Software architecture: Introduction

Alexander Serebrenik
This week sources

- www.iso-architecture.org/ieee-1471/index.html
- Welcome to the ISO/IEC/IEEE 42010 Website
  This is the website for ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description, the latest edition of the original IEEE Std 1471:2000, Recommended Practice for Architectural Description of Software-intensive Systems.

- Slides by Johan Lukkien and Rudolf Mak
Software architecture

• Software architecture is the fundamental organization of a system embodied in
  • its elements,
  • relationships,
  • and in the principles of its design and evolution.

[IEEE Std. 42010-2011]
“Fundamental organization”

• Less change-prone:
  • “Architecture of software is a collection of design decisions that are expensive to change.”
    Alexander Ran, Nokia Research
    European Conf on Software Engineering, 2001

• Not all aspects of system organization are fundamental
  • Hence, not all design decisions impact a system’s architecture

• How one defines “fundamental” depends on the stakeholders’ perspectives on the system goals
Stakeholders?

- **Stakeholder** is a person, group or organization with an interest in a project
  - we focus on the *roles* of stakeholders not their identities.
  - the same person can hold multiple roles.
Stakeholders?

- **Stakeholder** is a person, group or organization with an interest in a project
  - we focus on the **roles** of stakeholders not their identities.
  - the same person can hold multiple roles.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description [Rozanski Woods]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquirers</td>
<td>Oversee the procurement of the system</td>
</tr>
<tr>
<td>Assessors</td>
<td>Oversee conformance to standards and legal regulations</td>
</tr>
<tr>
<td>Communicators</td>
<td>Explain the system via documentation and training</td>
</tr>
<tr>
<td>Developers</td>
<td>Construct and deploy the system from its specifications</td>
</tr>
<tr>
<td>Maintainers</td>
<td>Manage the evolution of the system once operational</td>
</tr>
<tr>
<td>Suppliers</td>
<td>Provide hardware/software platform on which the system will run</td>
</tr>
<tr>
<td>Support staff</td>
<td>Assist users to make use of the running system</td>
</tr>
<tr>
<td>System administrators</td>
<td>Run the system once deployed</td>
</tr>
<tr>
<td>Testers</td>
<td>Check whether the system meets its specifications</td>
</tr>
<tr>
<td>Users</td>
<td>Define the system’s functionality and use it once running</td>
</tr>
</tbody>
</table>
Company A, a manufacturer of computer hardware, wants an enterprise resource planning (ERP) system to better manage all aspects of its supply chain from ordering through delivery. Management anticipates the system being constructed from a custom off-the-shelf (COTS) package combined with some in-house development for specialized aspects of functionality. The new system must be deployed within one year, in time for a meeting where shareholders will consider future funding for the company.

- Identify the stakeholders in this story.
• **Acquirers**: the business sponsor (senior management), responsible for funding; the purchasing department and representatives of IT, who will evaluate a number of potential ERP packages.

• **Users**: internal staff, including those who work in order entry, purchasing, finance, manufacturing, and distribution.

• **Developers, system administrators, and maintainers**: the internal IT department

• **Assessors**: internal acceptance test team.

• **Communicators**: internal trainers.

• **Support**: an internal help desk (possibly in conjunction with the COTS supplier).
Stakeholders, goals and concerns

• Different stakeholders have different concerns
• Different concerns imply different viewpoints
Stakeholders, goals and concerns

- Different **stakeholders** have different **concerns**
- Different **concerns** imply different **viewpoints**
  - **Viewpoint** a way of looking at a system from the position of a certain **stakeholder** with a particular **concern**
    - defines creation, depiction and analysis of a view
    - language, used models, notation, methods, analysis techniques
• Different **stakeholders** have different **concerns**
• Different **concerns** imply different **viewpoints**
  • **Viewpoint**: a way of looking at a system from the position of a certain **stakeholder** with a particular **concern**
    – defines creation, depiction and analysis of a view
    – language, used models, notation, methods, analysis techniques
• **View**: whatever you see in a system from a particular viewpoint
  – collection of system models
  – conforming to the viewpoint
Stakeholders, goals and concerns

- Different **stakeholders** have different **concerns**
- Different **concerns** imply different **viewpoints**
  - **Viewpoint**: a way of looking at a system from the position of a certain **stakeholder** with a particular **concern**
    - defines creation, depiction and analysis of a view
    - language, used models, notation, methods, analysis techniques
  - **View**: whatever you see in a system from a particular viewpoint
    - collection of system models
    - conforming to the viewpoint
  - **Model**:
    - leaves out details irrelevant to concerns
    - preserves the properties of interest with respect to those concerns
Medical example

• **System:** human heart
• **Stakeholder:** cardiologist [*class:* Tester/Maintainer]
• **Concerns:** identify and treat disorders of the heart
• **Viewpoint:**
  - used models: electrocardiography (ECG), echocardiography, ballistocardiography
  - analysis techniques: ECG interpretation, ...
• **View:**

Model kinds

Models
**Viewpoint vs View**

- **Viewpoint** should govern the **View**
  - Relation similar to type-instance, grammar/language, …
  - Viewpoints are generic, and can be stored in libraries for re-use.
  - A view is always specific to the architecture for which it is created.

*Schema by Johan Lukkien*
So far: stakeholders, concerns, views, viewpoints, models and their kinds
Library viewpoints

• “Viewpoints are generic, and can be stored in libraries for re-use”

• What are the popular library viewpoints?
  – IEEE Standards to not indicate specific viewpoints
  – Kruchten’s 4+1 [1995]
  – Rozanski, Woods [2005]
Kruchten’s 4+1

• **1995**: Kruchten, Philippe. *Architectural Blueprints — The “4+1” View Model of Software Architecture.*
  • popularized the “multiple views” idea

• **2000**: IEEE Standard 1471
  • formal conceptual model for architectural descriptions
  • standard terminology
  • distinguish between *views* and *viewpoints*

• Be careful: Kruchten’s “views” are viewpoints
  • The word “view” is used for historical reasons
A number of standard (library) views

- **End Users**
  - Functionality
  - Logical View

- **Programmers**
  - Software management
  - Development View

- **Scenarios**
  - Process View
  - Deployment View

- **System Integrators**
  - Performance, scalability

4+1 View Model

[Kruchten 95]
Logical view

• The **user**’s view on the system, i.e. what a user encounters while using the system
  • **classes** and objects (class instances) documenting user-visible entities
    – including *interfaces*, that imply *responsibilities*
  • **interaction diagrams**: interactions describing usage scenario’s
  • **state diagrams**: describing state changes as result of interactions

• Typical relations: is-a, has-a, …
Twitter example: Logical view

A domain model
• class diagram
• captures concepts from the application domain and their relationships

Additional models will typically be used to represent the logical view.
Development view

- **Components**, functions, **subsystems** organized in modules and **packages**
  - Component/module interface descriptions, access protocols
- Logical organization – **layering** of functionality, **dependencies**
  - Don’t misunderstand the name ‘logical’
- Organization into files and folders

- Typical relations: uses, contains, shares, part-of, depends-on
Twitter example: Development view

With thanks to Rudolf Mak
Process view

- Concurrent **activities** inside the system
- Mapping of applications to distinct memory spaces and units of execution
  - unit of execution: process, thread
  - memory space: associated with a process
- Choice of communication protocols
- Scheduling of activities such as to satisfy **time** and resource constraints
  - Performance
Twitter example: process view

http://www.slideshare.net/raffikrikorian/qcon-nyc-2012-twitters-real-time-architecture
Deployment view

- **Machines** (processors, memories), networks, connections
  - including specifications, e.g. speeds, sizes
- **Deployment**: mapping of elements of other views to machines

- Typical relations: connects-to, contains, maps-to

- Concerns: performance (throughput, latency), availability, reliability, etc., together with the process view
Twitter example: Deployment view

- Not very convincing
- No physical components
- It is hard to find a twitter deployment model

With thanks to Rudolf Mak
Scenarios

• Set of **use cases**
  • Small set
  • Representative use cases
Twitter scenario example: insertion of a tweet

1. User:
   a) Logon to GUI
   b) Send tweet

2. Process the insertion request

3. Put the request in the queue system

4. Back-end
   a) Get request from front of queue
   b) Insert tweet into memory.
   c) Update the search engine with an index entry

5. Send feedback to queuing system

6. Notify GUI
A number of standard (library) views

- Class Diagram
- Object Diagrams
- State machine
- Activity Diagram
- Sequence Diagram
- Timing Diagram

Use Cases

- Logical View
  - end users
  - functionality
- Development View
  - programmers
  - software management
- Use Cases
  - Scenarios
  - End users
  - Functionality
- Development View
  - Programmers
  - Software management
- Deployment View
  - System integrators
  - Performance, scalability
  - System engineers
  - Topology, communication

4+1 View Model

[Kruchten 95]
Taylor, Medvidovic, Dashofy: Alternative classification of viewpoints

- **Logical**: capture the logical (often software) entities in a system and how they are interconnected.
- **Physical**: capture the physical (often hardware) entities in a system and how they are interconnected.
- **Deployment**: Capture how logical entities are mapped onto physical entities.
- **Concurrency**: Capture how concurrency and threading will be managed in a system.
- **Behavioral**: Capture the expected behavior of (parts of) a system.
How would we map different classifications?

Kruchten’s “views” (viewpoints)

A number of standard (library) views

- **Logical View**
  - end users functionality
  - users functionality

- **Development View**
  - programmers software management
  - system integrators performance, scalability

- **Deployment View**
  - system engineers topology, communication

- **Process View**
  - scenarios

4+1 View Model [Kruchten 95]

Link Kruchten’s views and viewpoints of Taylor, Medvidovic Dashofy to each other

Taylor, Medvidovic, Dashofy: Alternative classification ov viewpoints

- **Logical**: capture the logical (often software) entities in a system and how they are interconnected.
- **Physical**: capture the physical (often hardware) entities in a system and how they are interconnected.
- **Deployment**: Capture how logical entities are mapped onto physical entities.
- **Concurrency**: Capture how concurrency and threading will be managed in a system.
- **Behavioral**: Capture the expected behavior of (parts of) a system.
How would we map different classifications?

Kruchten’s “views” (viewpoints)

A number of standard (library) views

- Logical View: end users functionality
- Development View: programmers software management
- Process View: system integrators performance, scalability
- Deployment View: system engineers topology, communication
- Scenarios

4+1 View Model [Kruchten 95]

Taylor, Medvidovic, Dashofy: Alternative classification of viewpoints

- **Logical**: capture the logical (often software) entities in a system and how they are interconnected.
- **Physical**: capture the physical (often hardware) entities in a system and how they are interconnected.
- **Deployment**: Capture how logical entities are mapped onto physical entities.
- **Concurrency**: Capture how concurrency and threading will be managed in a system.
- **Behavioral**: Capture the expected behavior of (parts of) a system.
Rozanski Woods: Alternative classification

- **Viewpoint library**
  - Functional viewpoint
  - Information viewpoint
  - Concurrency viewpoint
  - Development viewpoint
  - Deployment viewpoint
  - Operational viewpoint

Not explicitly addressed by Kruchten’s 4+1
Rozanski Woods: Alternative classification

- **Viewpoint library**
  - Functional viewpoint
  - Information viewpoint
  - Concurrency viewpoint
  - Development viewpoint
  - Deployment viewpoint
  - Operational viewpoint

- **Perspectives** (a.k.a. non-functional requirements)
  - Security
  - Performance and scalability
  - Availability and resilience
  - Evolution
  - and many more …

Not explicitly addressed by Kruchten’s 4+1

Orthogonal to viewpoints/views

Slide inspired by Rudolf Mak’s slides
Recall: Kruchten popularized “multiple views” idea

Architectural description aggregates multiple viewpoints and views.
Recall: Kruchten popularized “multiple views” idea

Architectural description aggregates multiple viewpoints and views.

What risks associated with multiplicity of views can you identify?
Risks due to multiplicity of views

Wrong views

Fragmentation

Inconsistency
Wrong views

• Which views are suitable?

• Kruchten, Rozanski-Woods provide guidance but **specifics of the system** should be taken into account
  • Is concurrency important? Is security important?

• Matter of experience and skill
Fragmentation

- Kruchten: 4+1, Rozanski-Woods: 6
- What about 50 views?

- Each view has to be created and maintained ⇒ costs!
- Views have to be kept consistent, the more views the more potential inconsistencies ⇒ costs!

- Consider combining less crucial views
  - e.g. deployment & concurrency
  - do not overdo, combined views are more difficult to understand
Inconsistency

• All views are related
  • They describe the same system!
Inconsistency

- All views are related
  - They describe the same system!

- Recall: *interaction diagrams* should be consistent with the corresponding *class diagrams* and *use case diagrams*
Inconsistency

- All views are related
  - They describe the same system!

- Recall: *interaction diagrams should be consistent with the corresponding class diagrams and use case diagrams*
Of course, there are more rules

Quote: “Package A depends on package B if A contains a class which depends on a class in B”

⇒ Development view should be consistent with the logical view
Of course, there are more rules

Quote: “Package A depends on package B if A contains a class which depends on a class in B”

⇒ Development view should be consistent with the logical view

Quote: “Manifestation maps artifacts to … / components / packages”

⇒ Deployment view should be consistent with the development view
Of course, there are more rules

Quote: “Package A depends on package B if A contains a class which depends on a class in B”

⇒ Development view should be consistent with the logical view

⇒ Deployment view should be consistent with the development view

Quote: “Manifestation maps artifacts to … / components / packages”

⇒ Deployment view should be consistent with scenarios

Quote: “Manifestation maps artifacts to use cases”
If something changes at the end of the arrow, a change will probably be required at the start of the arrow.
Still...

• We need a mechanism to record in the architecture description **relations between different views**
  • What views should be consistent with each other?
  • Does one view refine another? (package vs class?)
  • Should one be traced to another?
Still...

- We need a mechanism to record in the architecture description **relations between different views**
  - What views should be consistent with each other?
  - Does one view refine another? (package vs class?)
  - Should one be traced to another?

- Architectural description (AD) elements
  - stakeholders, views, viewpoints, models…
- Correspondences vs. Correspondence rules
  - similarly to views vs. viewpoints
Putting it all together: architecture

Rationale: explanation, justification or reasoning about architecture decisions
Rationale: closer look
Why do we distinguish between architecture and architecture description?
Different flavors of architecture

What other kind of architecture flavor would you expect?
Different flavors of architecture

Architecture as intended

Architecture as described

Architecture as implemented
Different flavors of architecture

Architecture as intended

Architecture as described

Architecture as implemented

Implementation, forward engineering

Architecture reconstruction, reverse engineering
Different flavors of architecture

- **Prescriptive architecture**
  - Architecture as intended
  - Implementation, forward engineering

- **Descriptive architecture**
  - Architecture as described
  - Architecture reconstruction, reverse engineering

- **Architecture as implemented**
Beware: incompleteness

- **Architecture as described** can/should never be complete with respect to
  - the system
    - architecture = “fundamental organization”
    - not everything is fundamental
  - architecture as intended
    - limitation of the architecture description language(s)
  - architecture as implemented
    - freedom of implementation choices

- Architecture as intended should at least be **complete** wrt the stakeholders’ concerns
Architectural Evolution

• When a system evolves
  • ideally its **prescriptive architecture** is modified first
  • in practice, the system – and thus its **descriptive architecture** – is often directly modified

• This happens because of
  • Developer sloppiness
  • Perception of short deadlines which prevent thinking through and documenting
  • Lack of documented prescriptive architecture
  • Need or desire for code optimizations
  • Inadequate techniques or tool support
Architectural Degradation

- **Architectural drift** is introduction of principal design decisions into a system’s descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which **do not violate** any of the prescriptive architecture’s design decisions
Architectural Degradation

• **Architectural drift** is introduction of principal design decisions into a system’s descriptive architecture that
  • are not included in, encompassed by, or implied by the prescriptive architecture
  • but which **do not violate** any of the prescriptive architecture’s design decisions

• **Architectural erosion** is the introduction of architectural design decisions into a system’s descriptive architecture that **violate** its prescriptive architecture
Architectural Degradation

• **Architectural drift** is introduction of principal design decisions into a system’s descriptive architecture that
  • are not included in, encompassed by, or implied by the prescriptive architecture
  • but which **do not violate** any of the prescriptive architecture’s design decisions

• **Architectural erosion** is the introduction of architectural design decisions into a system’s descriptive architecture that **violate** its prescriptive architecture

• If architectural degradation is allowed to occur,
  • one will be forced to *reconstruct* the system’s architecture sooner or later
Software architecture

- Software architecture is the **fundamental organization** of a system embodied in
  - its **elements**,
  - relationships,
  - and in the principles of its **design** and **evolution**.

  [IEEE Std. 42010-2011]

---

Stakeholders, goals and concerns

- Different **stakeholders** have different **concerns**
- Different **concerns** imply different **viewpoints**
  - **Viewpoint** a way of looking at a system from the position of a certain **stakeholder** with a particular **concern**
    - defines creation, depiction and analysis of a view
    - language, used models, notation, methods, analysis techniques
  - **View**: whatever you see in a system from a particular viewpoint
    - collection of system models
    - conforming to the viewpoint
  - **Model**:
    - leaves out details irrelevant to concerns
    - preserves the properties of interest with respect to those concerns

---

UML diagrams and Kruchten’s view(point)s?

A number of standard (library) views

- **Class Diagram**
  - end users
  - programmers
- **Interaction Diagrams**
  - Logical View
- **State machine**
- **Activity Diagram**
  - Process View
- **Sequence Diagram**
- **Timing Diagram**

**Scenarios**

**Use Cases**

- **Deployment Diagram**
- **Deployment View**
  - system integrators
  - performance, scalability

- **Development View**
  - system engineers
  - topology, communication

**Component Diagram**

**Package Diagram**

**4+1 View Model**

[Kruchten 95]

---

Putting it all together: architecture

[Diagram of architecture and corresponding viewpoints]

Why do we distinguish between architecture and architecture description?