measures, technologies, and tools
  - Motivator of the team
  - Organization of the team according the the project plan (e.g., task distribution/delegation)
  - Supports project- and quality management (e.g., progress tracking, quality assurance, etc.)
  - Support the follow-up activities regarding system operation, evolution, and maintenance

Definition: Software Architecture

"A Software Architecture defines a set of components and their relationships."

Software architecture:

- Structures a software system (or a view on a software system) into its components with defined responsibilities and interactions
  - Refinement of a system into sub-systems: components
  - Composition of sub-systems into an integrated system
- Supports the evaluation of the overall system quality by evaluating the quality of its parts
- Is the basis for a shared understanding of the system
- Defines interfaces for future extensions/enhancements
- Affects significantly the development- and maintenance processes

Definition: Software Architecture

A more pragmatic point of view:

Software architecture as a discipline is a smart combination of proven concepts, best practices, and tools of a computer scientist/engineer:

- **Abstraction**: emphasize the key problem and abstract from details
  → leads to the concept: Interface
- **Divide and Conquer**: break down a complex problem to many smaller, less complex but better manageable (sub-)problems
  → leads to the concept: Component
- **Reuse**: awareness regarding a component's embodiment and its later use dramatically increases reusability
  → don't re-invent the wheel (use and improve stuff that's already there, don't make mistakes twice…)

The architecture design process

Overall goals:

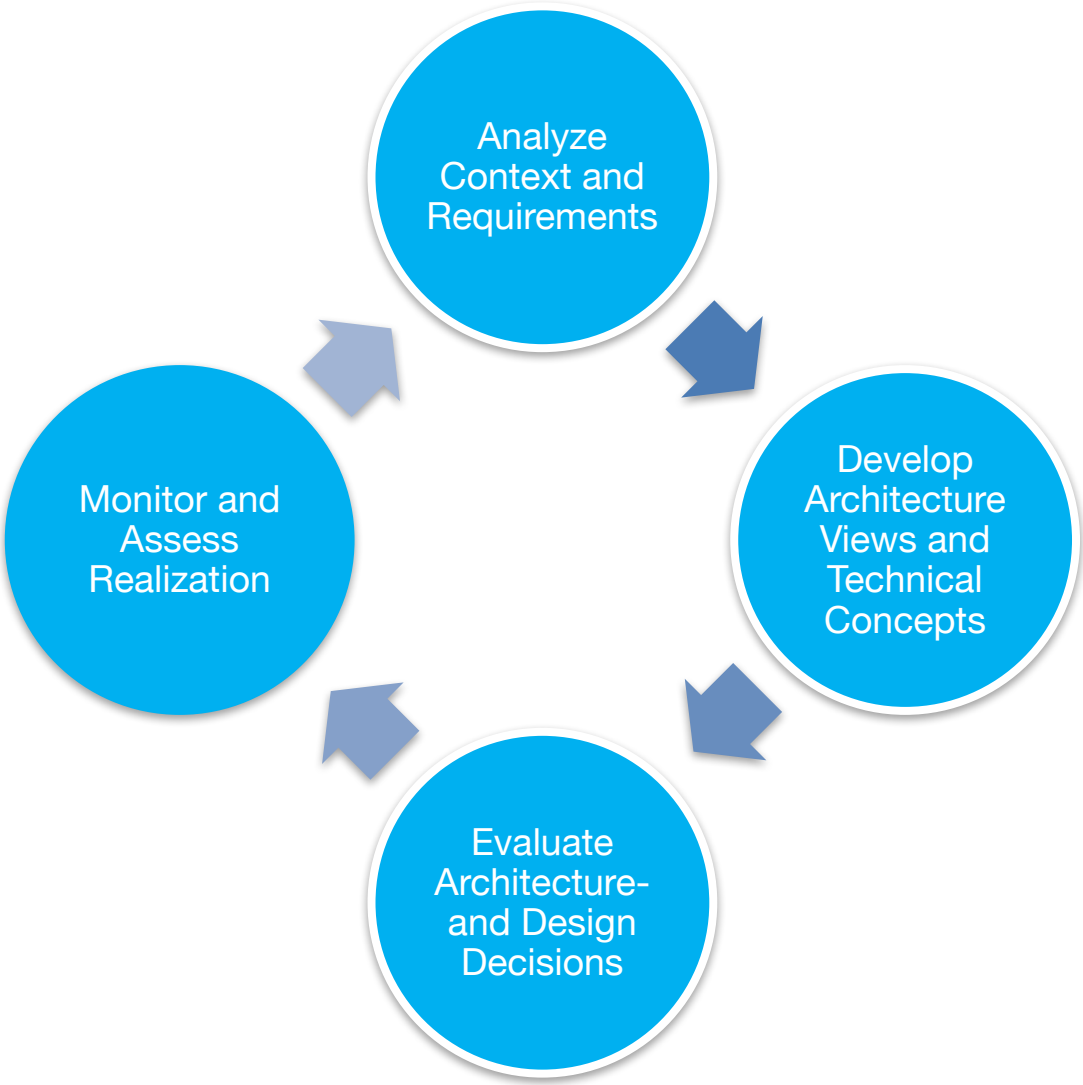
- Develop the architecture (the blueprint) for the system as a basis for development and maintenance
- Develop a complete and concise architecture description
- Ensure that all (non-)functional requirements are addressed

The design process is a **process** → inputs and outputs

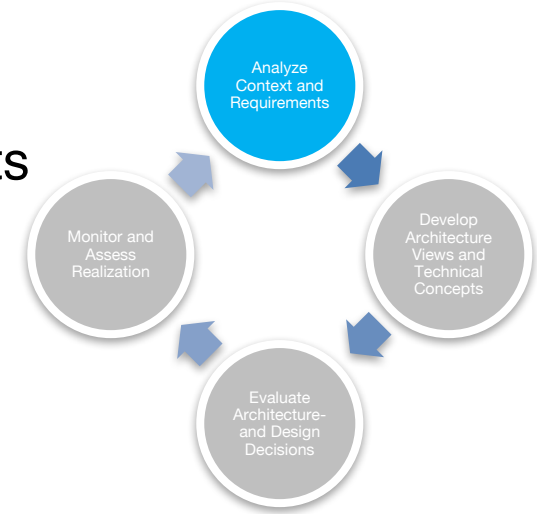
- Inputs:
  - Requirements (user, organizational, non-functional, …)
  - Existing systems (for reuse, for embedding, …)
- Outputs:
  - Software architecture (with its descriptions)

Note: That's a simplified perspective – more details will follow…

Steps in the architecture design process (overview)



Step: Analyze Context and Requirements



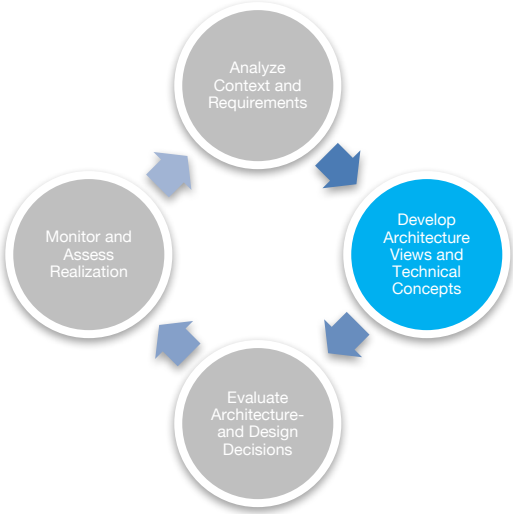
Key task:

- Analyze functional and nonfunctional requirements
- Analyze the context

And, while doing so:

- Evaluate quality and stability of the requirements
- Identify gaps
- Improve/refine the requirements, e.g., non-functional requirements, which are often considered naturally
- In cooperation with the stakeholders: develop an (initial) understanding of the architecture style and technical infrastructure
  ➔ essential, as this serves as *general* architecture metaphor, e.g., a layered architecture with…

Step: Develop Architecture Views and Technical Concepts



Key task:

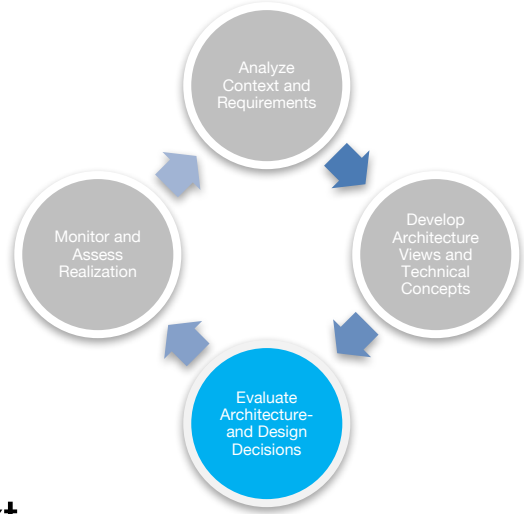
- Refinement of the architecture

And, while doing so:

- View-based description of the different architecture perspectives, e.g., logical and technical perspectives
- Break down functional requirements into corresponding logical architecture
- Break down non-functional requirements into corresponding cross-cutting architecture components for the technical architecture

Note: all this needs to fit into the previously agreed architecture style…

Step: Evaluate Architecture- and Design Decisions



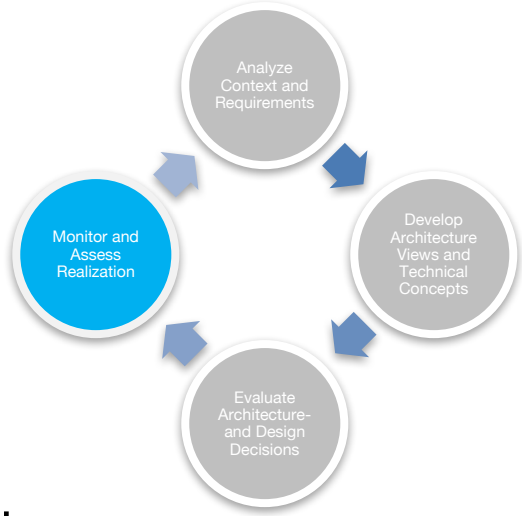
Key task:

- Quality assurance
  (*note:* this is a preview)

And, while doing so:

- Architecture needs to be evaluated against the requirements, technical environment, other context…
- Apply different techniques, e.g.:
  - Reviews
  - Prototyping and test
  - Further analytic and/or assessment methods
- Important: evaluation/assessment requires a reference → define use cases/scenarios, or in general, testable requirements, and analyze whether they are adequately represented in the architecture…

Step: Monitor and Assess Realization



Key task:

- Communicate the architecture
- Evaluate the concrete realization

And, while doing so:

- Interact with all stakeholders: Do they understand the architecture? Is the architecture accepted? Is the architecture realized correctly? Does it deliver the expected tradeoff?
- This is a very important task, as it:
  - Helps all stakeholders to reflect on the architecture
  - Establishes feedback loops allowing for continuous learning
  - Helps to identify gaps, improvement potential, errors, misunderstandings, and so forth

PROGRESS CONTROL

Design of Software Systems

The course so far

- Today:
  - Introduction: Requirements and user stories
  - Introduction: Software architecture
  - Lab: Assignment 1 on UML

- Next class and Lab:
  - Lecture:
    - Guest lecture on Software Architecture patterns and styles
  - Lab: Reflect on Assignment 1