Software Architecture Recovery: Importance, Challenges, and Methods

Ingrid Nunes
UFRGS, Porto Alegre, Brazil

in collaboration with
Vanius Zapalowski and Daltro Nunes
"Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled."
- Eoin Woods
What is your definition of software architecture?

**WHAT IS YOUR DEFINITION OF SOFTWARE ARCHITECTURE?**

The SEI has compiled a list of modern, classic, and bibliographic definitions of software architecture. Modern definitions are definitions from Software Architecture: A Foundation for Software Engineering (2003), with new definitions from books and conferences since then. An applied definition emphasizes the plurality of structures present in every software system. These structures, carefully chosen and designed by the architect, are the key to achieving and reasoning about the system's design goals. And those structures are the key to understanding the architecture. Therefore, they are the focus of our approach to...
"The software architecture of a program or computing 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."

- Bass et al. 2003
"The software architecture of a program or computing 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."

- Bass et al. 2003

• Modules
  • Including design-time, test-time, and run-time hardware and software parts
Software Architecture

"The software architecture of a program or computing 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."

- Bass et al. 2003

- **Modules**
  - Including design-time, test-time, and run-time hardware and software parts

- **Externally visible properties**
  - Modules with interfaces, hardware units, objects
"The software architecture of a program or computing 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."

- Bass et al. 2003
Software Architecture

- Key principles

  - **Modularization**
    - **Decomposition** of a system into groups of subsystems and components
    - Physical packaging of entities

  - **Separation of Concerns**
    - Isolation of responsibilities
    - If a component plays different roles in different contexts, these roles must be separated

There must be a way to implement architectural modules into the source code (e.g. packages)!
Software Architecture

- Key design decisions
  - **Adopted technologies**
    - spring Framework
    - Python
    - Django
  - **Non-functional requirements**
    - Scalability
      - Volume
      - Performance
      - Stress
      - Load Testing
      - Safety Awareness
      - Accountability

10/15/2019

ingridnunes@inf.ufrgs.br
If you think good architecture is expensive, try bad architecture."

- Brian Foot and Joseph Yoder
Software Architecture

- Fundamental for the organised evolution of software systems

1. Business Rule
2. Business
3. Bug: violated business rule
4. Duplicated code!
Software Architecture

• Majority of existing systems
  • Architecture documentation does not exist
  • If it exists, it is outdated
Software Architecture

Software Architecture

- Software Reflexion Models

But… many studies report a high number of architecture violations.
Why?
Understanding architecture non-conformance

Software Architecture Recovery
Research Questions

1. **What** is the gap between conceptual architectural rules and implemented module dependencies?
2. How can implemented module dependencies be **categorized** in relation to conceptual architectural rules?
3. Are implemented module dependencies **distinguishable** considering their categorization?
Procedure Overview

Source Code → Dependencies

Extract Dependency
infra.service → core.event

Source Code Dependencies

Run Association Rule to Calculate Support
infra.service → core.event (90%)

Dependencies with Support value

Analyze Rules [metrics]

Subject Systems

Software Architects

Recover System Architectures [Conceptual Rules and Modules]

Conceptual Rules [infra → core]
## Target Systems

<table>
<thead>
<tr>
<th>System</th>
<th>LOC</th>
<th>Rules</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArchStudio</td>
<td>236.9K</td>
<td>53</td>
<td>Heterogeneous</td>
</tr>
<tr>
<td>AspectJ</td>
<td>217.9K</td>
<td>31</td>
<td>Heterogeneous</td>
</tr>
<tr>
<td>EC</td>
<td>11.7K</td>
<td>19</td>
<td>Layered</td>
</tr>
<tr>
<td>Metrics</td>
<td>15.6K</td>
<td>8</td>
<td>Extended MVC</td>
</tr>
<tr>
<td>OLIS</td>
<td>11.4K</td>
<td>13</td>
<td>Layered</td>
</tr>
<tr>
<td>RecSys</td>
<td>22.8</td>
<td>19</td>
<td>Heterogeneous</td>
</tr>
</tbody>
</table>
RQ1: Conceptual Architecture vs. Dependencies

- Analysis of implemented dependencies
- Architecture conformance
RQ1: Conceptual Architecture vs. Dependencies

- Architecture conformance is low
  - Except EC and OLIS
- Consistency with previous work

<table>
<thead>
<tr>
<th>System</th>
<th>Implemented Dependencies</th>
<th>Architecture Conformance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArchStudio</td>
<td>1178</td>
<td>26.1%</td>
</tr>
<tr>
<td>AspectJ</td>
<td>683</td>
<td>28.7%</td>
</tr>
<tr>
<td>EC</td>
<td>135</td>
<td>94.1%</td>
</tr>
<tr>
<td>Metrics</td>
<td>45</td>
<td>55.6%</td>
</tr>
<tr>
<td>OLIS</td>
<td>86</td>
<td>93.0%</td>
</tr>
<tr>
<td>RecSys</td>
<td>375</td>
<td>36.0%</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td></td>
<td><strong>55.6%</strong></td>
</tr>
<tr>
<td><strong>SD</strong></td>
<td></td>
<td><strong>31.2%</strong></td>
</tr>
</tbody>
</table>
RQ1: Conceptual Architecture vs. Dependencies

- Analysis of allowed dependencies
  - Rule conformance
RQ1: Conceptual Architecture vs. Dependencies

• Most systems have very low results
  • Exception: Metrics
    • Small system
• Architectural rules are possibly too permissive

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ArchStudio</td>
<td>1178</td>
<td>1.8%</td>
</tr>
<tr>
<td>AspectJ</td>
<td>683</td>
<td>14.0%</td>
</tr>
<tr>
<td>EC</td>
<td>135</td>
<td>16.8%</td>
</tr>
<tr>
<td>Metrics</td>
<td>45</td>
<td>39.7%</td>
</tr>
<tr>
<td>OLIS</td>
<td>86</td>
<td>11.4%</td>
</tr>
<tr>
<td>RecSys</td>
<td>375</td>
<td>14.6%</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td>16.4%</td>
</tr>
<tr>
<td>SD</td>
<td></td>
<td>12.8%</td>
</tr>
</tbody>
</table>
RQ2: Dependency Categories

- Conceptual
- Sub-conceptual
- Intra-module
- Unexpected
RQ2: Dependency Categories

- Low number of conceptual dependencies
  - Expected
- Higher number of sub-conceptual and intra-module
  - But this might be a problem…
RQ2: Dependency Categories

- Sub-conceptual
  - Too coarse-grained conceptual dependencies
    - Business vs. Business.Service

- Intra-module dependencies
  - Possible need for refinement of architectural rules
  - Disorganized system evolution
RQ2: Dependency Categories

- High number of unexpected dependencies
  - Expected
  - Not only violations but also undocumented rules
    - AspectJ and RecSys
    - Ignored dependencies (e.g. util)
RQ3: Distinction of Dependencies

• Automation of the recovery of architectural rules
  • Analysis of the support metric
    • Percentage of the elements of module X that depend on elements of module Y
    • Different perspectives

Legend:
- Implemented Dependency
- (Sub-)Module
- Allowed Dependency
AVG and MED of sub-conceptual dependencies higher than conceptual dependencies (but Metrics)
Confirms that architectural rules should be finer-grained
Lack of intra-module rules (only for some systems)
Unexpected dependencies: low support unless they correspond to undocumented rules (significant difference)
There is a large gap between conceptual architecture and source code. Both architecture and rule conformance.

Sub-conceptual and intra-module dependencies typically not investigated. They can provide information about the quality of the system architecture.

Unexpected dependencies can be identified by the support metric. Identification of undocumented rules or architectural violations.
"Most architectures are accidental, not intentional"
- Grady Booch
The WGB Method
Software Architecture Recovery
Architecture Recovery

- Approaches to identify modules
  - Pattern-based approaches
    - Rely on catalogues that contain known high-level patterns
  - Clustering approaches
    - Search for similarities among source code elements to group them into clusters

- Metrics to evaluate these approaches
  - MoJoFM
    - Number of move or join operations
  - Architecture-to-architecture (a2a)
    - Distance between ground-truth and recovered architecture
  - Cluster-to-Cluster Coverage (c2c\text{cvg})
    - Degree of overlap between the implementation-level entities contained in two clusters

Recovering Architectural Rules: Goals

- Rules must be expressed at the **highest granularity level** as possible
  - Implemented rules may capture hidden information, not expressed in conceptual rules
  - Rules associated with sparse dependencies must be fine-grained
WGB Method Overview

• **Input**
  - Package structure
  - Source code dependencies

• **Steps**
  - Calculation of the module dependency strength metric
  - Pairwise clusterisation of dependencies
  - Selection of architectural rules

Module Dependency Strength (MDS)

- **Intensity**
  - Captures the percentage of elements of a module $X$ that depend on elements of a module $Y$

$$
\text{int}_S(F_3, S_2) = \frac{1}{3} = 0.33 \quad \text{int}_T(F_3, S_2) = \frac{1}{2} = 0.50
$$
Module Dependency Strength (MDS)

• **Distribution**
  - Captures the percentage of sub-modules of the module X that depend on sub-modules of the module Y

\[
dst_{Spp}(P,S) = \frac{2}{3} = 0.67 \quad \text{and} \quad dst_{Tpp}(P,S) = \frac{3}{4} = 0.75
\]
Module Dependency Strength (MDS)

MDS is the sum of the intensity of the source and target modules, weighted by their normalised distribution.

$$MDS(P, S) = 0.47 \times 0.44 + 0.53 \times 0.36 = 0.21 + 0.19 = 0.40$$
### Pairwise Clusterization of Dependencies

- **Possible architectural rules**

<table>
<thead>
<tr>
<th>Parent-to-Parent</th>
<th>Parent-to-Child</th>
<th>Child-to-Parent</th>
<th>Child-to-Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>P → S</td>
<td>P → S1</td>
<td>F2 → S</td>
<td>F2 → S1</td>
</tr>
<tr>
<td></td>
<td>P → S2</td>
<td>F3 → S</td>
<td>F2 → S2</td>
</tr>
<tr>
<td></td>
<td>P → S3</td>
<td></td>
<td>F2 → S3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F3 → S1</td>
<td>F3 → S1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F3 → S2</td>
</tr>
</tbody>
</table>

![Diagram](image-url)
**Pairwise Clusterization of Dependencies**

- **Possible architectural rules**

<table>
<thead>
<tr>
<th>Parent-to-Parent</th>
<th>Parent-to-Child</th>
<th>Child-to-Parent</th>
<th>Child-to-Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>P → S</td>
<td>P → S1</td>
<td>F2 → S</td>
<td>F2 → S1</td>
</tr>
<tr>
<td></td>
<td>P → S2</td>
<td>F3 → S</td>
<td>F2 → S2</td>
</tr>
<tr>
<td></td>
<td>P → S3</td>
<td></td>
<td>F2 → S3</td>
</tr>
</tbody>
</table>

![Diagram showing pairwise clusterization of dependencies with arrows indicating parent-to-parent, parent-to-child, child-to-parent, and child-to-child relationships.]
• Pairwise Clusterization of Dependencies

• Possible architectural rules

<table>
<thead>
<tr>
<th>Parent-to-Parent</th>
<th>Parent-to-Child</th>
<th>Child-to-Parent</th>
<th>Child-to-Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>P → S</td>
<td>P → S1</td>
<td>F2 → S</td>
<td>F2 → S1</td>
</tr>
<tr>
<td></td>
<td>P → S2</td>
<td>F3 → S</td>
<td>F2 → S2</td>
</tr>
<tr>
<td></td>
<td>P → S3</td>
<td></td>
<td>F2 → S3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F3 → S1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F3 → S2</td>
</tr>
</tbody>
</table>

![Diagram](image)
### Pairwise Clusterization of Dependencies

- **Possible architectural rules**

<table>
<thead>
<tr>
<th>Parent-to-Parent</th>
<th>Parent-to-Child</th>
<th>Child-to-Parent</th>
<th>Child-to-Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \to S$</td>
<td>$P \to S_1$</td>
<td>$F_2 \to S$</td>
<td>$F_2 \to S_1$</td>
</tr>
<tr>
<td></td>
<td>$P \to S_2$</td>
<td>$F_2 \to S_2$</td>
<td>$F_2 \to S_2$</td>
</tr>
<tr>
<td></td>
<td>$P \to S_3$</td>
<td>$F_2 \to S_3$</td>
<td>$F_2 \to S_3$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$F_3 \to S$</td>
<td>$F_3 \to S_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$F_3 \to S_2$</td>
</tr>
</tbody>
</table>

![Diagram showing pairwise clusterization of dependencies](image)

10/15/2019  
ingridnunes@inf.ufrgs.br  
46
# Pairwise Clusterization of Dependencies

<table>
<thead>
<tr>
<th>Parent-to-Parent</th>
<th>Parent-to-Child</th>
<th>Child-to-Parent</th>
<th>Child-to-Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \rightarrow S(0.40)$</td>
<td>$P \rightarrow S_1(0.20)$</td>
<td>$F_2 \rightarrow S(0.31)$</td>
<td>$F_2 \rightarrow S_2(0.40)$</td>
</tr>
<tr>
<td>$P \rightarrow S_2(0.43)$</td>
<td>$F_3 \rightarrow S(0.38)$</td>
<td>$F_2 \rightarrow S_3(0.25)$</td>
<td></td>
</tr>
<tr>
<td>$P \rightarrow S_3(0.17)$</td>
<td>Mean = 0.27</td>
<td>$F_3 \rightarrow S_1(0.33)$</td>
<td>Mean = 0.35</td>
</tr>
<tr>
<td>Mean = 0.40</td>
<td>Mean = 0.34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Highest MDS Mean  

![Diagram](image)

10/15/2019  ingridnunes@inf.ufrgs.br  47
Selection of Architectural Rules

• Package hierarchies might have redundant rules

• Selection of the subset of non-redundant rules that maximizes the MDS metric

\[
\max_{1 \leq i \leq |D|} \sum_{i=1}^{\frac{|D|}{2}} \left( \omega(d_i) \times MDS(d_{i_S}, d_{i_T}) \right)
\]

\[
in_x + in_y \leq 1, \forall d_x, d_y (\text{redundant}(d_x, d_y))
\]
• **Empirical Study**
  • Architecture recovery
    • By a set of developers
    • By the WGB method
  • Comparison between architectures
    • Evaluation of divergences by developers

• **Results**
  • The WGB method provides **finer-grained rules**
  • The provided information is **useful**
  • Developers would use the provided model as an **architecture model**
Challenges

• How to generate **views** that are at the right level for communication

• How to **enforce rule compliance** during software evolution

• How to distinguish **undocumented rules** from **architectural violations**
Conclusion

Software Architecture

- Fundamental for the organised evolution of software systems

3. Bug: violated business rule

View
Business
Data

4. Duplicated code!

WGB Method Overview

- Input
  - Package structure
  - Source code dependencies

- Steps
  - Calculation of the module dependency strength metric
  - Pairwise clusterisation of dependencies
  - Selection of architectural rules

Summary

- There is a large gap between conceptual architecture and source code
- Both architecture and rule conformance

- Sub-conceptual and intra-module dependencies
- Typically not investigated
- Can provide information about the quality of the system architecture

- Unexpected dependencies
- Can be identified by the support metric
- Identification of undocumented rules or architectural violations

Challenges

- How to generate views that are at the right level of communication

- How to enforce rule compliance during software evolution

- How to distinguish undocumented rules from architectural violations
Inscrições para mestrado e doutorado abertas!

PPGC-INF/UFRGS abre seleção para Mestrado.

Seleção de Doutorado para 2020/1
Concursos para Professor no INF-UFRGS
Seleção para mestrado
Alunos especiais aceitos no PPGC - 2019/2

http://www.inf.ufrgs.br/ppgc/
Thank you!

- Software Architecture Recovery: Importance, Challenges, and Methods
- Ingrid Nunes
  - ingridnunes@inf.ufrgs.br

Vaniu Zapalowski
Daltro Nunes