Introduction to Software Engineering

10. Software Architecture
Andrea Caracciolo

Adapted from slides by Oscar Nierstrasz and Mircea Lungu
Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles
- UML diagrams for architectures
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> Architectural styles
> UML diagrams for architectures
Example Architecture

SYSTEM cannot contain cycles
PersistencePackage cannot depend on ServicePackage
ImplClass must have annotation "@Service"
Batch Process

Probo
ST image

Parser
reasoner
tool handler
report generator
Client/Server

Client

Server

Rest

CLI
Integration - Analysis as service

Java plug-in

PHP broker

Java

sonarqube

TeamCity

Rest

CLI
Integration - Analysis as service
Scalability

Tomcat

Rest

Rules

Probo

CLI

Probo

CLI
What is Software Architecture?

Grady Booch @Grady_Booch · Nov 14
All architecture is design, not all design is architecture; architecture is most significant design decisions

Architecture: The set of design decisions about any system (or subsystem) that keeps its implementors and maintainers from exercising needless creativity.
What is Software Architecture?

Grady Booch @Grady_Booch · Nov 14
All architecture is design, not all design is architecture; architecture is most significant design decisions

*design decisions* resulting in element properties that are *not visible* (make no difference outside the element) are *non-architectural*.
What is Software Architecture?

The architecture of a system consists of:

1. the *structure(s) of its parts*
   - e.g. design-time, test-time, and run-time software and hardware parts

2. the *externally visible properties* of those parts
   - e.g. provided services, performance, fault handling, shared resource usage

3. the *relationships and constraints* between them

— *Bass & Clements, IEEE 1471*
architectural decisions are ones that permit a system to meet its **quality attribute** and **behavioral requirements**.
Rationale: Design Decisions

- D01: Extend System B to implement interactive approval processing.
- D03: Continue to use System A database to store product-specific data.
- D04: Rollout only new marketing campaigns on new platform.
- D02: Use message-based middleware platform for real-time interfaces.
- D06: Use XML as message format.
- D07: All batch interfaces will be replaced.
- D08: Use API-based middleware for current clients.
- D05: Continue to populate data warehouse from System A database.
- D09: Create interfaces between message-based and API-based middleware.
<table>
<thead>
<tr>
<th>WHAT</th>
<th>HOW</th>
<th>WHERE</th>
<th>WHO</th>
<th>WHEN</th>
<th>MOTIVATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>FUNCTION</td>
<td>NETWORK</td>
<td>PEOPLE</td>
<td>TIME</td>
<td>END/MEANS</td>
</tr>
<tr>
<td>SCOPE (contextual)</td>
<td>Planner</td>
<td>Entity = Class of business things</td>
<td>List of things important to the business</td>
<td>List of processes the business performs</td>
<td>List of locations in which the business operates</td>
</tr>
<tr>
<td>BUSINESS MODEL (Conceptual)</td>
<td>Owner</td>
<td>Entity = Business Entity Relationship = Business</td>
<td>e.g., Semantic Model</td>
<td>e.g., Business Process Model</td>
<td>e.g., Business Logistics System</td>
</tr>
<tr>
<td>SYSTEM MODEL (Logical)</td>
<td>Designer</td>
<td>Entity = Data Entity Relationship = Data Relationship</td>
<td>e.g., Logical Data Model</td>
<td>e.g., Application Architecture</td>
<td>e.g., Distributed System Model</td>
</tr>
<tr>
<td>TECHNOLOGY MODEL (Physical)</td>
<td>Builder</td>
<td>Entity = Segment/Table Relationship = Pointer/key</td>
<td>e.g., Physical Data Model</td>
<td>e.g., System Design</td>
<td>e.g., Technology Architecture</td>
</tr>
<tr>
<td>DETAILED REPRESENTATIONS (Out-of-context)</td>
<td>Subcontractor</td>
<td>Entity = Field Relationship = Address</td>
<td>e.g., Data Definition</td>
<td>e.g., Program</td>
<td>e.g., Network Architecture</td>
</tr>
<tr>
<td>FUNCTION ENTERPRISE</td>
<td>e.g., DATA</td>
<td>e.g., FUNCTION</td>
<td>e.g., NETWORK</td>
<td>e.g., ORGANISATION</td>
<td>e.g., SCHEDULE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BUSINESS MODEL</th>
<th>e.g., Semantic Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner</td>
<td>Entity = Business Entity, Relationship = Business</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYSTEM MODEL</th>
<th>e.g., Logical Data Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Entity = Data Relationship, Relationship = Data Relationship</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TECHNOLOGY MODEL</th>
<th>e.g., Physical Data Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Builder</td>
<td>Entity = Segment/Table, Relationship = Pointer/key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DETAILED REPRESENTATIONS</th>
<th>e.g., Data Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subcontractor</td>
<td>Entity = Field, Relationship = Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Business Process</th>
<th>e.g., Business Process Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Business Process</td>
</tr>
<tr>
<td>I/O</td>
<td>Business Resource</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural Design</th>
<th>e.g., Application Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Application Function</td>
</tr>
<tr>
<td>I/O</td>
<td>User Views</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Architectural Design</th>
<th>e.g., System Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Computer Function</td>
</tr>
<tr>
<td>I/O</td>
<td>Data Elements/sets</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>e.g., Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Language Statement</td>
</tr>
<tr>
<td>I/O</td>
<td>Control Block</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e.g., Business Logistics System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node = Business Location, Link = Business Linkage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e.g., Workflow Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>People = Organisation unit, Work = Deliverable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e.g., Human Interface Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node = H/w /System s/w, Relationship = Line Specifications</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e.g., Technology Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>People = User, Work = Screen Formats</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e.g., Presentation Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node = Address, Link = Protocol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e.g., Security Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>People = Identity, Work = Job</td>
</tr>
</tbody>
</table>
Non-Architectural Design

Objects, Functions, DB tables, ..
Architectural Design

Components, Processes, ..

- performance
- security
- maintainability

IT System
Architectural design: WHO

- identify actors (human/not human)
- what kind of information do they need/produce?
Architectural design: WHO

Diagram:
- Tomcat
- Rules
- TeamCity
- SonarQube
- Jenkins
- Probo
- Probo
Architectural design: WHAT

- domain abstraction model
Architectural design: WHAT
Architectural design: HOW / WHEN

• How/When is information generated, processed and transmitted (activities and information flows)
Architectural design: HOW / WHEN

\[ T < 10 \text{min} \]

\[ T < 1 \text{min} \times \text{MLOC} \]
Architectural design: WHERE

- Where actors, sources and sinks are physically and logically located
  - tech. infrastructure
  - network topology
  - ...

[Image of network diagram]
Architectural design: WHERE

Reporting clients

- Chrome
- Jenkins
- TeamCity
- SonarQube

Analysis server

- HTTP
- HTML
- HTTP
- Json/XML

Rest

CLI
Describing Software Architecture

Variable range of complexity (structure -> rationale)
Architectural Viewpoints

_Run-time_  How are responsibilities distributed amongst run-time entities?

_Process_  How do processes communicate and synchronize?

_Dataflow_  How do data and tasks flow through the system?

_Deployment_  How are components physically distributed?

_Module_  How is the software partitioned into modules?

_Build_  What dependencies exist between modules?
> “Use a 3-tier client-server architecture: all business logic must be in the middle tier, presentation and dialogue on the client, and data services on the server; that way you can scale the application server processing independently of persistent store.”
All teams will henceforth expose their data and functionality through **service interfaces**

Teams must communicate exclusively through these **interfaces** with each other.

It doesn’t matter what technology they use.

There will be **no other form of inter-process communication** allowed: no direct linking, no direct reads of another team’s data store, no shared-memory model, no back-doors whatsoever.

**Anyone who doesn’t do this will be fired.**

Thank you; have a nice day!
Formal languages for representing and reasoning about software architecture.

Provide a **conceptual framework** and a concrete syntax for characterizing architectures.

Some are **executable**, or implemented in a general-purpose programming language.

**Wright** underlying model is CSP, focuses on connectivity of concurrent components

**Darwin** focuses on supporting distributed applications. Components are single-threaded active objects
ADL example

process implementation process1.basic

subcomponents
  A: thread t1.basic; B: thread t2.basic; C: thread t2.basic;

connections
  cn1: data port signal -> A.p1;
  cn2: data port A.p2 -> B.p1;
  cn3: data port B.p2 -> result1;
  cn4: data port A.p2 -> C.p1;
  cn5: data port C.p2 -> result2;
  cn6: data port A.p3 -> status;
  cn7: event port init -> C.reset;

flows
  f1: flow path signal->cn1->A.fs1->cn2->B.fs1->cn3->result1;
  f2: flow path signal->cn1->A.fs1->cn4->C.fs1->cn5->result2;
  f3: flow sink init->cn7->C.fs2;
  f4: flow source A.fs2->cn6->status;

end process1.basic;

system implementation Software.Basic

subcomponents
  Sampler_A : process Collect_Samples { 
    Source_Text => ("collect_samples.ads", "collect_samples.adb") ;
    Period => 50 ms ;
  } ;
end Software.Basic ;
Roadmap

> What is Software Architecture?
> **Cohesion and Coupling**
> Architectural styles
> UML diagrams for architectures
Sub-systems, Modules and Components

> A **sub-system** is a system in its own right whose operation is *independent* of the services provided by other sub-systems.

> A **module** is a system component that *provides services* to other **modules** but would not normally be considered as a separate system.

> A **component** is an *independently deliverable unit* of software that encapsulates its design and implementation and offers interfaces to the out-side, by which it may be composed with other components to form a larger whole.
Cohesion is a measure of how well the parts of a component “belong together”.

> Cohesion is weak if elements are bundled simply because they perform similar or related functions (e.g., java.lang.Math).

> Cohesion is strong if all parts are needed for the functioning of other parts (e.g. java.lang.String).

—Strong cohesion promotes maintainability and adaptability by limiting the scope of changes to small numbers of components.

There are many definitions and interpretations of cohesion. Most attempts to formally define it are inadequate!
**Coupling** is a measure of the *strength of the interconnections* between system components.

> Coupling is **tight** between components if they depend heavily on one another, (e.g., there is a lot of communication between them).

> Coupling is **loose** if there are few dependencies between components.

— Loose coupling *promotes maintainability* and adaptability since *changes in one component are less likely to affect others.*

— Loose coupling *increases the chances of reusability.*
Tight Coupling

Subsystem A  Subsystem B

Subsystem C  Subsystem D

Shared data area
Loose Coupling

Subsystem A
  A’s data

Subsystem B
  B’s data

Subsystem D
  D’s data

Subsystem C
  C’s data
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> **Architectural styles**
  — Structure
  — Shared Data
  — Communication
  — Distribution
> UML diagrams for architectures
An architectural style defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style defines a vocabulary of components and connector types, and a set of constraints on how they can be combined.

— Shaw and Garlan
Architectural Style “Catalogues”
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> **Architectural styles**
  — Structure
  — Data flow
  — Call-return
  — Event-driven
> UML diagrams for architectures
"Big Ball of Mud"

The system is organized as a single element. No modularity. No constraints.

**Example:**
- Mainframe application

**Qualities:**
- Poor Extensibility
- Poor Maintainability
Component-based

Components have well defined interfaces and communicate via connectors linking their interfaces

Example:

Modules, WebServices, ..

Qualities:

+ Separation of concerns
+ Reuse
Layered

The elements in each layer communicate only with entities that are in the layers above and below.

Example:
OSI, web-apps (MVC)

Qualities:
+ Exchangeability
+ Limited error propagation
- Performance overhead
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> **Architectural styles**
  — Structure
  — **Data flow**
  — Call-return
  — Event-driven
> **UML diagrams for architectures**
Pipes & Filters

One element reading data at one end and writing it at the other end. Pipelines do not have to be linear.

Example:

Image processing, Compilers

Qualities:

+ Flexibility by recombination
- Performance (state/data sharing)
- Error handling
Blackboard

Elements share, post, update data written on the blackboard in order to collectively work on a solution to the problem.

Example:
   Sensor network, distributed computing

Qualities:
   - Difficult to test / Lack of control
   - Semantic coupling
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> **Architectural styles**
  — Structure
  — Data flow
  — **Call-return**
  — Event-driven
> UML diagrams for architectures
Client-server

One or more clients send requests to the server, which processes them before sending them back a response.

**Example:**
- Web browser, email reader, DB-app

**Qualities:**
- Communication overhead
+ Cheap infrastructure
- Single point of failure
Service oriented

Distributed components have well defined interfaces and communicate via specific connectors linking their interfaces.

Example: REST, SOAP

Qualities:
+ Loose structural coupling
+ Technology independent
Peer to peer

There is no central server as all elements can both act as client and as server and send one another requests and response messages.

Example:
Torrent

Qualities:
+ Adaptability, Scalability
- Lack of control
Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- **Architectural styles**
  - Structure
  - Data flow
  - Call-return
  - Event-driven
- UML diagrams for architectures
Event Driven system where elements are coupled by subscriptions and receive notifications when some interesting event happens

**Example:**
Message broadcasting, GUI

**Qualities:**
- Semantic coupling
+ Loose structural coupling
Rule-based

attempts to derive execution instructions from a starting set of data and rules

Example:
   Financial system, Natural language

Qualities:
   - Difficult to test / Lack of control
   + Convenient for certain domains
Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles
- UML diagrams for architectures
Decompose system into **packages** (containing any other UML element, incl. packages)
**UML support: Deployment Diagram**

*Physical layout* of run-time components on hardware nodes.
Sources

> *Objects, Components and Frameworks with UML*, D. D'Souza, A. Wills, Addison-Wesley, 1999
What you should know!

- What is software architecture
- What is the difference between non-architectural and architectural design
- What are architectural viewpoints and architectural styles
- What are ADLs, components and connectors
- Advantages and disadvantages of classical architectural styles
Can you answer the following questions?

- What kind of architectural styles are in your project?
- What are the characteristics of a multi-tier architecture?
- How can you reduce coupling between software layers?
- How would you implement a dataflow architecture in Java?
Exercise

- Customers can use the ATM from any bank to withdraw cash from their bank account.
- Each bank has its own system to deal with accounts (checking access rights, balance, etc...)
- Each ATM keeps a list of the transactions performed, so that banks can keep track of the amount of money they owe each other.
- At the end of each day, each ATM sends a report to the banks involved in each transaction.

- Bank A customer goes to an ATM of a bank different from his/her own bank to withdraw cash. The ATM machine (locally) verifies the correspondence between customer’s card and PIN. The customer asks for cash, the ATM connect the bank system, check the availability on customer’s account, log the operation and give cash.
Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

You are free to:
  Share — copy and redistribute the material in any medium or format
  Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

  Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

  ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

http://creativecommons.org/licenses/by-sa/4.0/