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

Carnegie Mellon University
Software Engineering Institute

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.

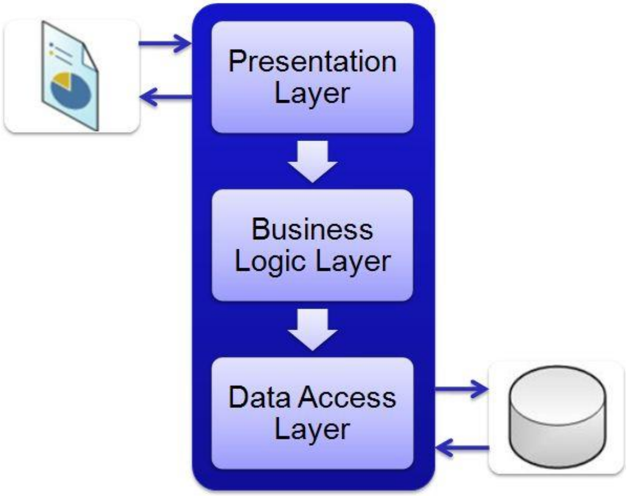
Modern definitions are definitions from Software

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

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

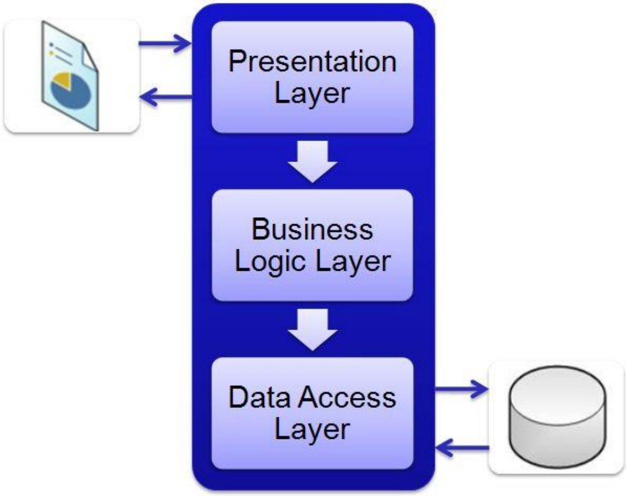


Software Architecture

"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

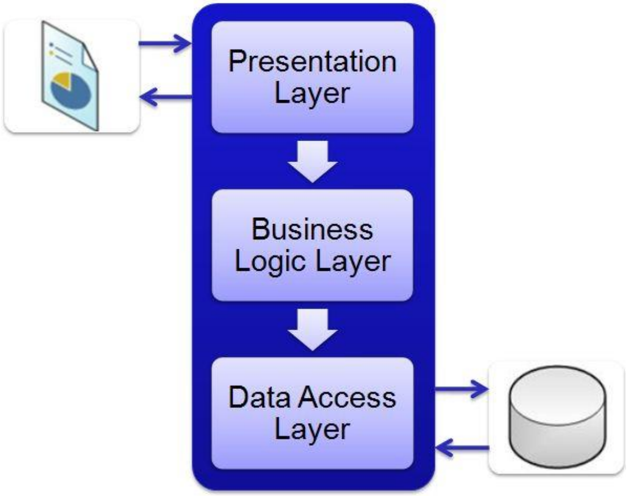


Software Architecture

"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

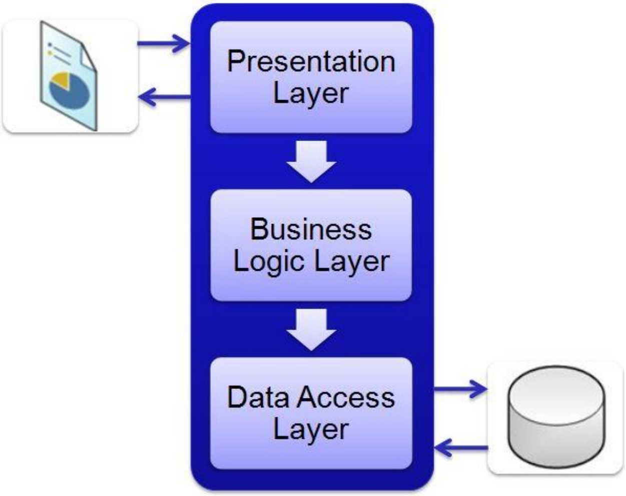


Software Architecture

"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
- Relationships and constraints
  - Dependencies



Software Architecture

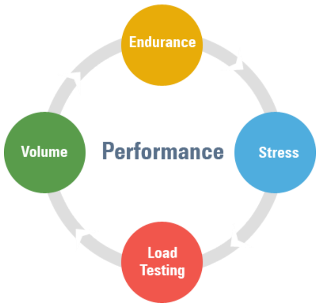
- Key principles

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

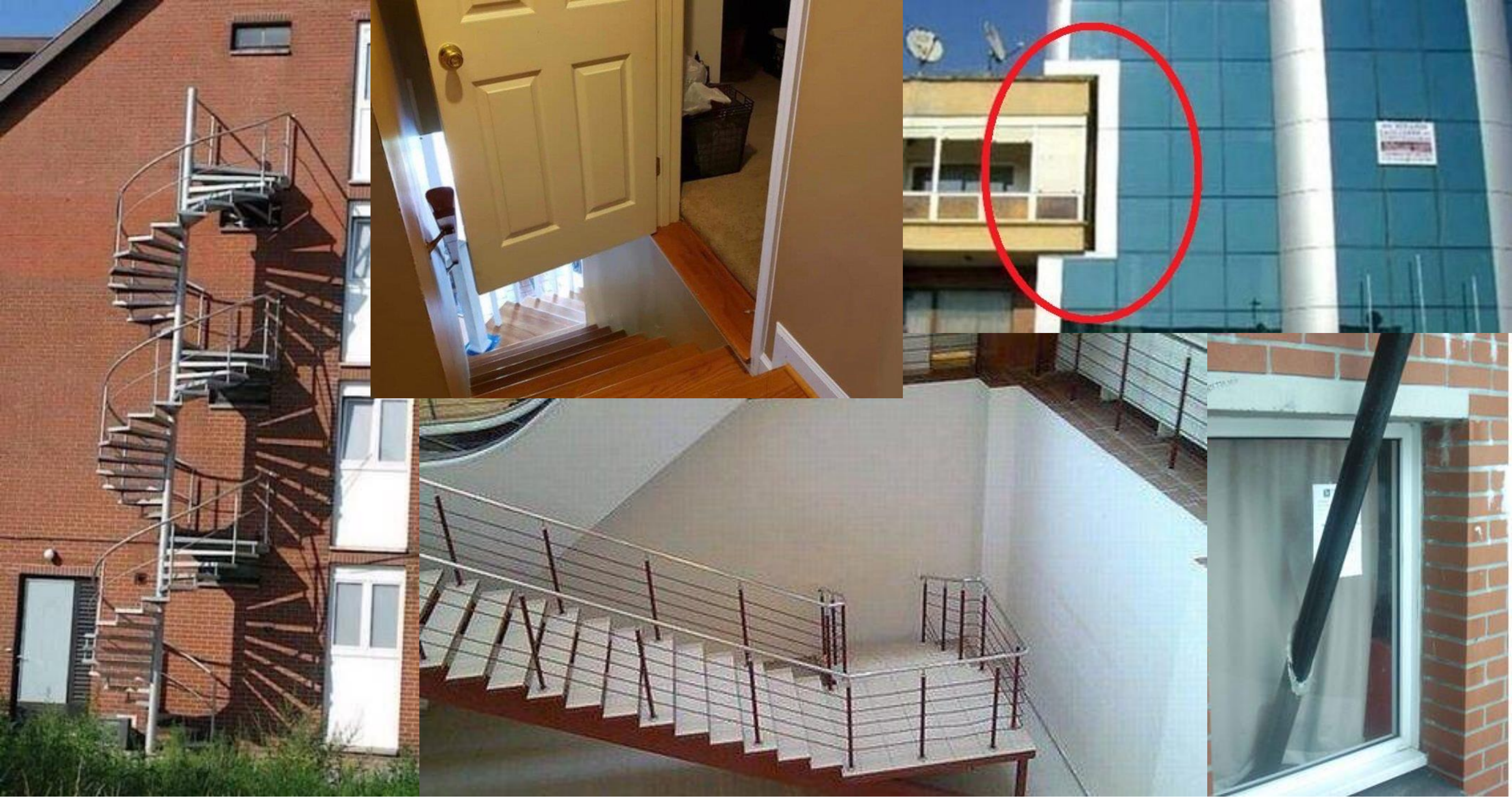
  - **Modularization**
    - **Decomposition** of a system into groups of subsystems and components
    - Physical packaging of entities
  - **Separation of Concerns**
    - Isolation of responsibilities
    - If a component plays different roles in different contexts, these roles must be separated

Software Architecture

- Key design decisions
  - **Adopted technologies**
  - **Non-functional requirements**



.inf INSTITUTO DE INFORMÁTICA UFRGS

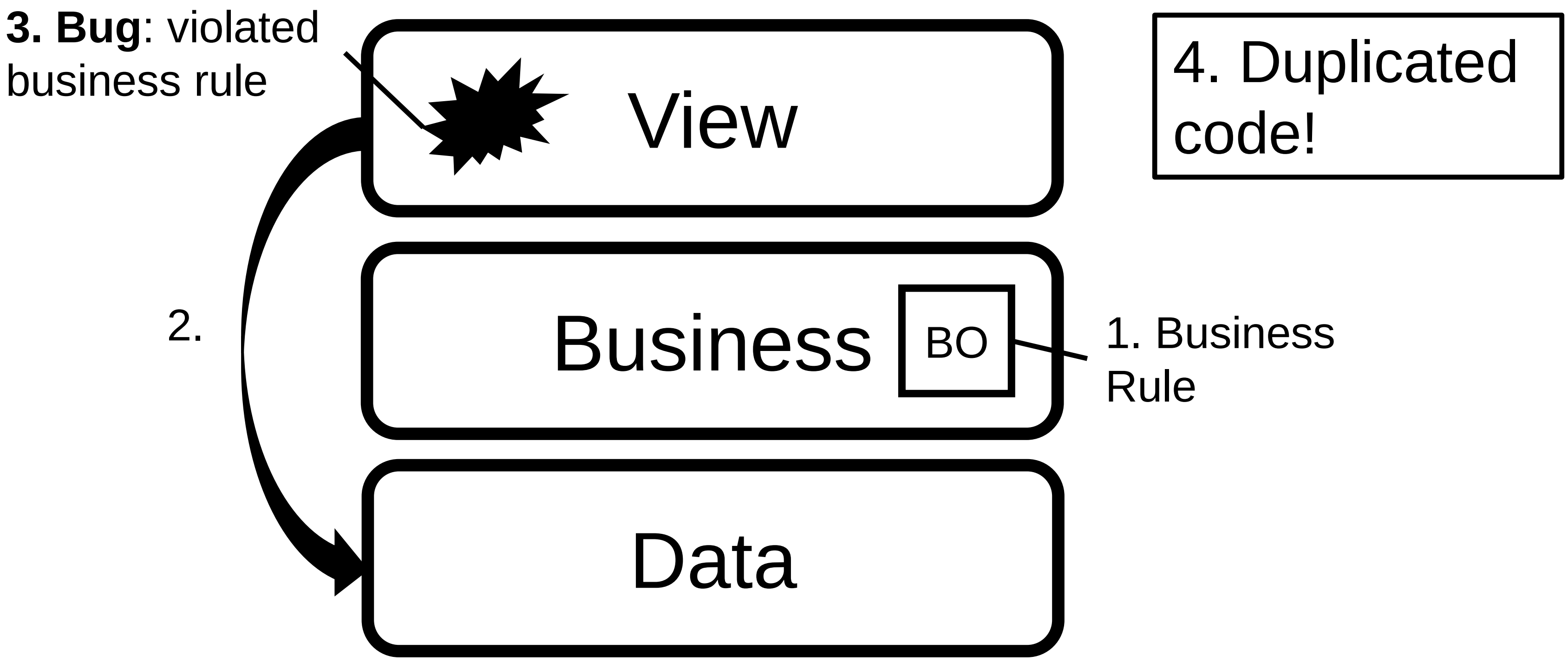


If you think good architecture is expensive, try bad architecture."

- Brian Foot and Joseph Yoder

Software Architecture

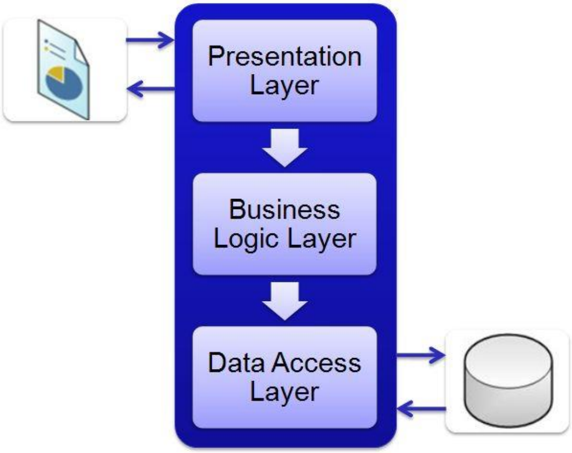
- Fundamental for the organised evolution of software systems



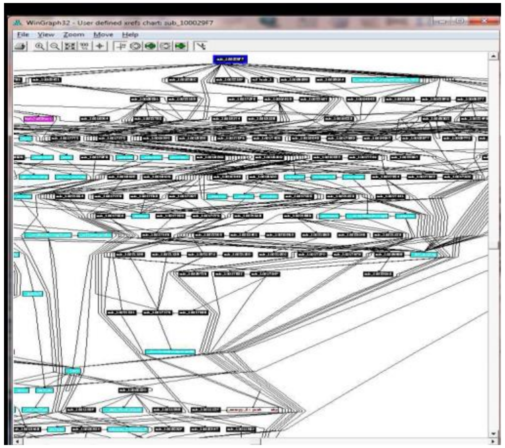
Software Architecture

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

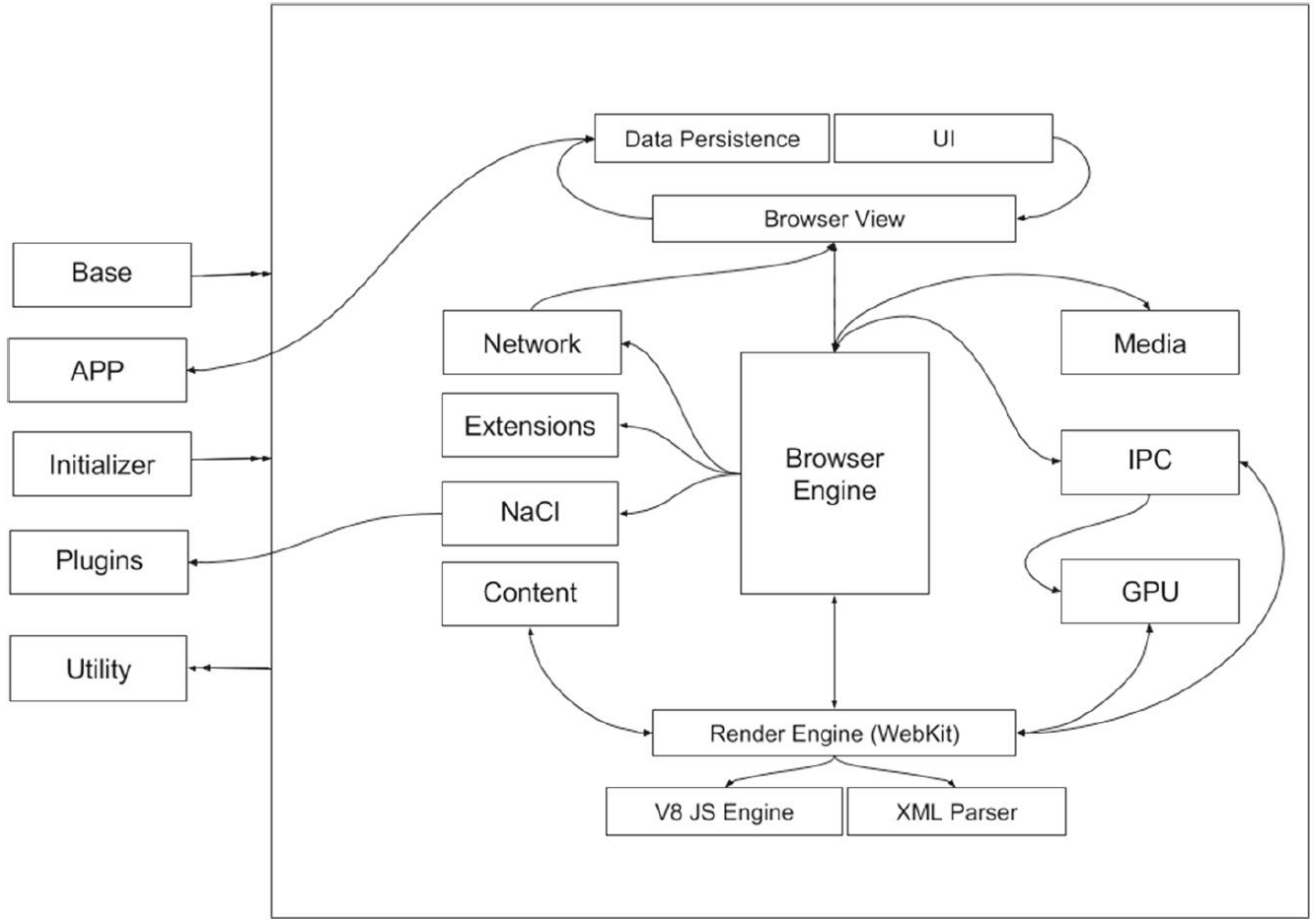
Conceptual Architecture



Concrete Architecture

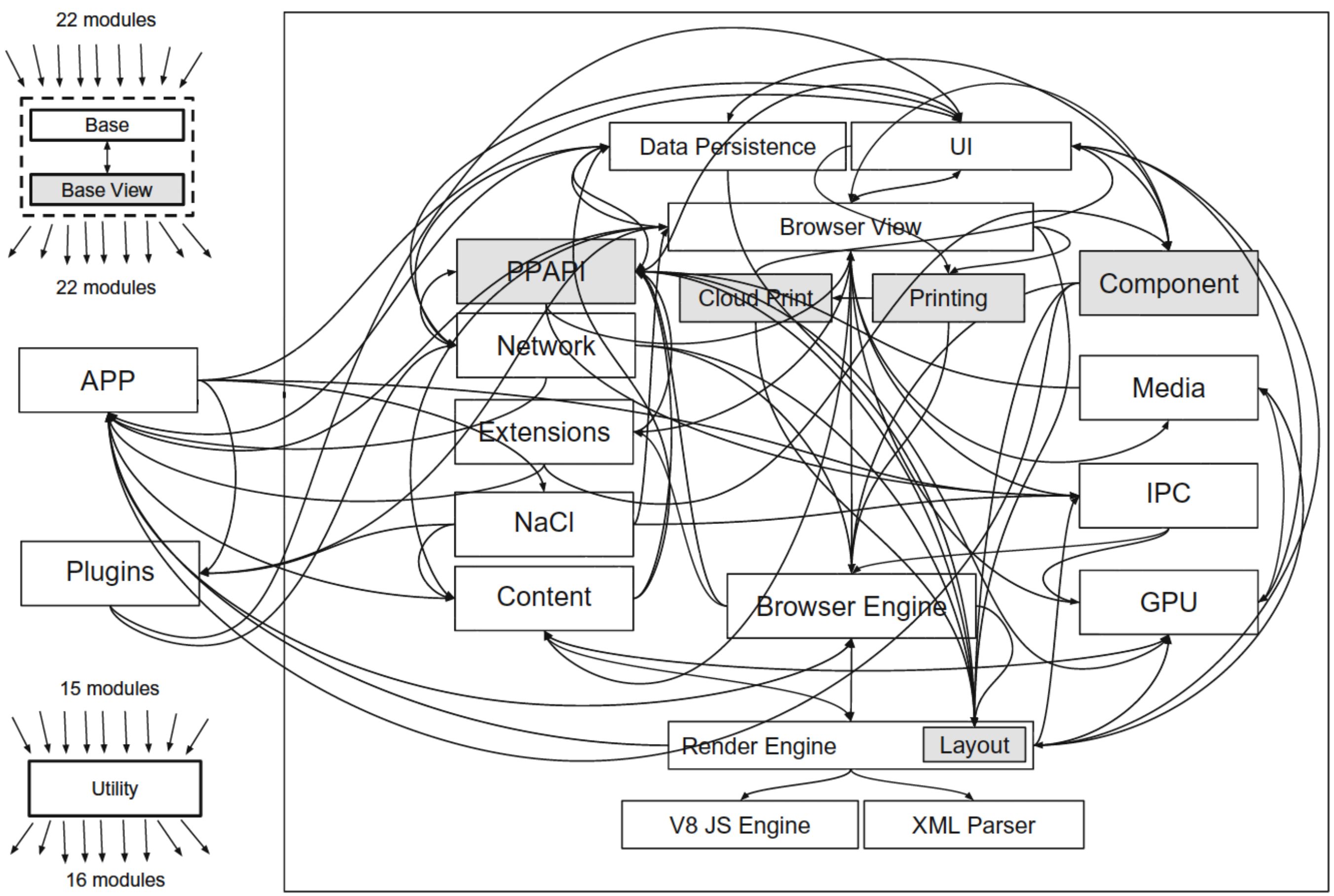


Software Architecture



M. T. Rahman, P. C. Rigby, E. Shihab, The modular and feature toggle architectures of Google Chrome, Empirical Software Engineering (2019) 24:826–853

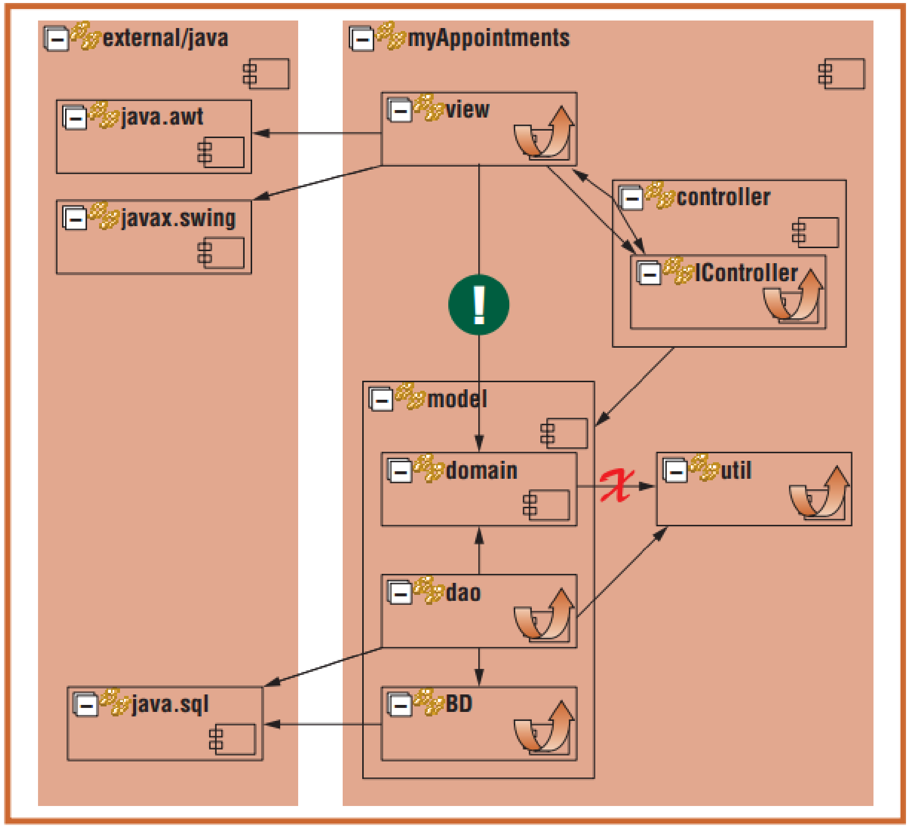
Software Architecture



M. T. Rahman, P. C. Rigby, E. Shihab, The modular and feature toggle architectures of Google Chrome, Empirical Software Engineering (2019) 24:826–853

Software Architecture

- Software Reflexion Models



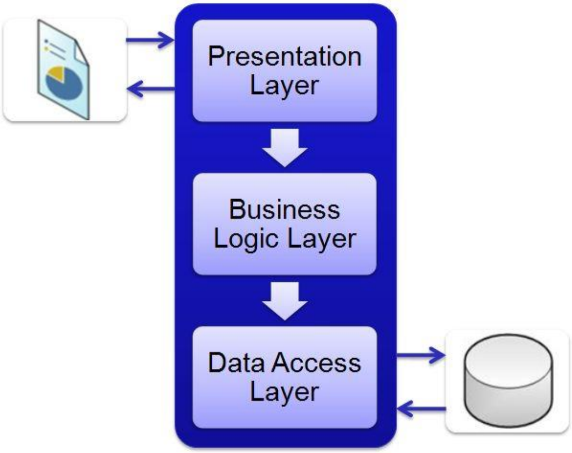
INSTITUTO DE INFORMÁTICA UFRGS

Gail C. Murphy, David Notkin, Kevin J. Sullivan. Software Reflexion Models: Bridging the Gap Between Source and High-Level Models. SIGSOFT FSE 1995: 18-28

Software Architecture

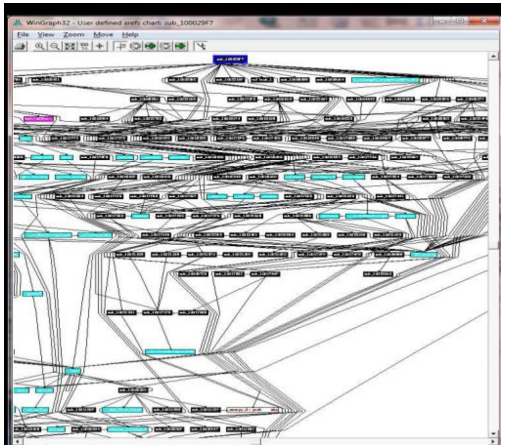
- But… many studies report a high number of **architecture violations**

Conceptual Architecture



≠

Concrete Architecture



Why?

Understanding architecture non-conformance

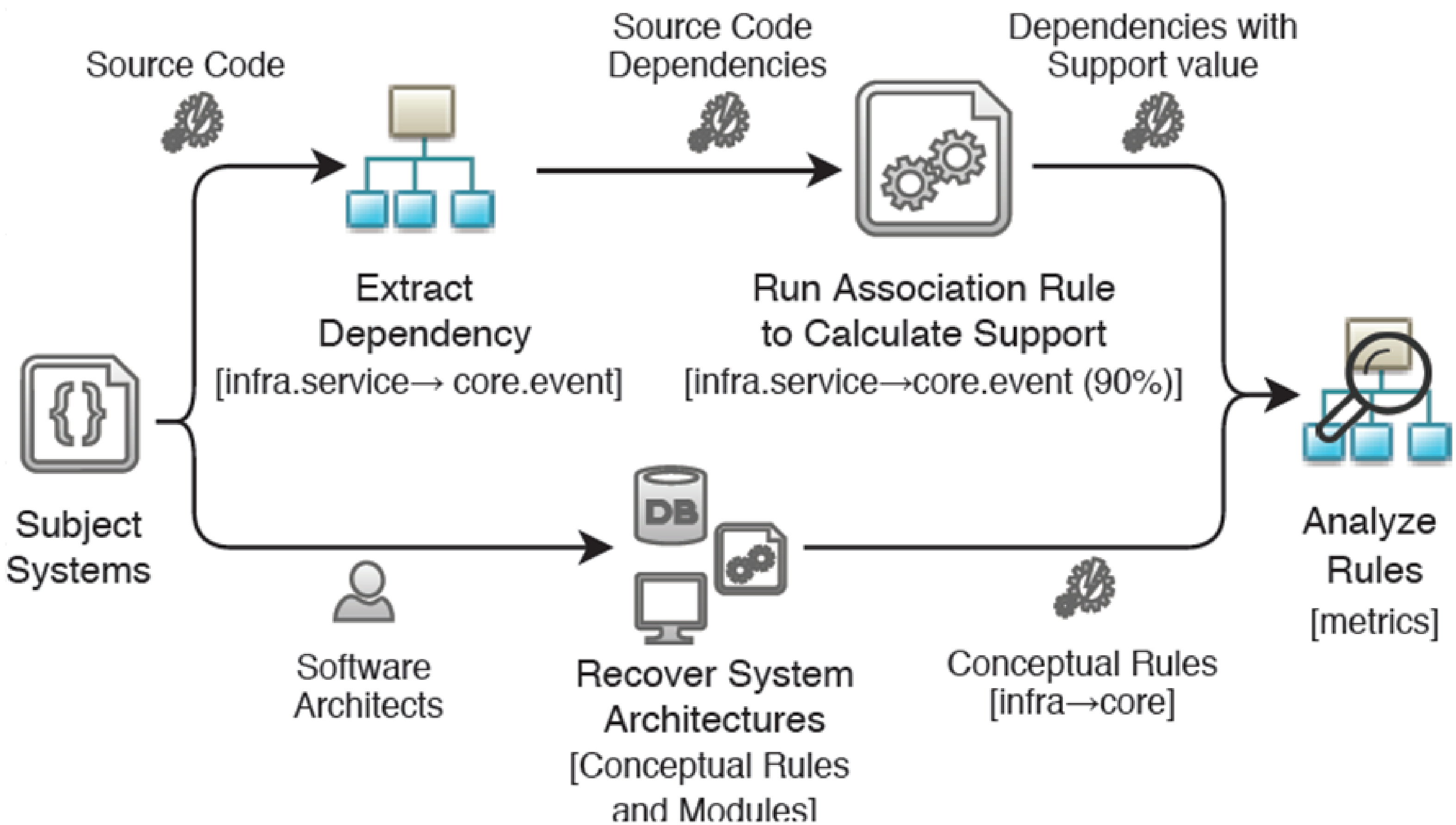
Software Architecture Recovery

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?

ZAPALOWSKI, V. ; NUNES, I. ; NUNES, D. Understanding architecture non-conformance: Why is there a gap between conceptual architectural rules and source code dependencies? SBES 2018.

Procedure Overview

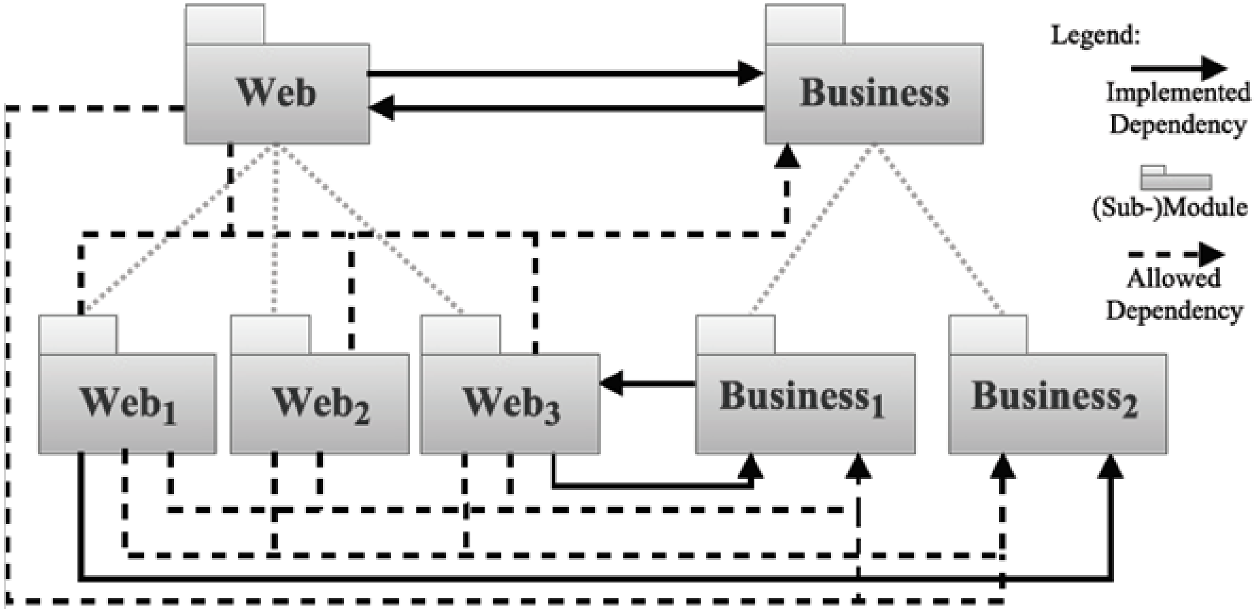


Target Systems

System	LOC	Rules	Architecture
ArchStudio	236.9K	53	Heterogeneous
AspectJ	217.9K	31	Heterogeneous
EC	11.7K	19	Layered
Metrics	15.6K	8	Extended MVC
OLIS	11.4K	13	Layered
RecSys	22.8	19	Heterogeneous

RQ1: Conceptual Architecture vs. Dependencies

- Analysis of implemented dependencies
  - Architecture conformance



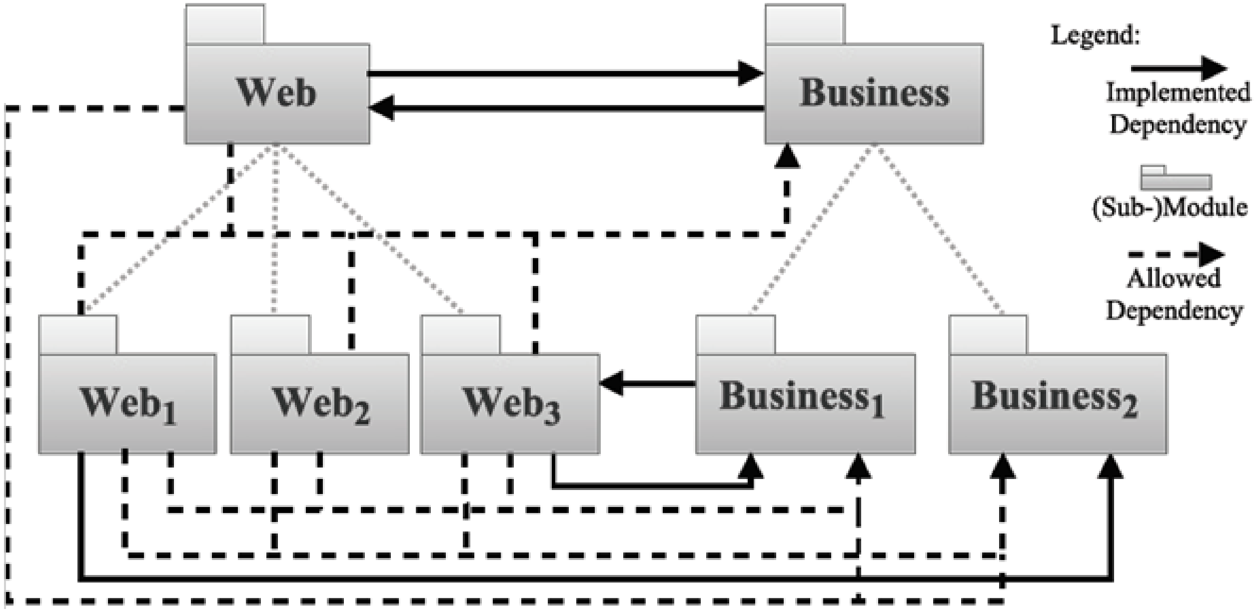
RQ1: Conceptual Architecture vs. Dependencies

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

System	Implemented Dependencies	Architecture Conformance
ArchStudio	1178	26.1%
AspectJ	683	28.7%
EC	135	94.1%
Metrics	45	55.6%
OLIS	86	93.0%
RecSys	375	36.0%
Mean		55.6%
SD		31.2%

RQ1: Conceptual Architecture vs. Dependencies

- Analysis of allowed dependencies
  - Rule conformance



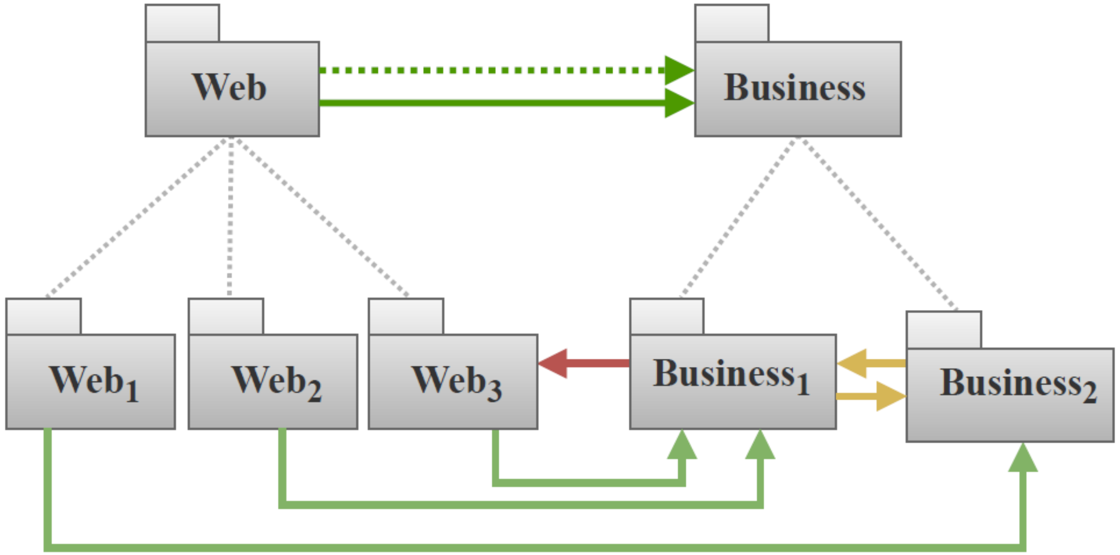
RQ1: Conceptual Architecture vs. Dependencies

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

System	Allowed Dep.	Rule Conf.
ArchStudio	1178	1.8%
AspectJ	683	14.0%
EC	135	16.8%
Metrics	45	39.7%
OLIS	86	11.4%
RecSys	375	14.6%
Mean		16.4%
SD		12.8%

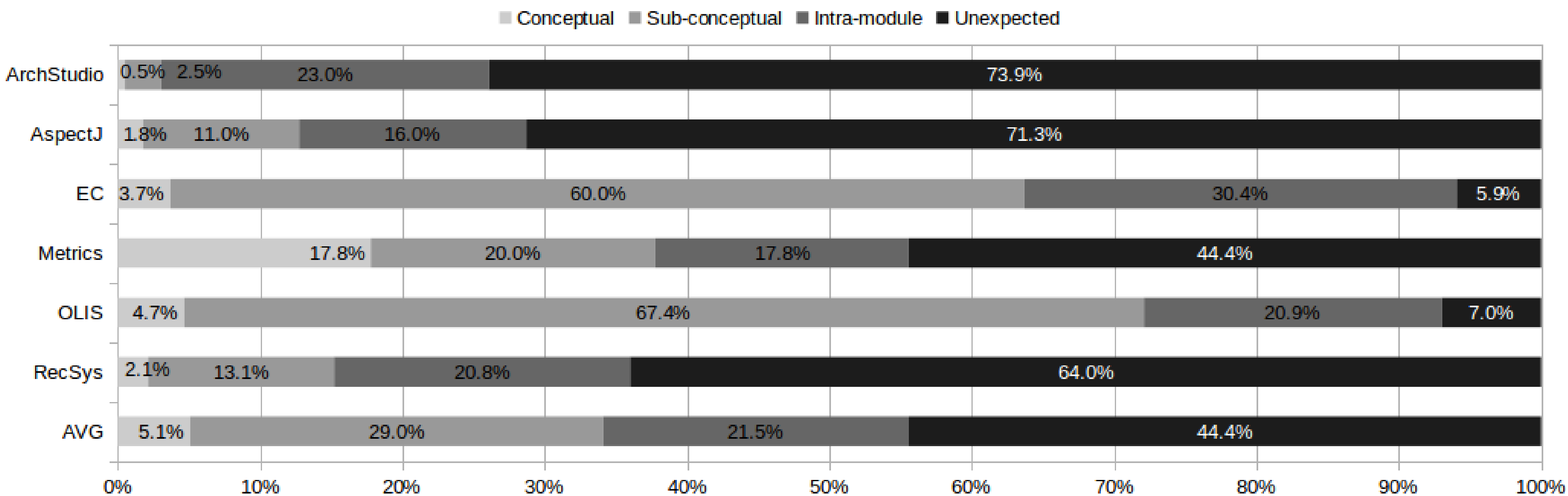
RQ2: Dependency Categories

- Conceptual
- Sub-conceptual
- Intra-module
- Unexpected



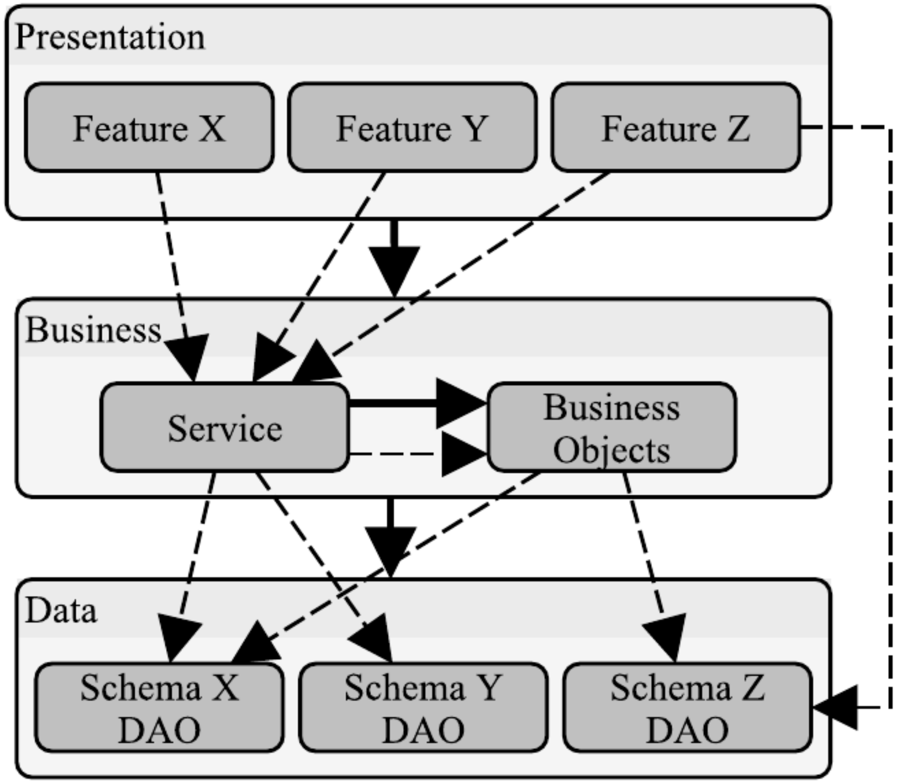
RQ2: Dependency Categories

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



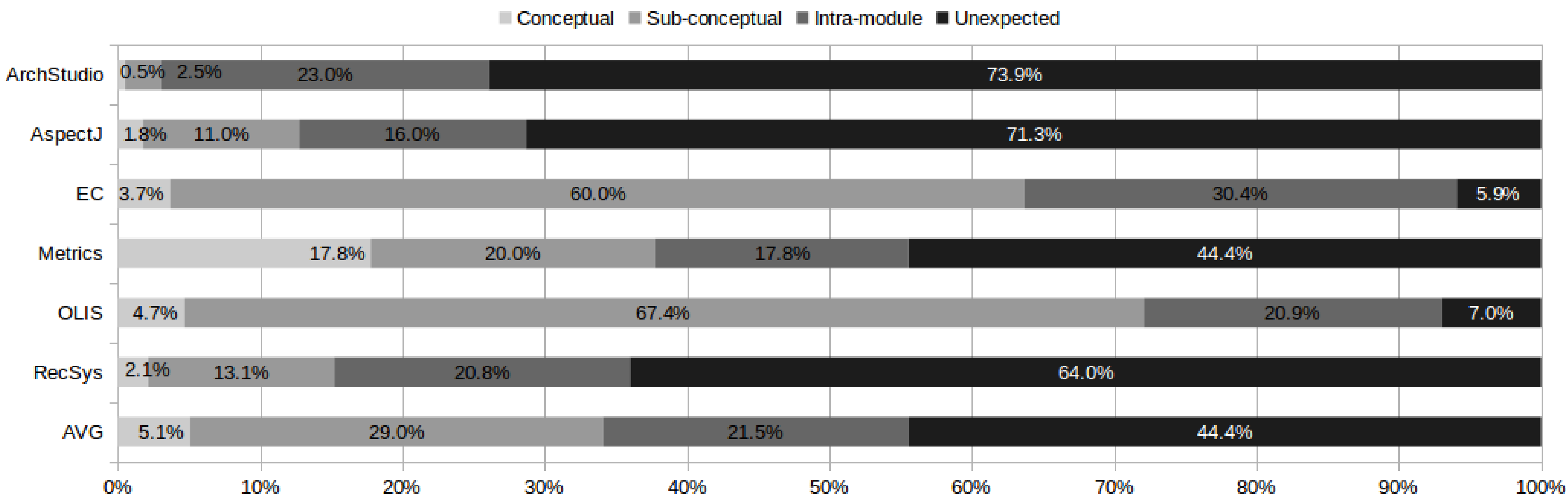
RQ2: Dependency Categories

- Sub-conceptual
  - Too coarse-grained conceptual dependencies
    - Business vs. Business.Service
- Intra-module dependencies
  - Possible need for refinement of architectural rules
  - Disorganized system evolution



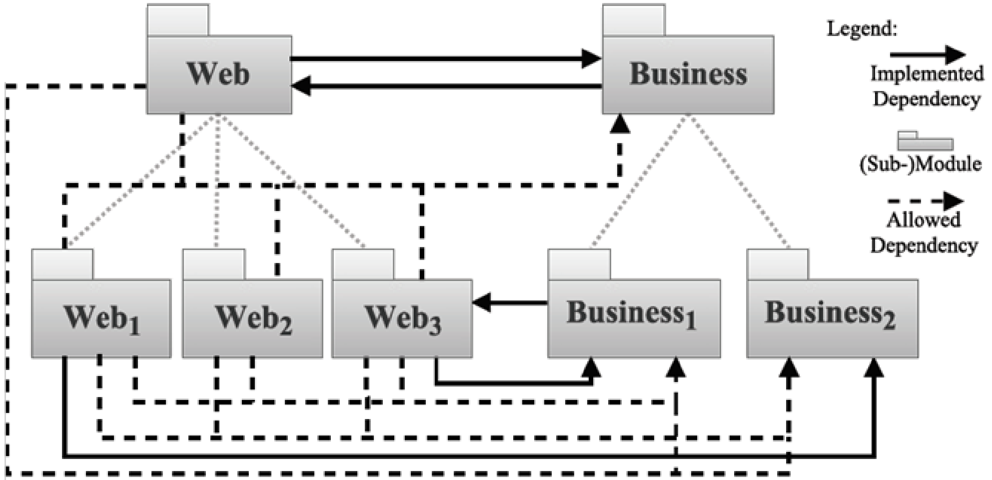
RQ2: Dependency Categories

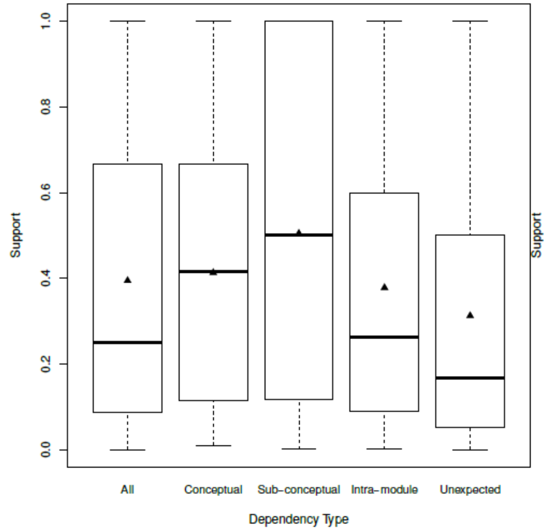
- High number of unexpected dependencies
  - Expected
  - Not only violations but also undocumented rules
    - AspectJ and RecSys
    - Ignored dependencies (e.g. util)



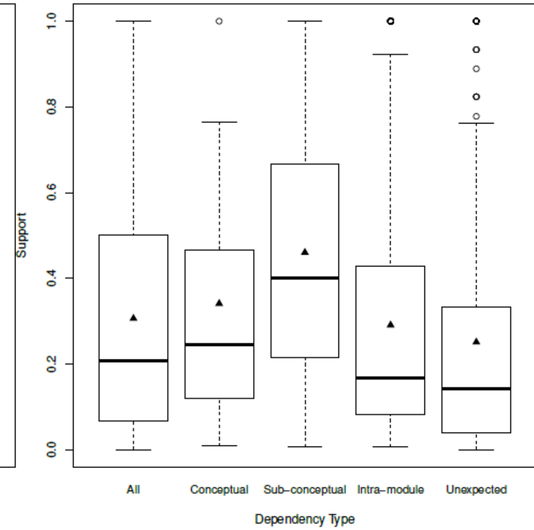
RQ3: Distinction of Dependencies

- 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

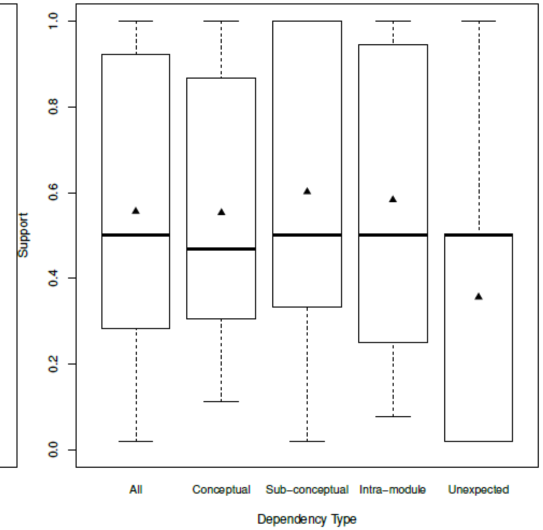




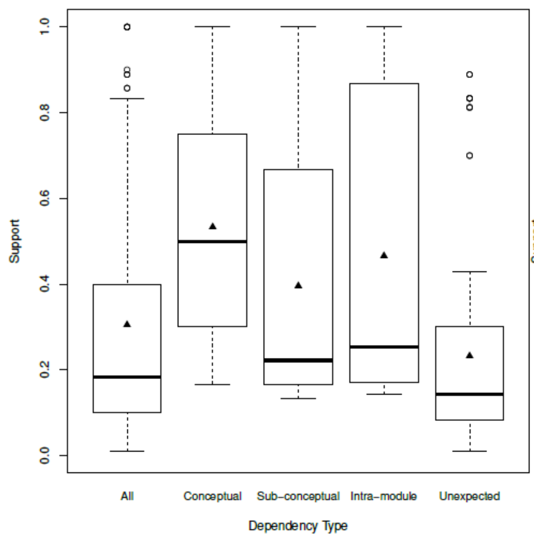
(a) ArchStudio



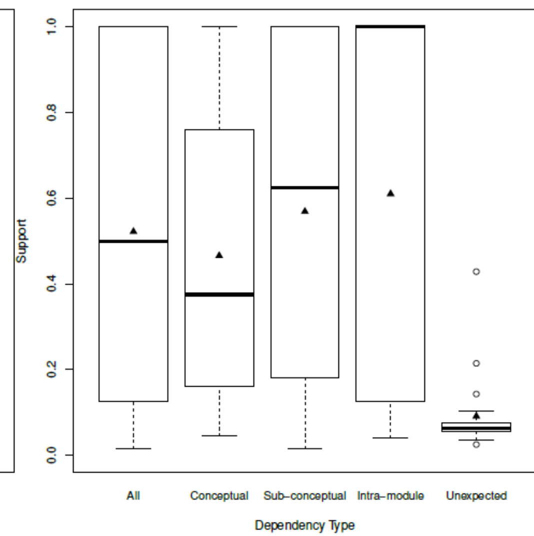
(b) AspectJ



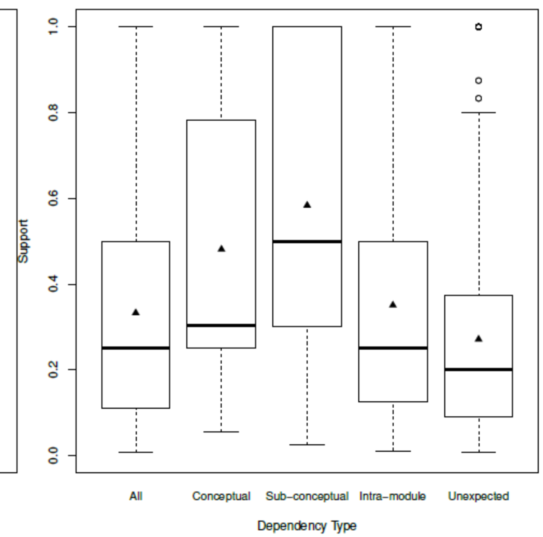
(c) EC



(d) Metrics

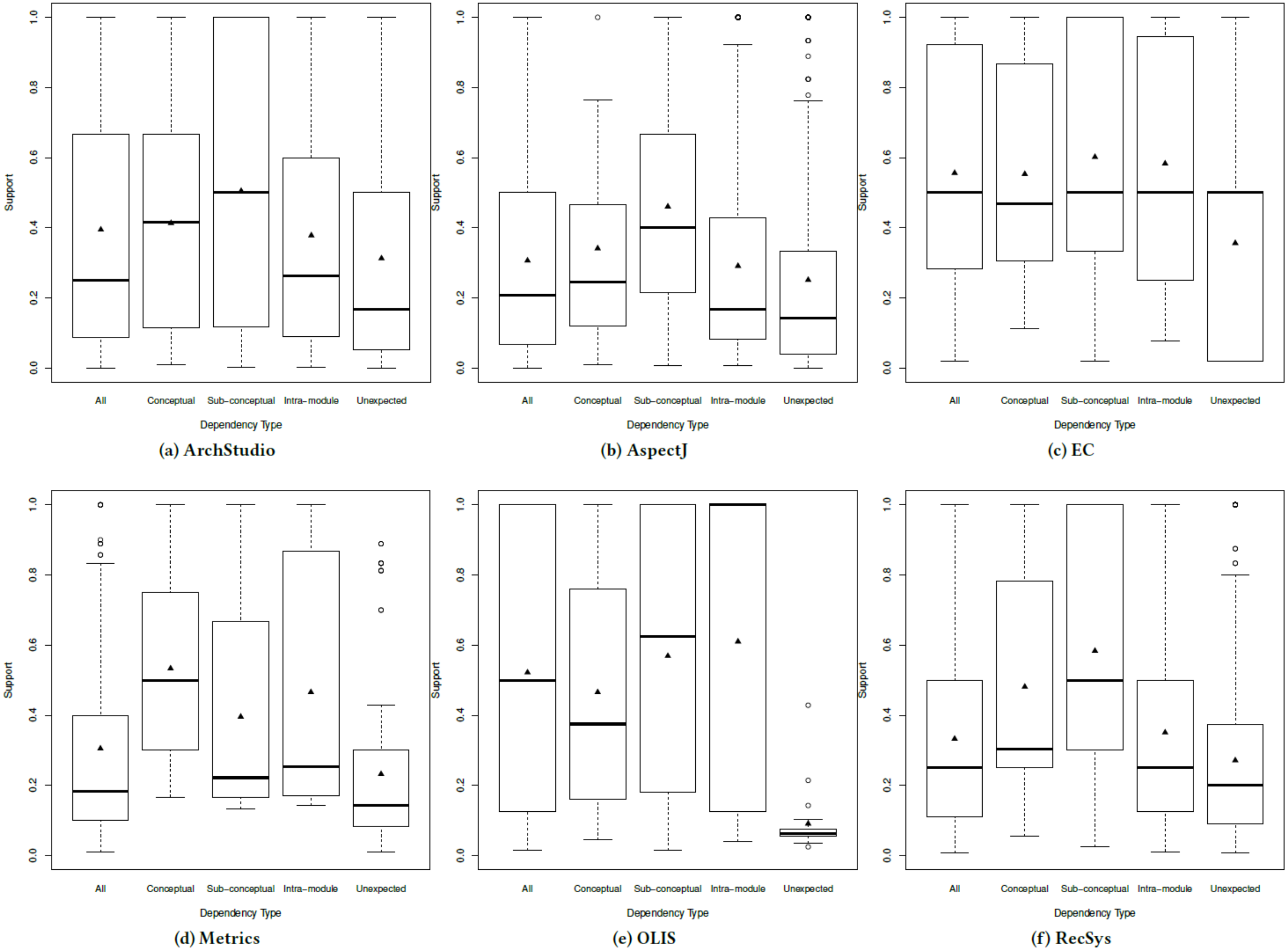


(e) OLIS



(f) RecSys

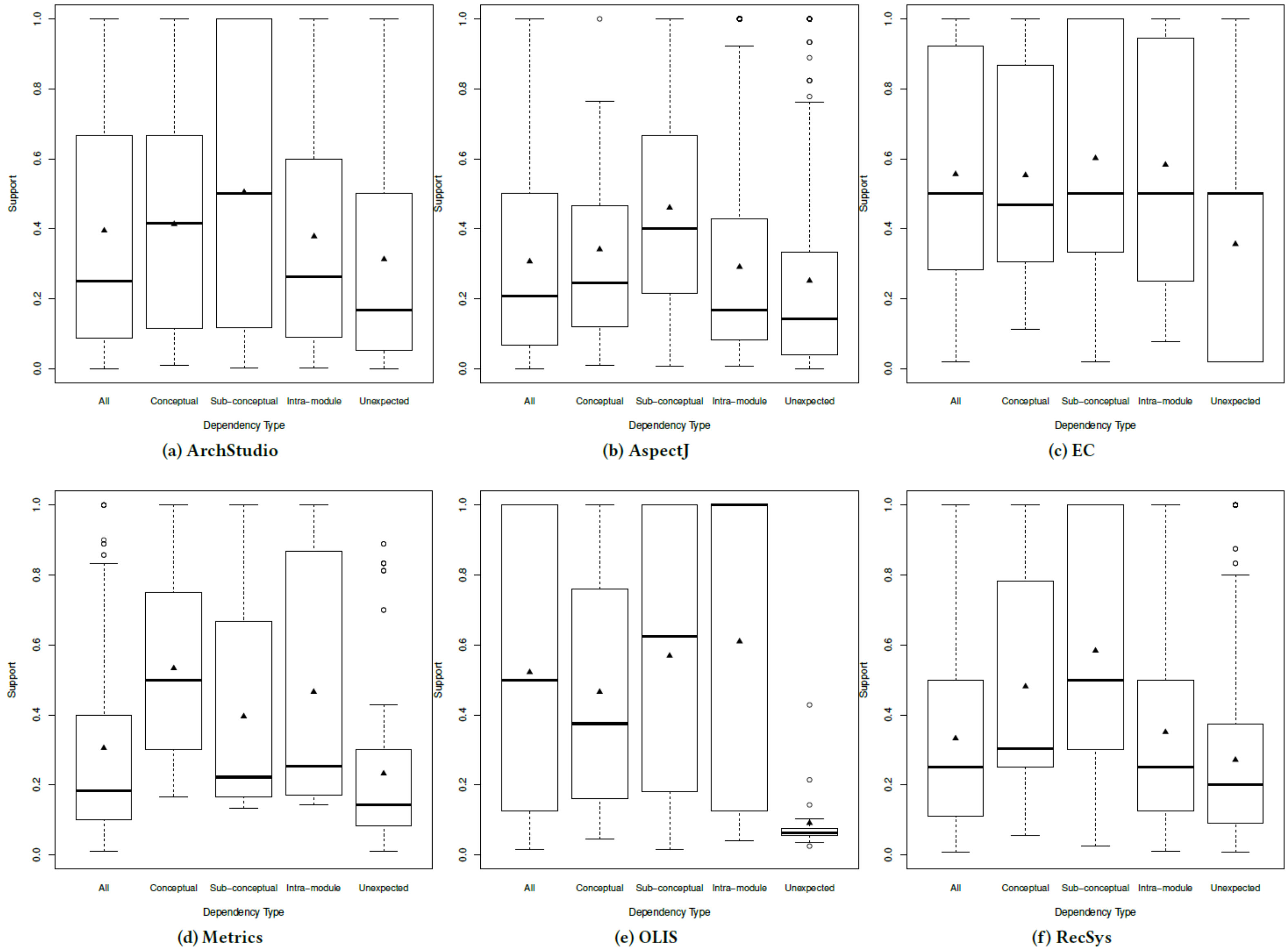
AVG and MED of sub-conceptual dependencies higher than conceptual dependencies (but Metrics)
Confirms that architectural rules should be finer-grained



(a) ArchStudio (b) AspectJ (c) EC

(d) Metrics (e) OLIS (f) RecSys

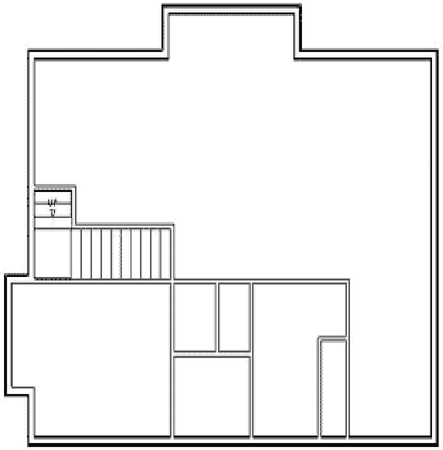
Lack of intra-module rules (only for some systems)

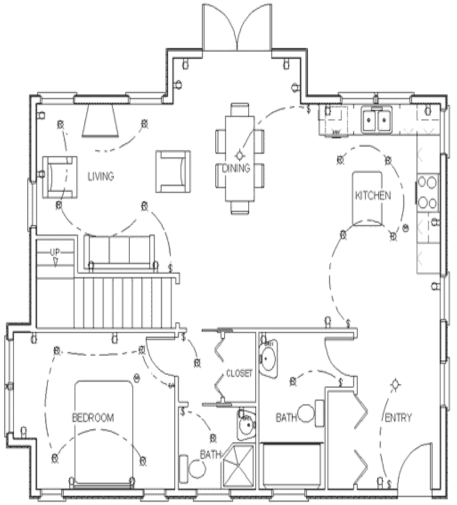


Unexpected dependencies: low support unless they correspond to undocumented rules (significant difference)

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





"Most architectures are accidental, not intentional"

- Grady Booch

The WGB Method

Software Architecture Recovery

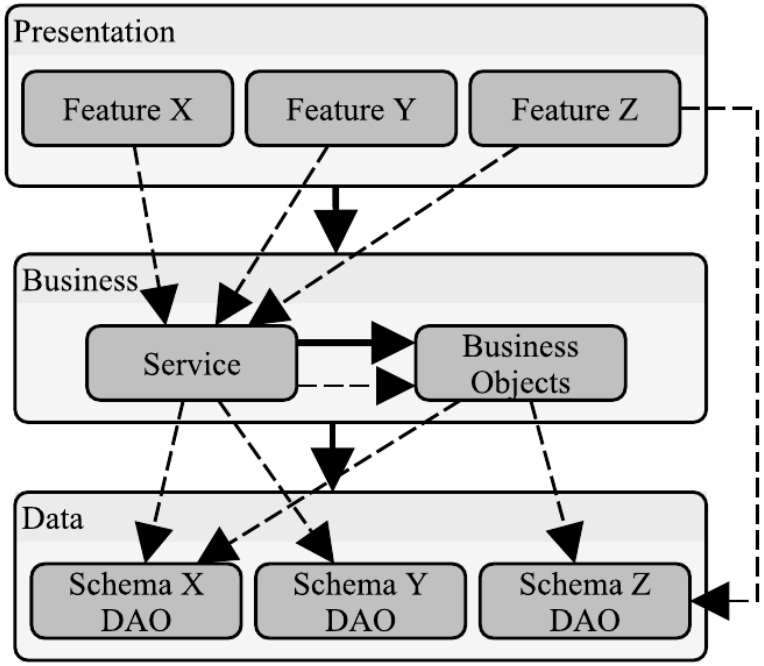
Architecture Recovery

- 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 ($c2c_{cvg}$)**
    - Degree of overlap between the implementation-level entities contained in two clusters

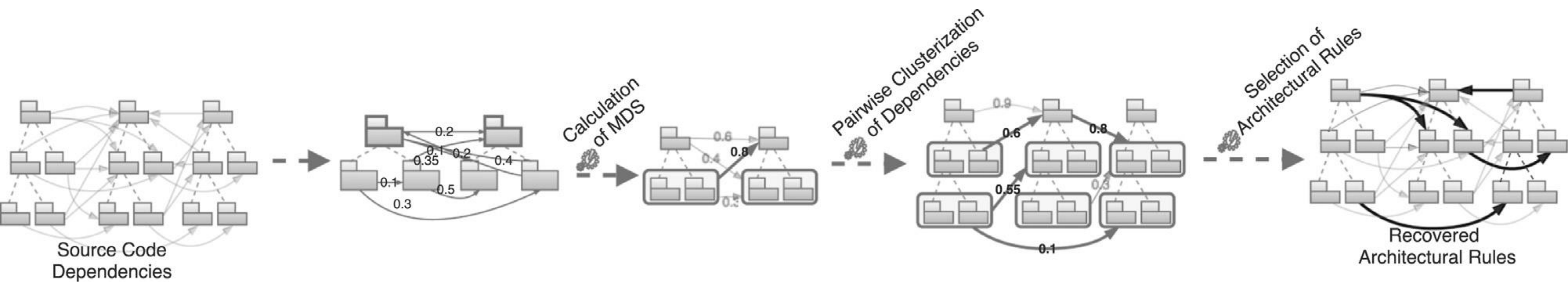
Garcia, Ivkovic, Medvidovic. (2013) A Comparative Analysis of Software Architecture Recovery Techniques. ASE 2013.

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

ZAPALOWSKI, VANIUS ; NUNES, I. ; NUNES, D. . The WGB method to recover implemented architectural rules. INFORMATION AND SOFTWARE TECHNOLOGY, v. 103, p. 125-137, 2018.

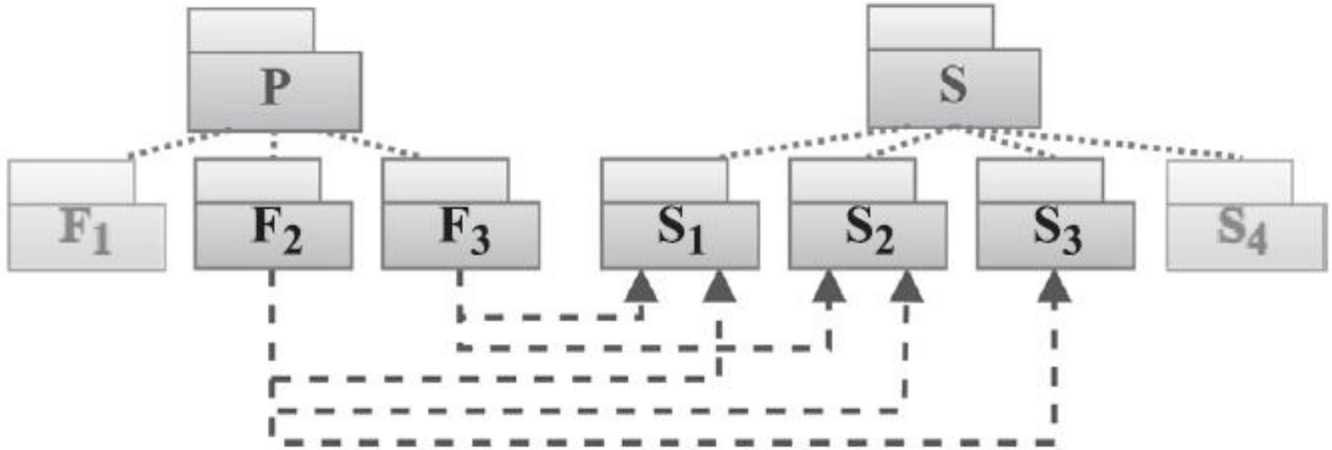
Module Dependency Strength (MDS)

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

$int_S(F_3,S_2) = 1/3 = 0.33 \quad int_T(F_3,S_2) = 1/2 = 0.50$

Module Dependency Strength (MDS)

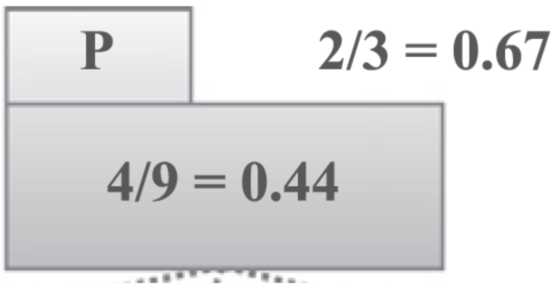
- **Distribution**
  - Captures the percentage of sub-modules of the module X that depend of sub-modules of the module Y

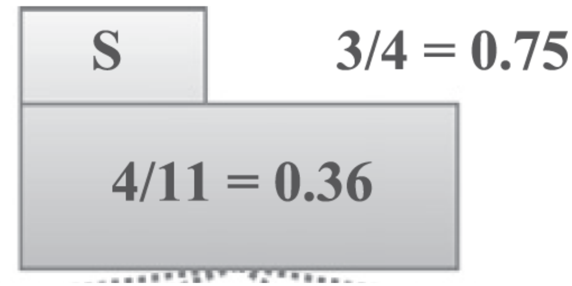


$$dst_{S_{PP}}(P,S) = 2/3 = 0.67 \qquad dst_{T_{PP}}(P,S) = 3/4 = 0.75$$

Module Dependency Strength (MDS)

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



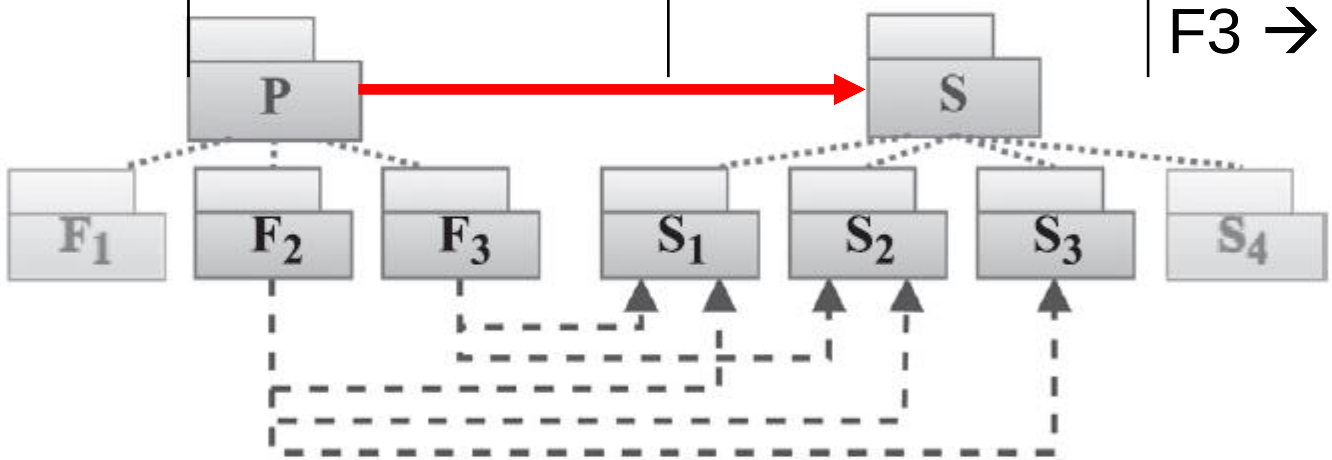


- MDS(P, S) = $0.47 \times 0.44 + 0.53 \times 0.36 =$
  $0.21 + 0.19 =$
  0.40

Pairwise Clusterization of Dependencies

- Possible architectural rules

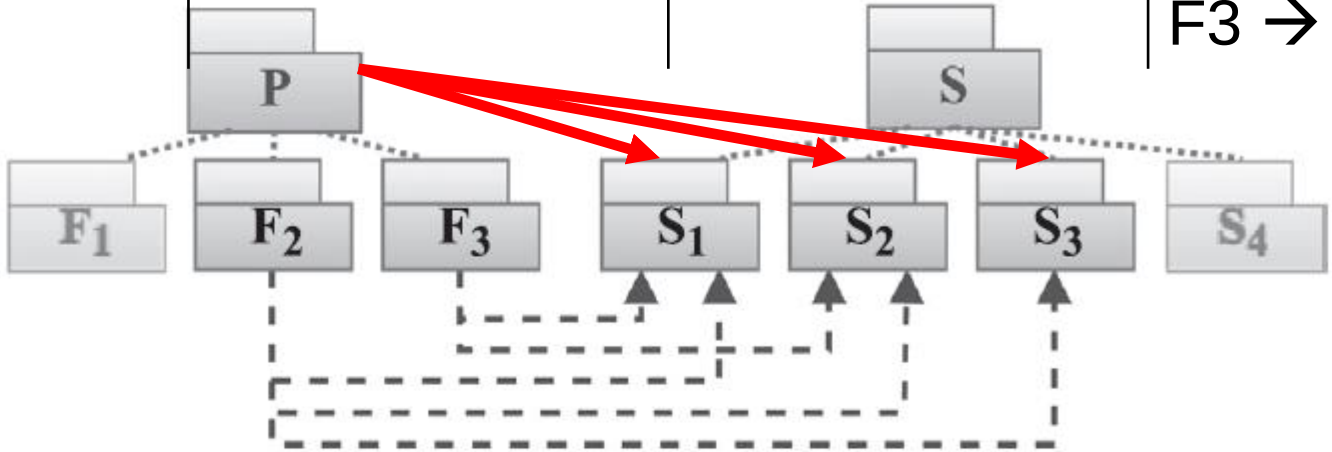
Parent-to-Parent	Parent-to-Child	Child-to-Parent	Child-to-Child
P → S	P → S1 P → S2 P → S3	F2 → S F3 → S	F2 → S1 F2 → S2 F2 → S3 F3 → S1 F3 → S2



Pairwise Clusterization of Dependencies

- Possible architectural rules

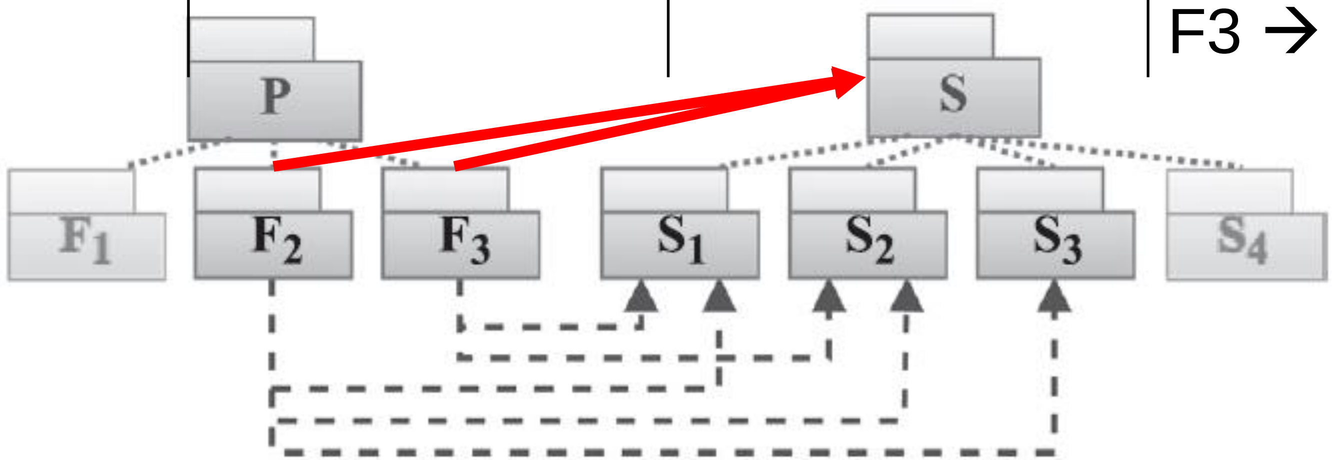
Parent-to-Parent	Parent-to-Child	Child-to-Parent	Child-to-Child
P → S	**P → S1**	F2 → S	F2 → S1
	P → S2	F3 → S	F2 → S2
	P → S3		F2 → S3
			F3 → S1
			F3 → S2



Pairwise Clusterization of Dependencies

- Possible architectural rules

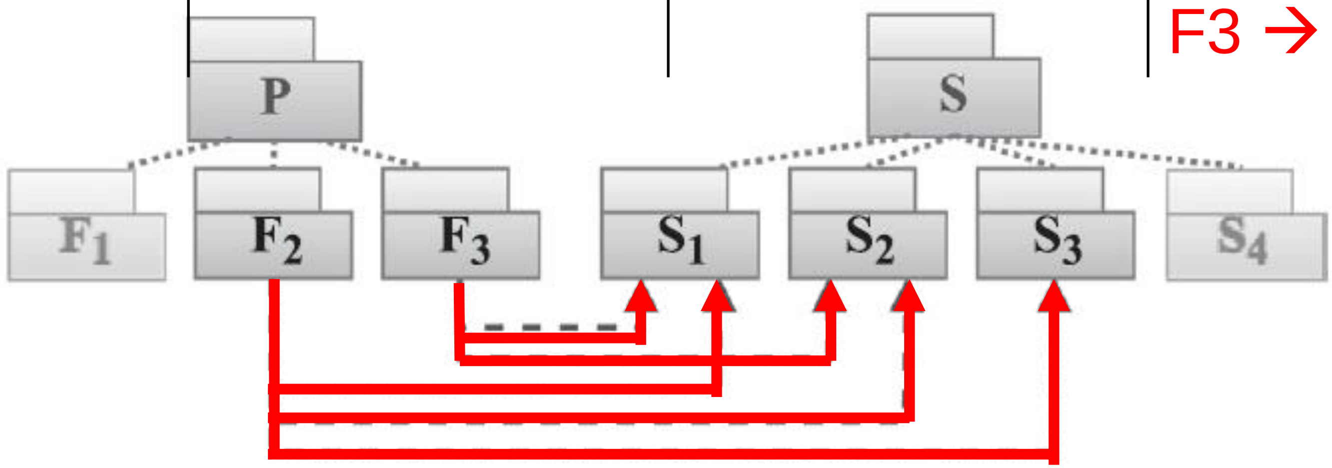
Parent-to-Parent	Parent-to-Child	Child-to-Parent	Child-to-Child
P → S	P → S1	F2 → S	F2 → S1
	P → S2	F3 → S	F2 → S2
	P → S3		F2 → S3
			F3 → S1
			F3 → S2



Pairwise Clusterization of Dependencies

- Possible architectural rules

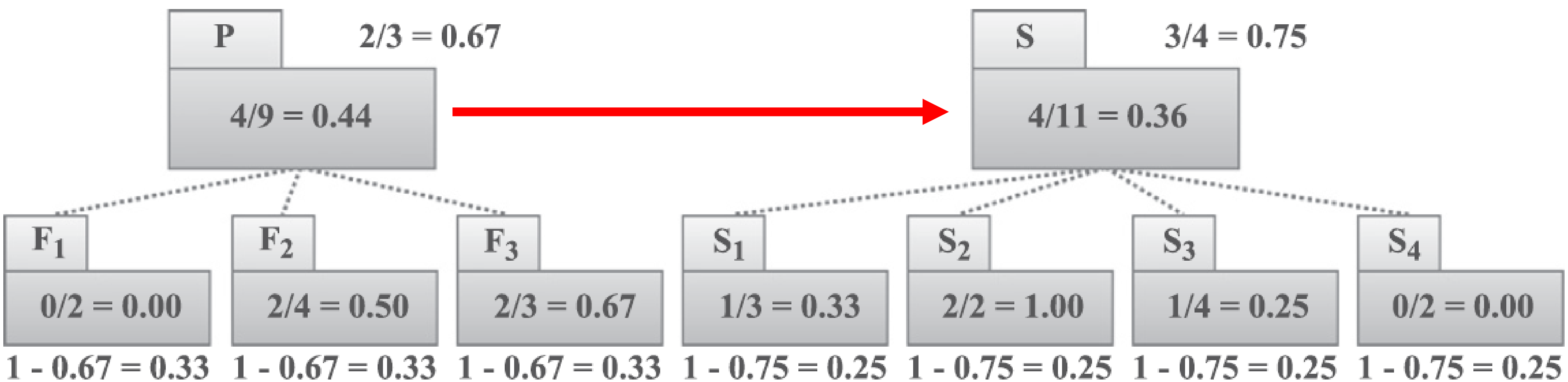
Parent-to-Parent	Parent-to-Child	Child-to-Parent	Child-to-Child
P → S	P → S1 P → S2 P → S3	F2 → S F3 → S	F2 → S1 F2 → S2 F2 → S3 F3 → S1 F3 → S2



Pairwise Clusterization of Dependencies

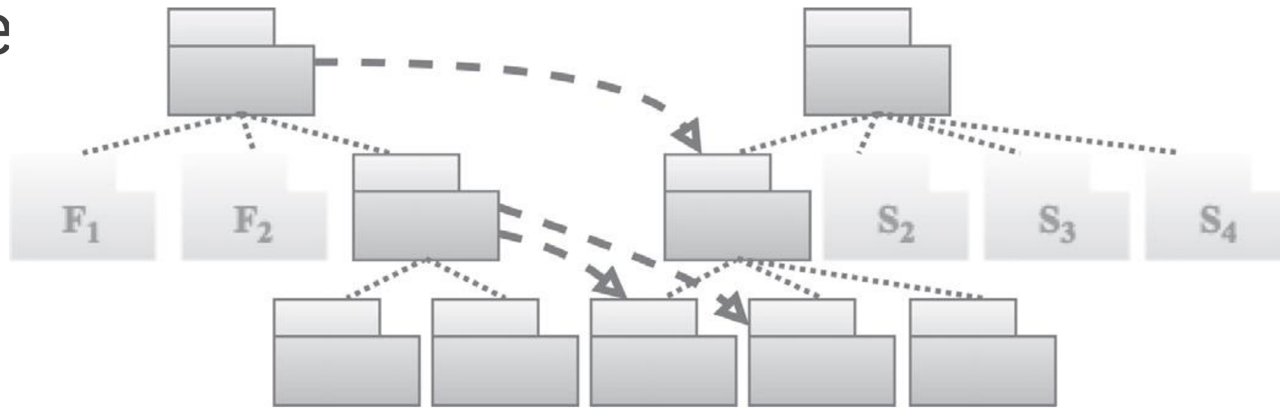
Parent-to-Parent	Parent-to-Child	Child-to-Parent	Child-to-Child
$P \rightarrow S(0.40)$	$P \rightarrow S_1(0.20)$	$F_2 \rightarrow S(0.31)$	$F_2 \rightarrow S_2(0.40)$
	$P \rightarrow S_2(0.43)$	$F_3 \rightarrow S(0.38)$	$F_2 \rightarrow S_3(0.25)$
	$P \rightarrow S_3(0.17)$		$F_3 \rightarrow S_1(0.33)$
			$F_3 \rightarrow S_2(0.43)$
Mean = 0.40	Mean = 0.27	Mean = 0.34	Mean = 0.35

Highest MDS Mean



Selection of Architectural Rules

- Package hierarchies might have redundant rule

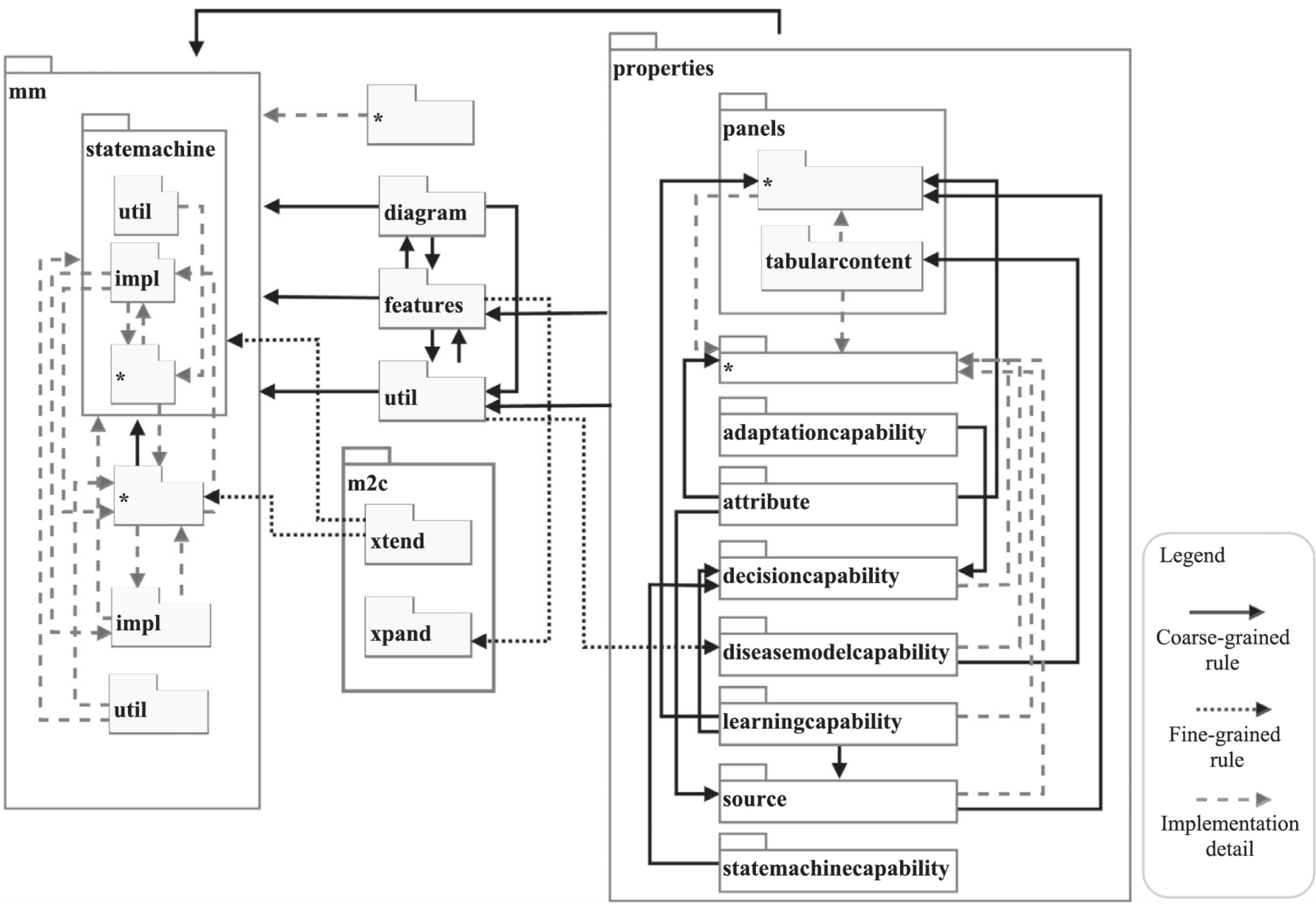


- Selection of the subset of non-redundant rules that m

$$\max \sum_{i=1}^{|D|} in_i \times (\omega(d_i) \times MDS(d_{i_S}, d_{i_T}))$$

$$in_x + in_y \leq 1, \forall\ d_x, d_y(redundant(d_x, d_y))$$

Case Study



mm
statemachine
util
impl
*
*
impl
util
*
diagram
features
util
m2c
xtend
xpand
properties
panels
*
tabularcontent
*
adaptationcapability
attribute
decisioncapability
diseasemodelcapability
learningcapability
source
statemachinecapability
Legend
Coarse-grained rule
Fine-grained rule
Implementation detail

Evaluation

- **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**

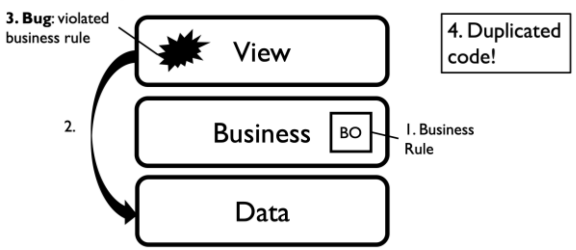
Challenges

- 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

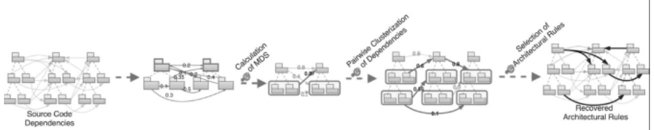
- Fundamental for the organised evolution of software systems



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

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

ZAPALOWSKI, VANIUS ; NUNES, I. ; NUNES, D. . The WGB method to recover implemented architectural rules. INFORMATION AND SOFTWARE TECHNOLOGY, v. 103, p. 125-137, 2018.

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.

.inf INSTITUTO DE INFORMÁTICA UFRGS

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

Thank you!

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

Vanius Zapalowski

Daltro Nunes