Continuous Detection of Code Anomalies: Synthesis of Code Anomalies

Towards Revealing Design Problems in Source Code

Alessandro Garcia – afgarcia@inf.puc-rio.br

Willian Oizumi – woizumi@inf.puc-rio.br

LES | DI |PUC-Rio - Brazil

OPUS Research Group

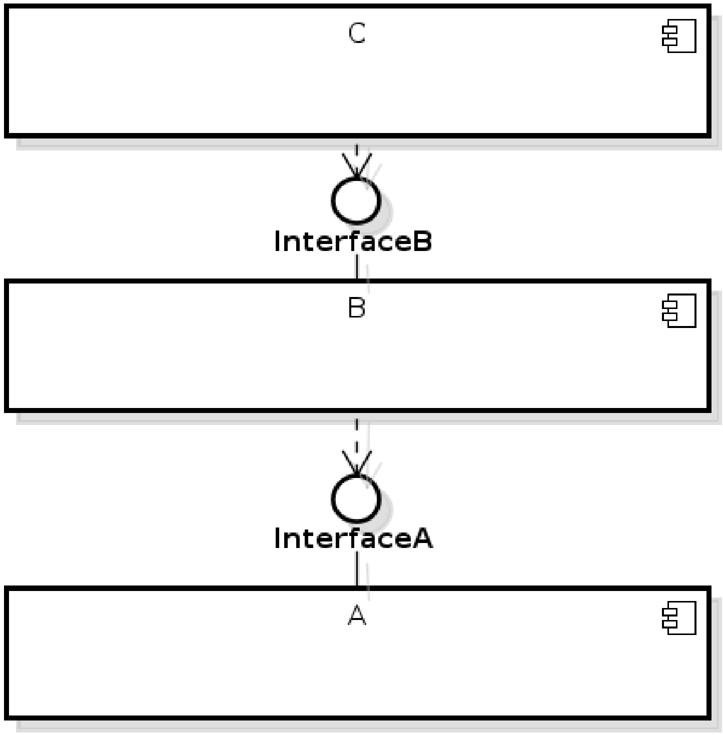
Challenges

- Continuous Anomaly Detection
    - How to reduce information overload to developers?
    - How to inform “meaningful” anomalies in the source code?
    - How to accurately report all the information they need?

- A first step is to synthesize code anomalies that represent (more critical) design problems to developers

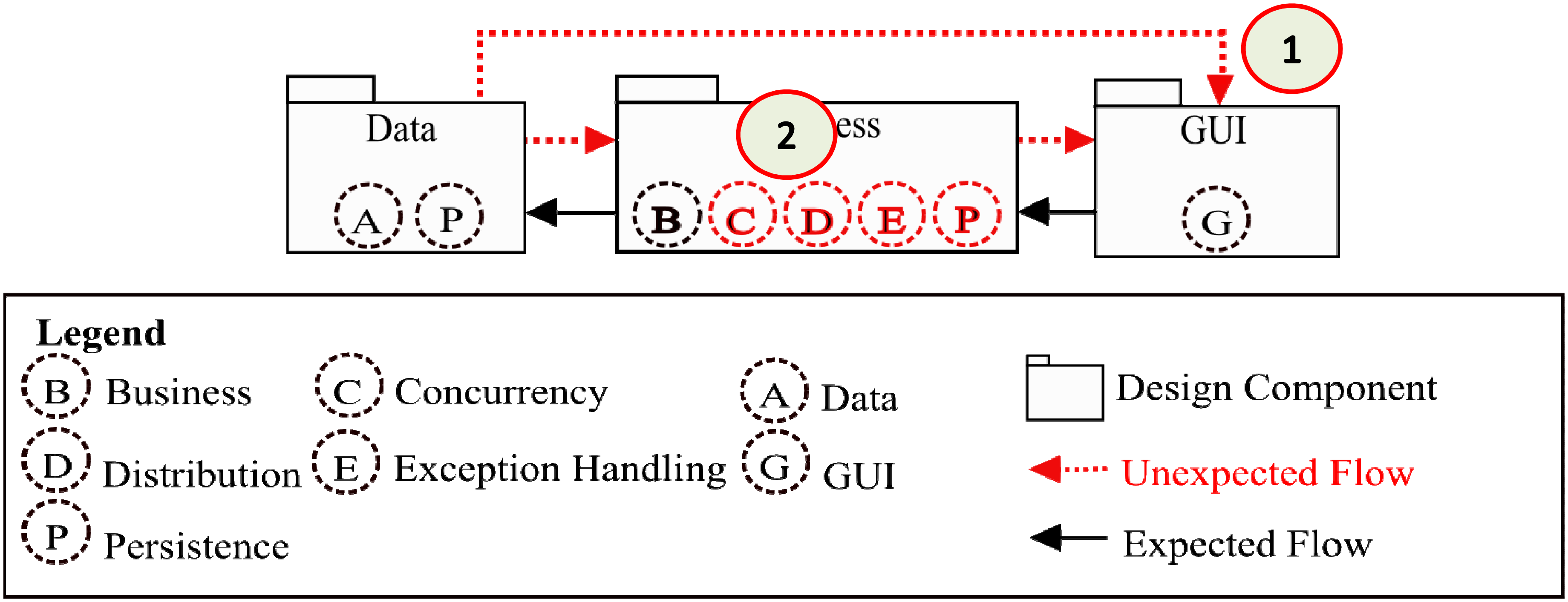
Software Design

- Software design represents the overall organization of the system into design components, interfaces and relationships among them (Bass *et al.* 2003)



Software Design Problem

- Design decision that either violates:
  1) Intended Design, or
  2) Modularity Principle



Why should I Care about Design Problems?

- When design problems are allowed to persist in a system:

- It may have to be **fundamentally reengineered** (Godfrey 2000; Gurp 2002; Schach 2002)
- It may even be **discontinued** (MacCormack 2006)

How to Identify Design Problems?

- Design **documentation** is often **informal** or **nonexistence**

- Therefore, **many developers** have to rely on **source code analysis**

Code Anomalies

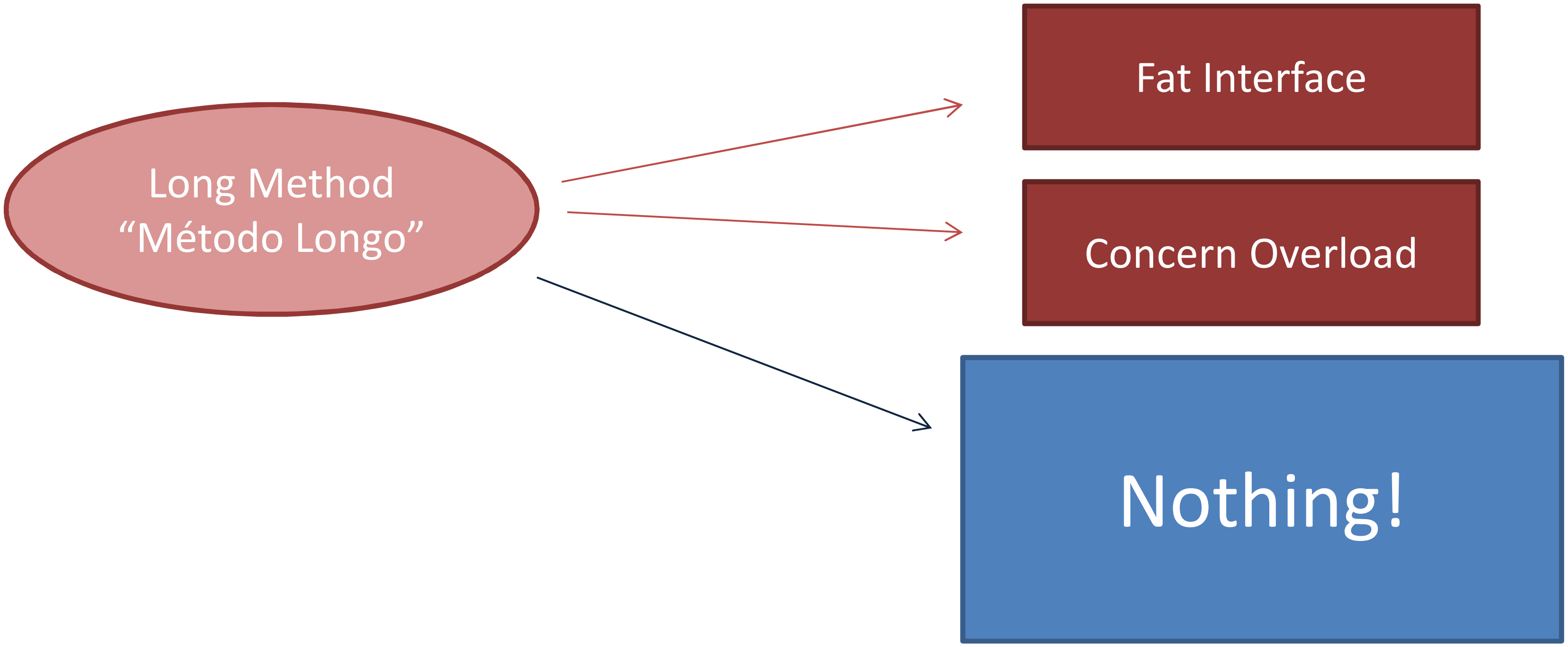
- A code anomaly is a **symptom** of a **bad decision**, such as a **design problem**, observed in a program's low-level structure (Fowler 1999; Lanza & Marinescu 2006)

- Different **techniques** for **code anomaly detection** have been **proposed** and **studied** (Emden & Moonen 2002; Lanza & Marinescu 2006; Wong *et al.* 2011)

- However, a **high proportion** of them **may not help** programmers to identify **design problems**

Limitations of Code Anomalies

- We observed that there is **no direct relation** between specific types of **Anomalies** and **Design Problems**
- Example:

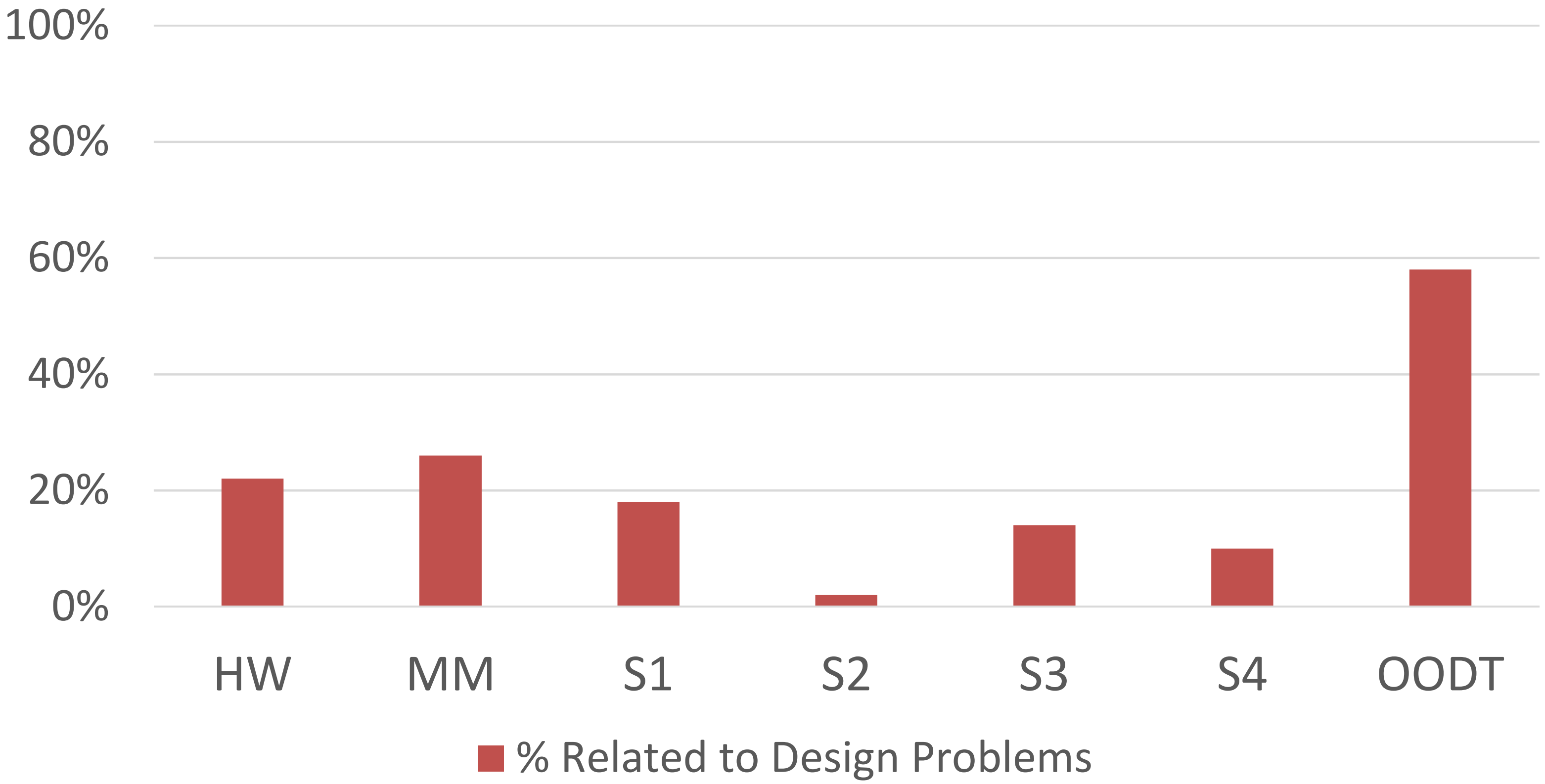


Detection of Code Anomalies

- The **impact** of **code anomalies** has been **largely studied** *(Khomh et al. 2009; Kim et al. 2005; Lozano & Wermelinger 2008; Olbrich et al. 2010; D'Ambros et al. 2010; Sjobert et al. 2013; Macia 2013)*

- However, existing **techniques** and **tools** for code anomaly detection *(Emden & Moonen 2002; Ratzinger et al. 2005; Wong et al. 2011; Marinescu 2004)* are **not enough** to help developers in the **identification** of **design problems**

Code Anomalies

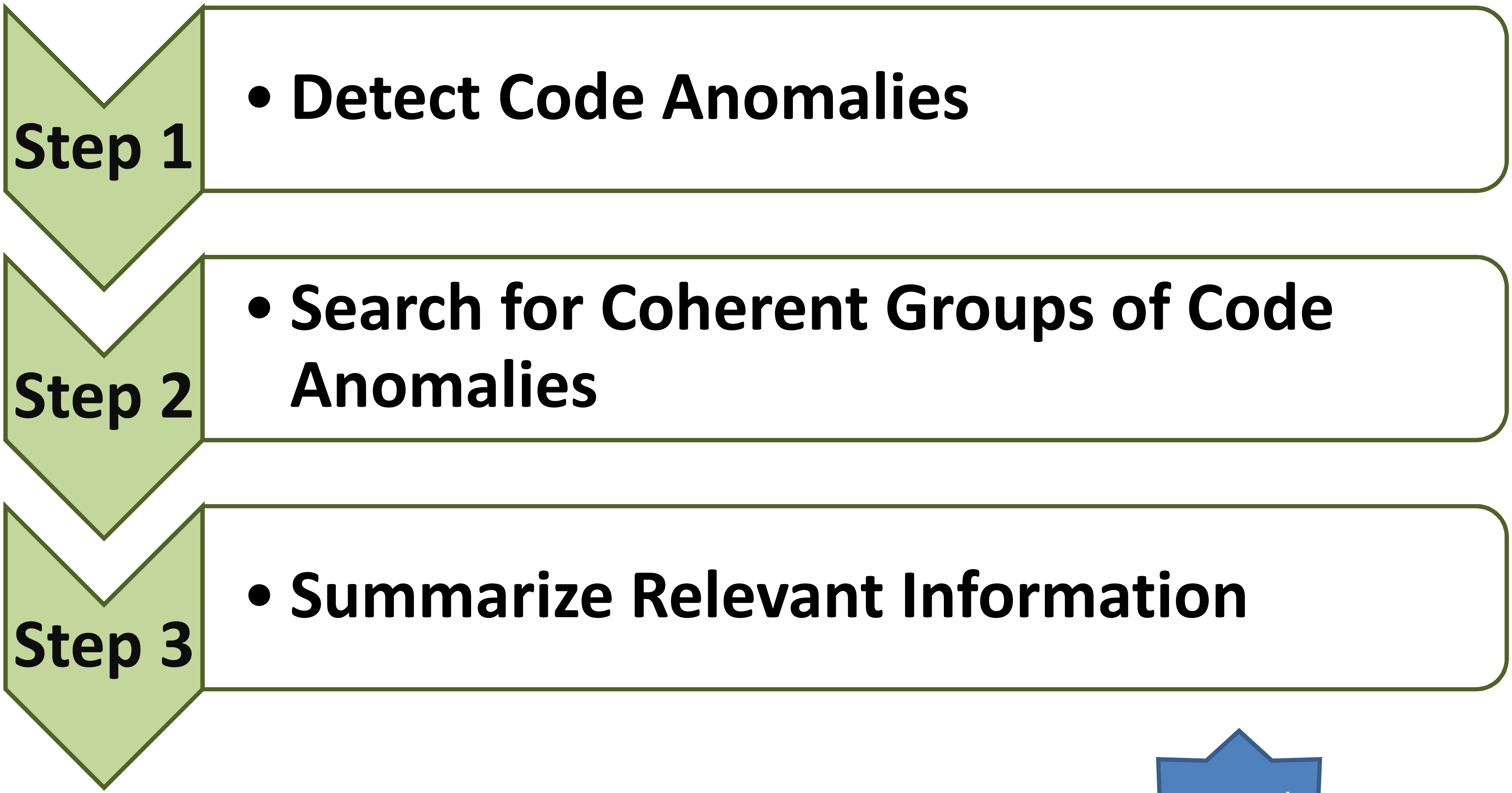


A considerable proportion of code anomalies do not represent design problems (Oizumi *et al.* 2014)

SBES 2014

How to provide better support?

Synthesis of Code Anomalies



Step 1

- **Detect Code Anomalies**

Step 2

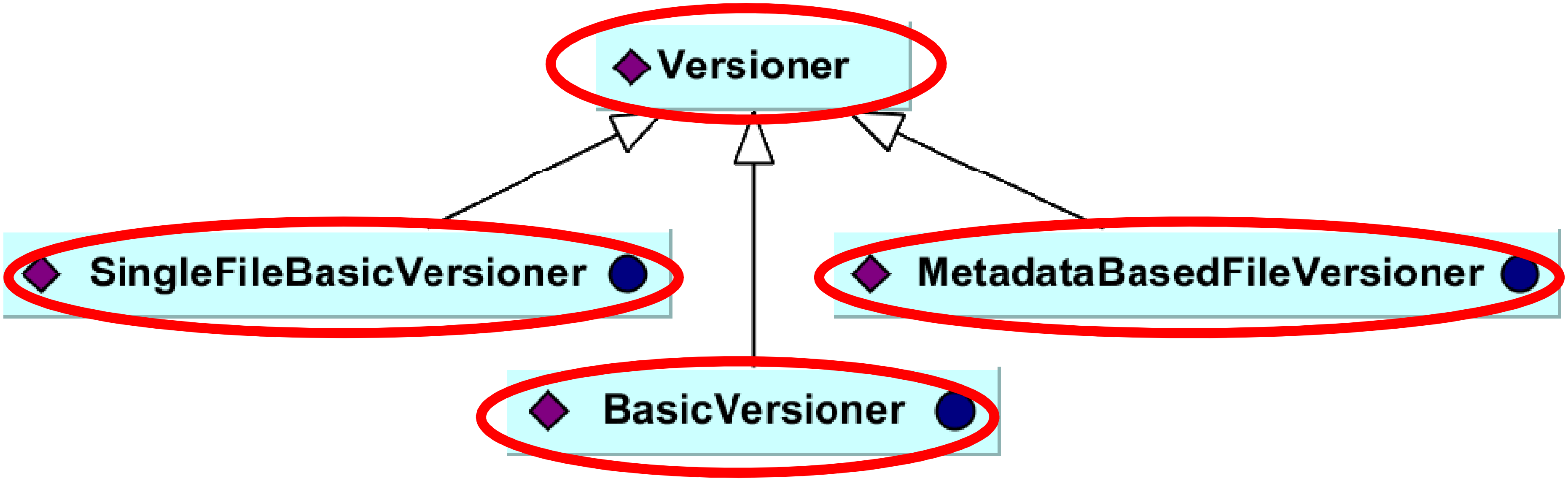
- **Search for Coherent Groups of Code Anomalies**

Step 3

- **Summarize Relevant Information**

WMod 2014

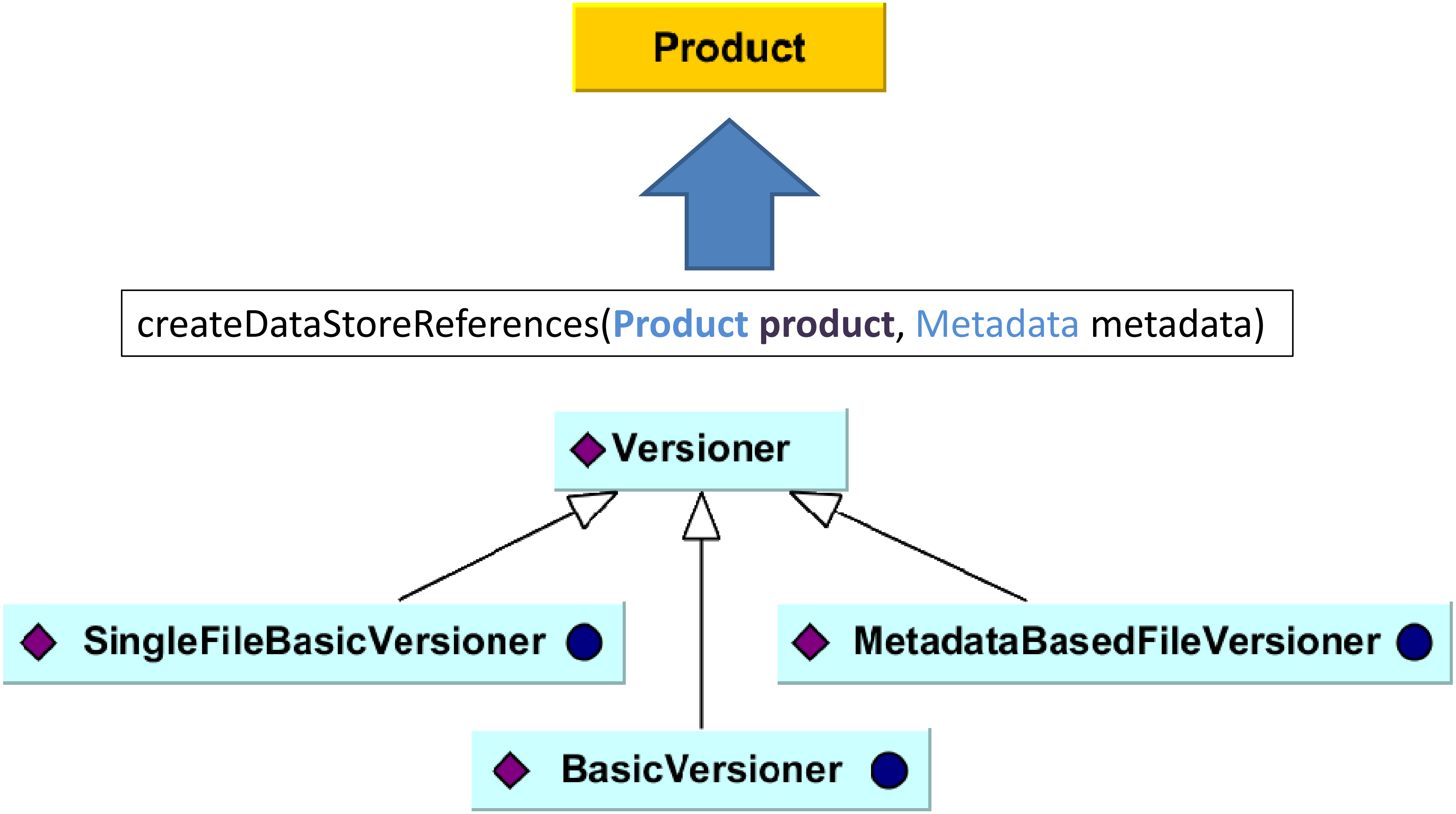
Example of Design Problem



◆ **Fat Interface:**

Interface incorporates too many operations on some data into an interface, only to find that most of the objects cannot perform the given operations.

Fat Interface



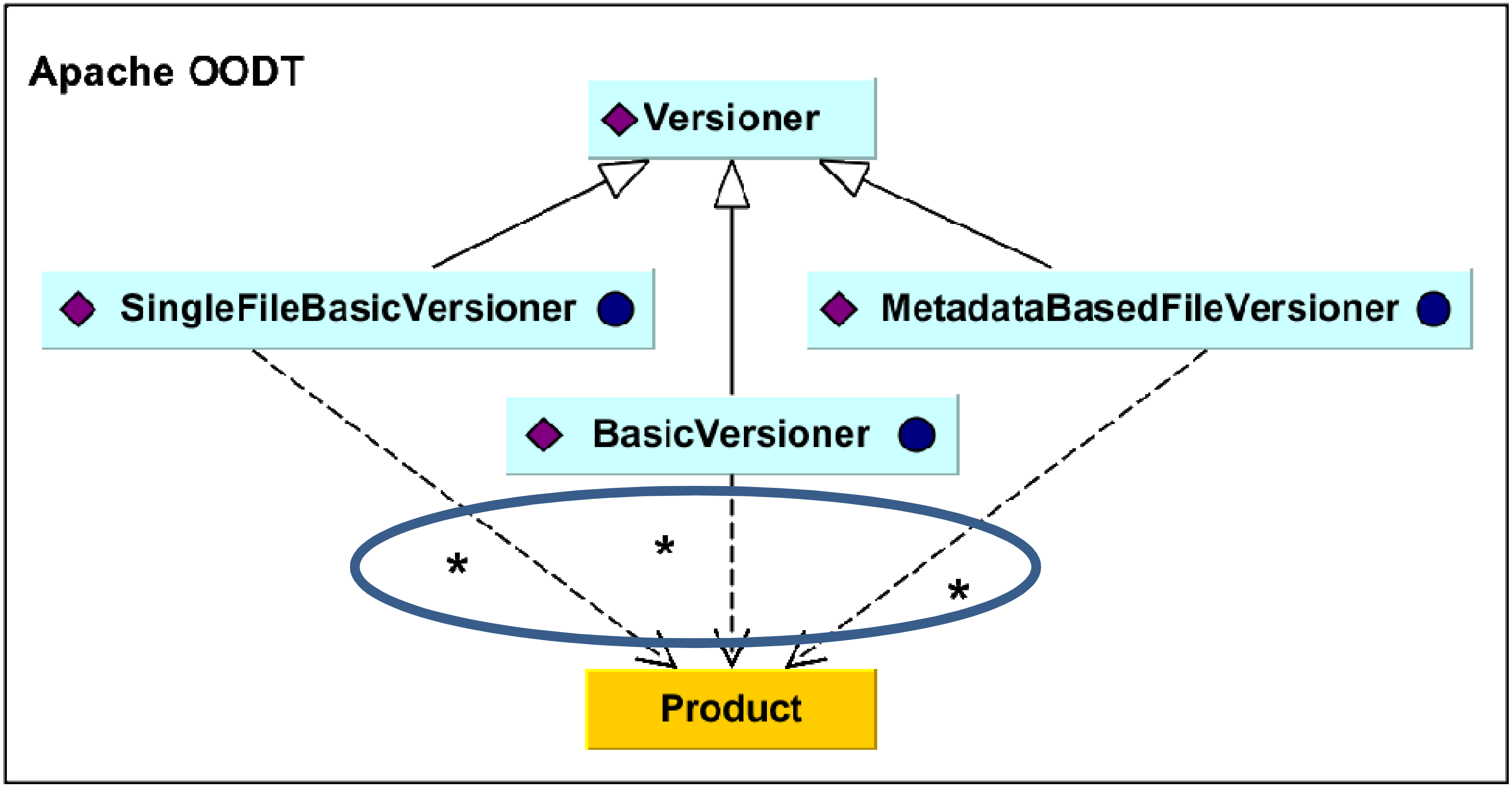
createDataStoreReferences(**Product product**, Metadata metadata)

Code Anomaly Detection

Step 1

- Detection of code anomalies using detection strategies (Marinescu 2004)
- Detection strategies based on source code metrics

Detection of Feature Envies



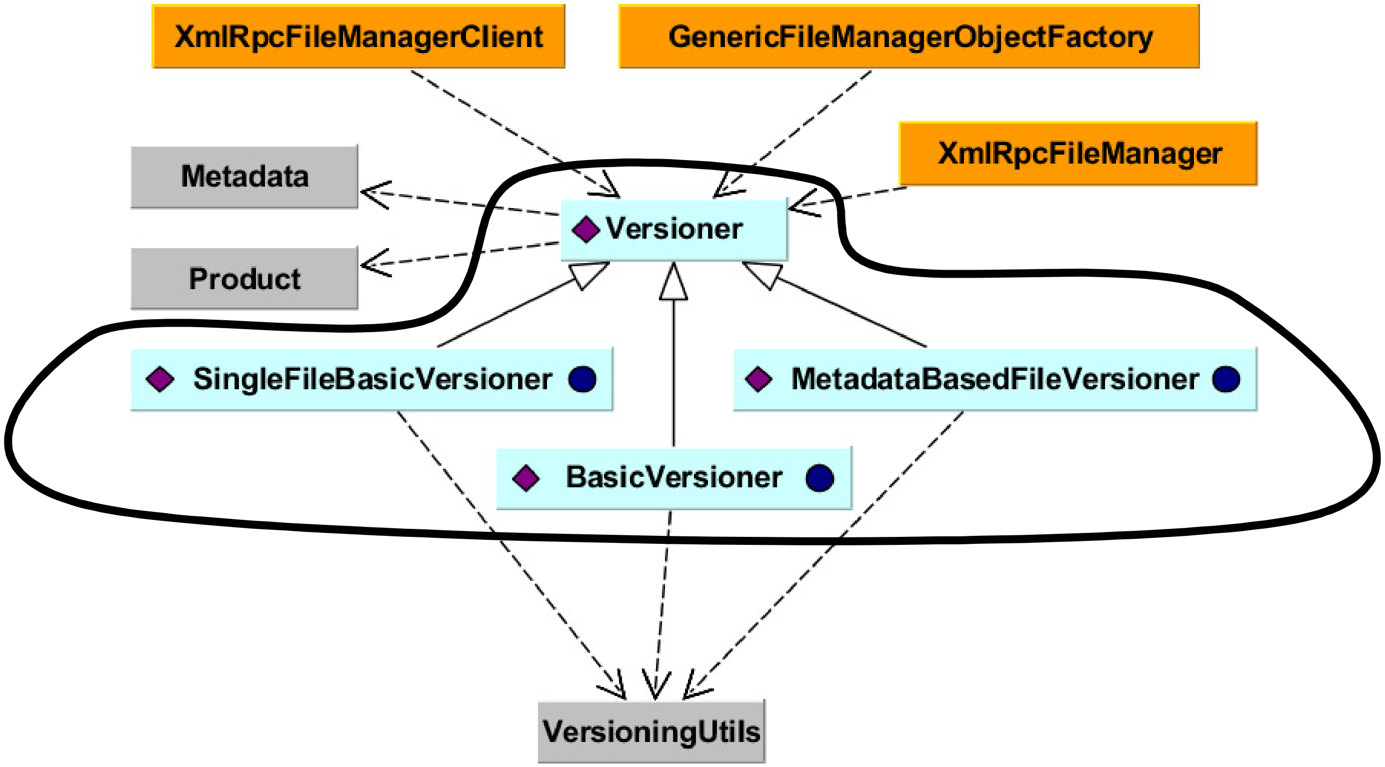
Example of Feature Envy

◆ BasicVersioner ●

```
public void createDataStoreReferences(Product product, Metadata metadata)
throws VersioningException {
String productName = product.getProductName();
String productRepoPath = product.getProductType().getProductRepositoryPath();
...
if (product.getProductStructure().equals(Product.STRUCTURE_HIERARCHICAL)) {
        if (product.getProductReferences() == null
        || (product.getProductReferences() != null &&
        product.getProductReferences().size() == 0)) {
...
} else if (product.getProductStructure().equals(Product.STRUCTURE_FLAT)) {
...
} else {
...
```

Information Scattered in the Source Code

- Information about the design problem is often scattered in several code elements



Code Anomaly Detection

Step 1

- Techniques for code anomaly detection do not explore relationships between anomalies
- However, design problems are often scattered in the source code
- Therefore, they are not enough to help developers diagnosing design problems

Grouping of Code Anomalies

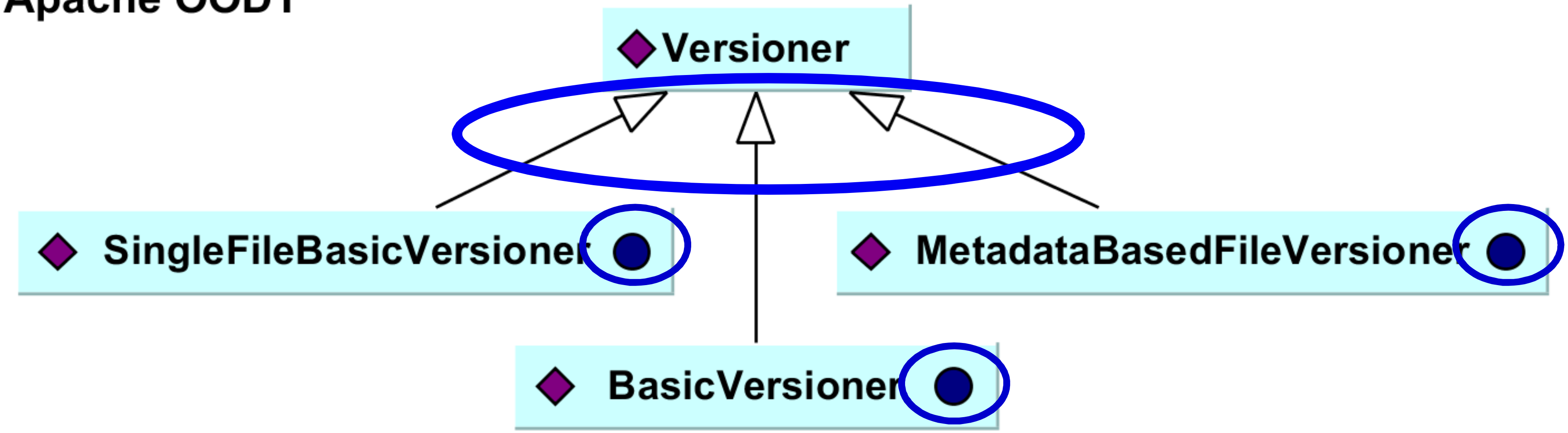
Step 2

- After detecting code anomalies, the synthesis technique uses different **topologies** to **search** for **agglomerations**

- A **code anomaly-agglomeration** is a coherent group of **code anomalies** that may **contribute** to the realization of a **design problem**

Example: Hierarchical Topology

Apache OODT



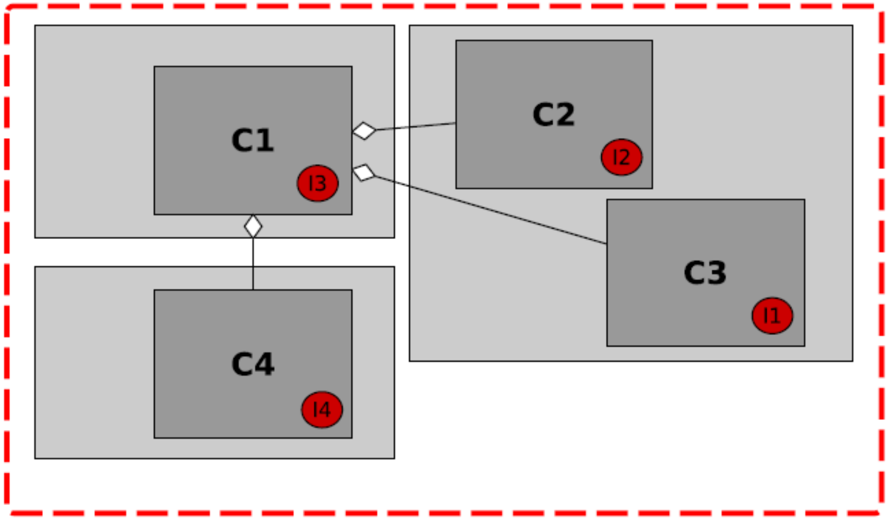
Code anomalies related through hierarchical relationships

Grouping with Hierarchical Topology

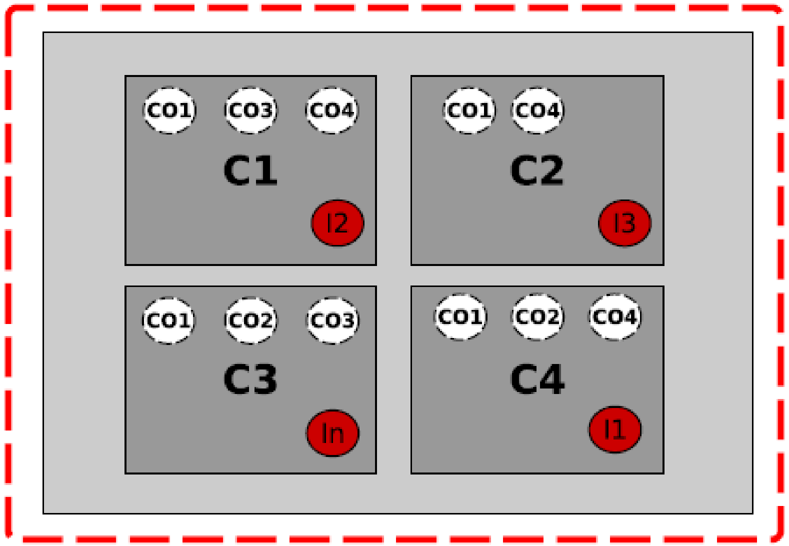
- Code Anomalies of the same type (e.g. Feature Envy)
- Occurring in the same hierarchy
  - Inheritance tree
  - Interface Implementation
- Satisfying a threshold

Examples of Other Topologies

- Cross-component



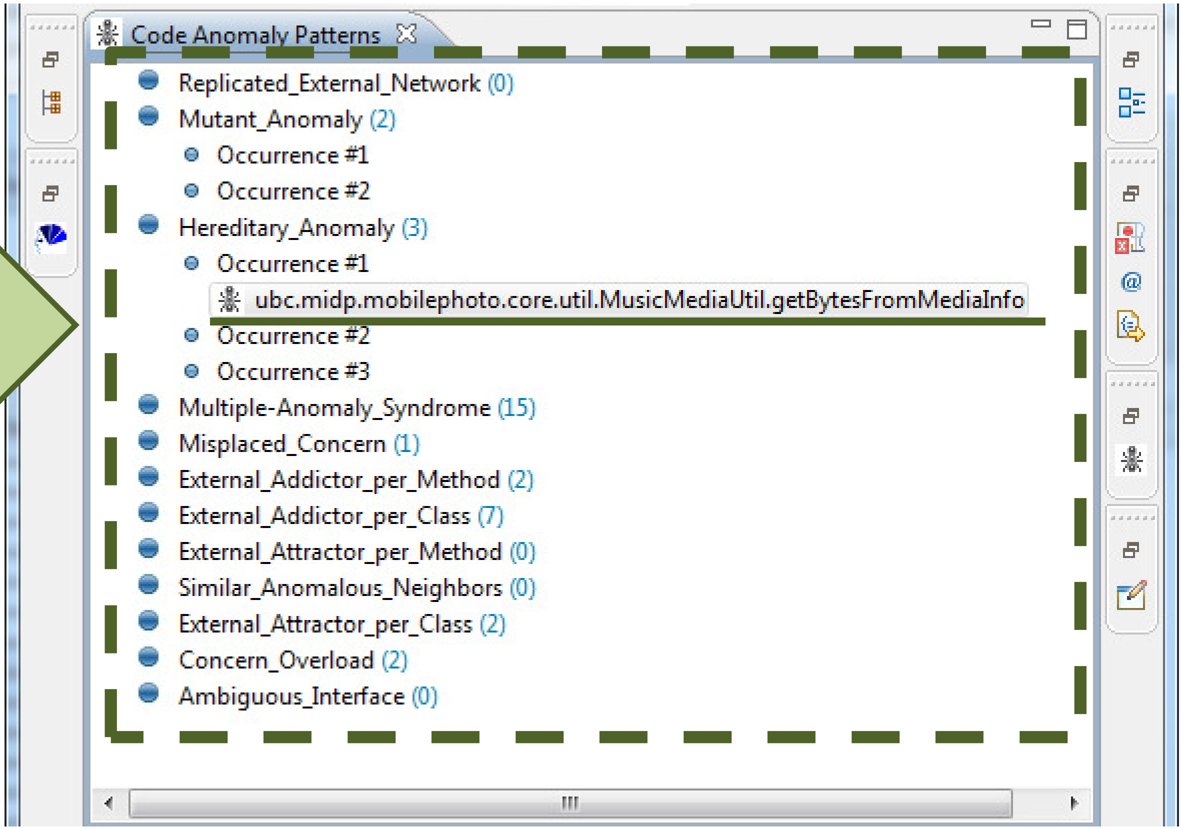
- Concern-based



Summarization of Relevant Information

Step 3

- Existing techniques provide few information about each code anomaly



Code Anomaly Patterns

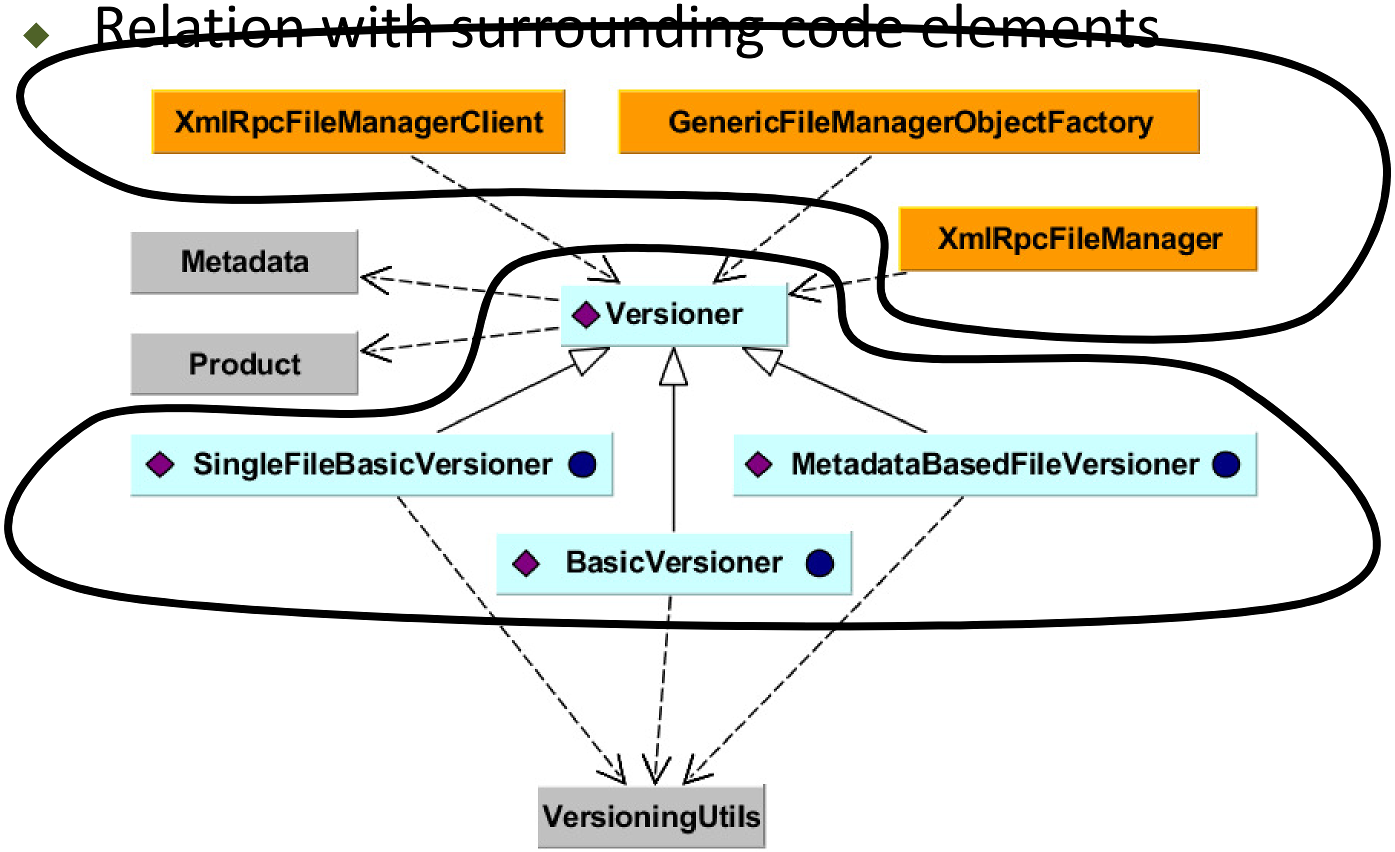
- Replicated_External_Network (0)
- Mutant_Anomaly (2)
  - Occurrence #1
  - Occurrence #2
- Hereditary_Anomaly (3)
  - Occurrence #1
    - ubc.midp.mobilephoto.core.util.MusicMediaUtil.getBytesFromMediaInfo
  - Occurrence #2
  - Occurrence #3
- Multiple-Anomaly_Syndrome (15)
- Misplaced_Concern (1)
- External_Addictor_per_Method (2)
- External_Addictor_per_Class (7)
- External_Attractor_per_Method (0)
- Similar_Anomalous_Neighbors (0)
- External_Attractor_per_Class (2)
- Concern_Overload (2)
- Ambiguous_Interface (0)

Few information about each group

Contextual Information

Step 3

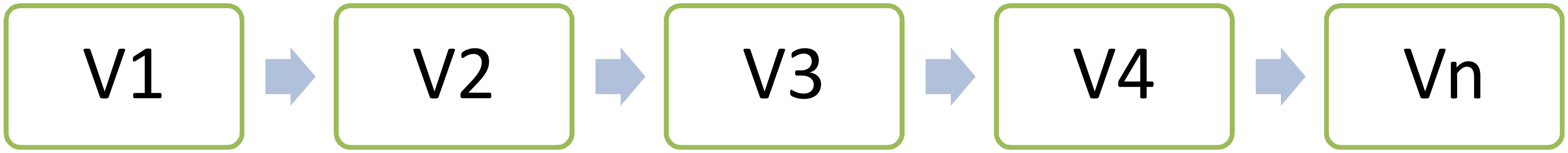
- We provide contextual information about each group of code anomalies
  - Relation with surrounding code elements



History Information

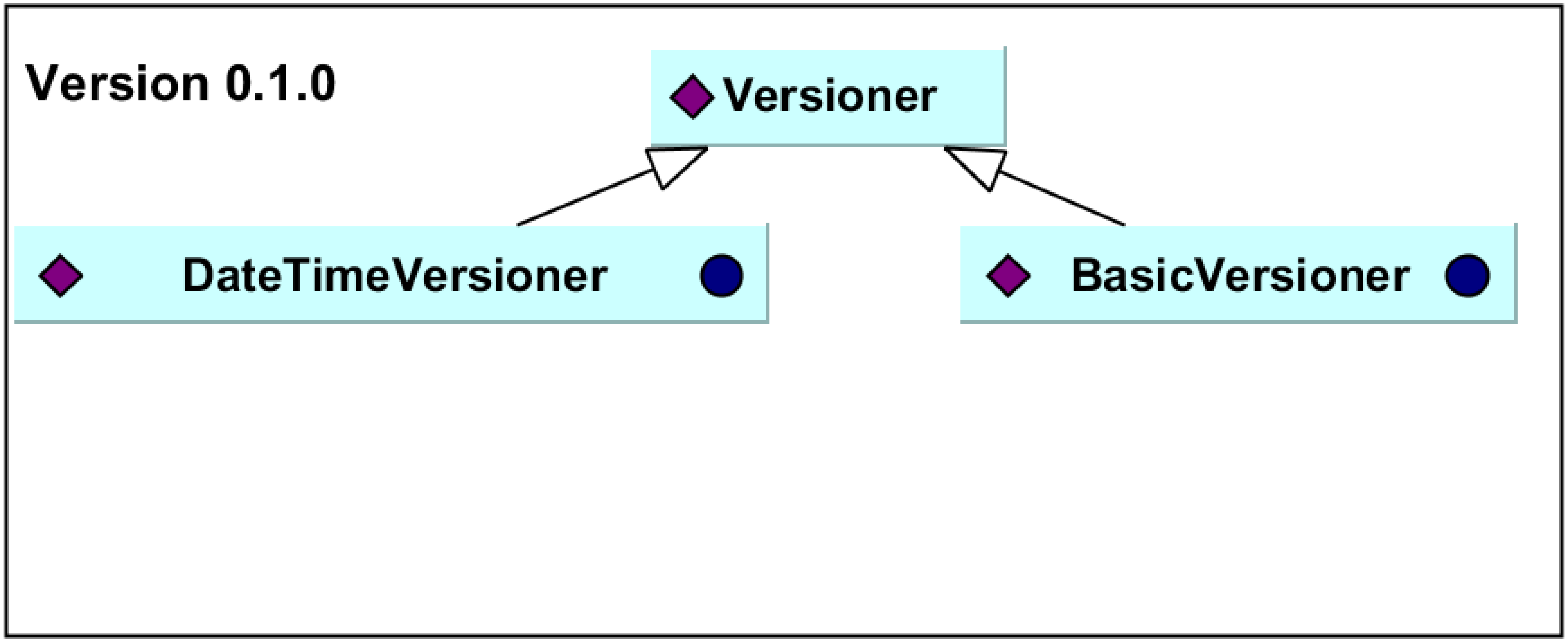
Step 3

- Providing history information about groups of anomalies:

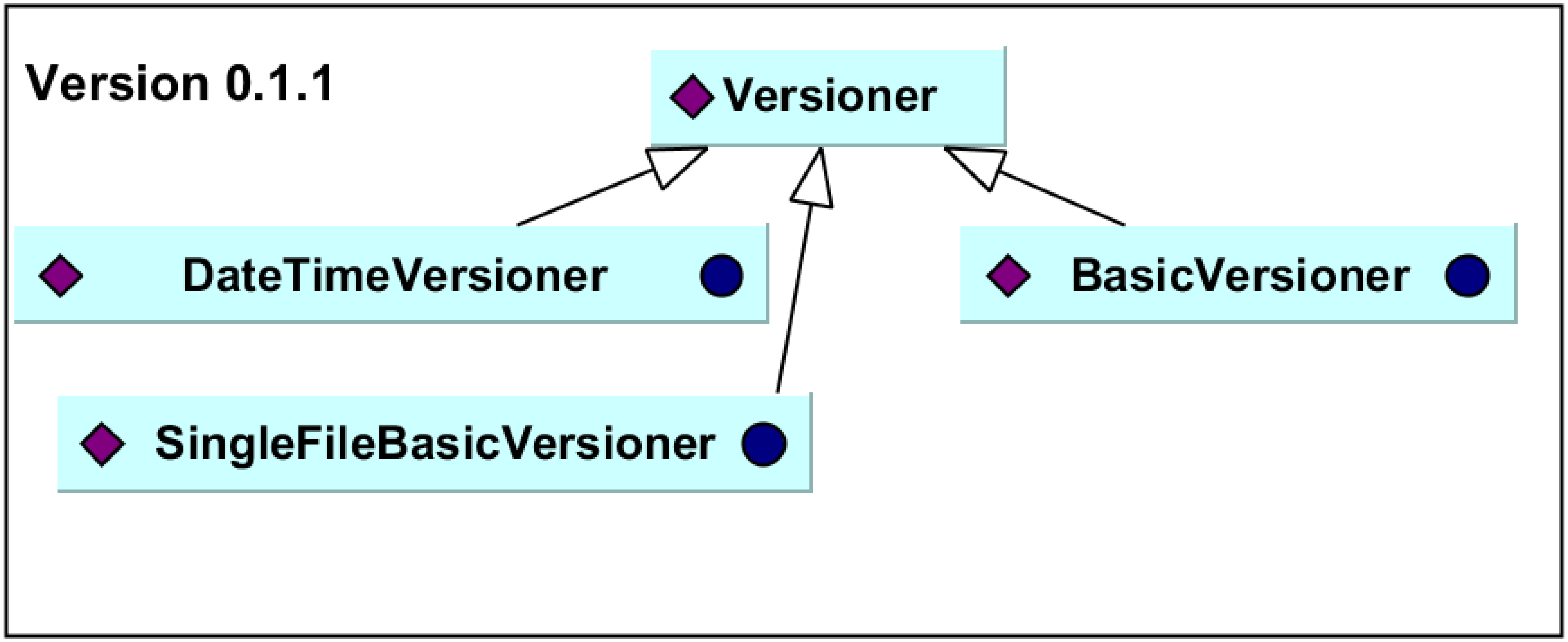


V1 → V2 → V3 → V4 → Vn

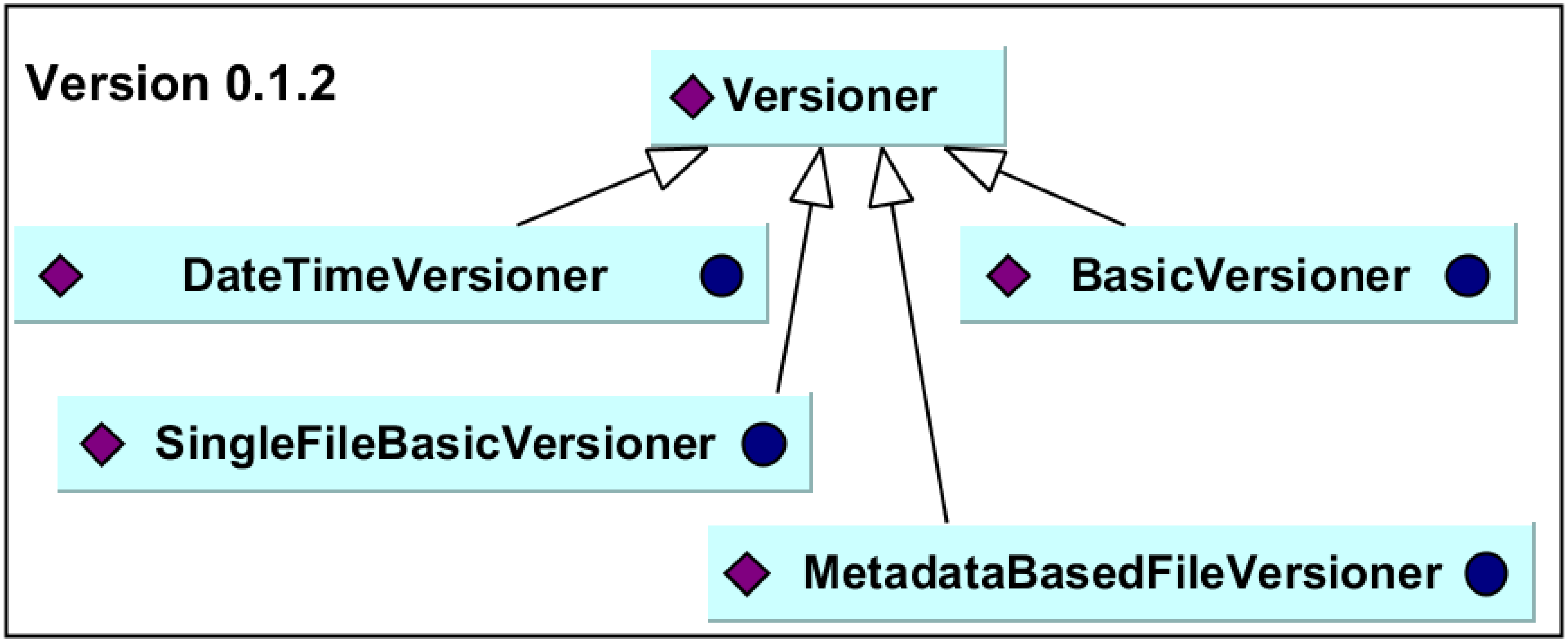
Growing Problem in OODT



Growing Problem in OODT

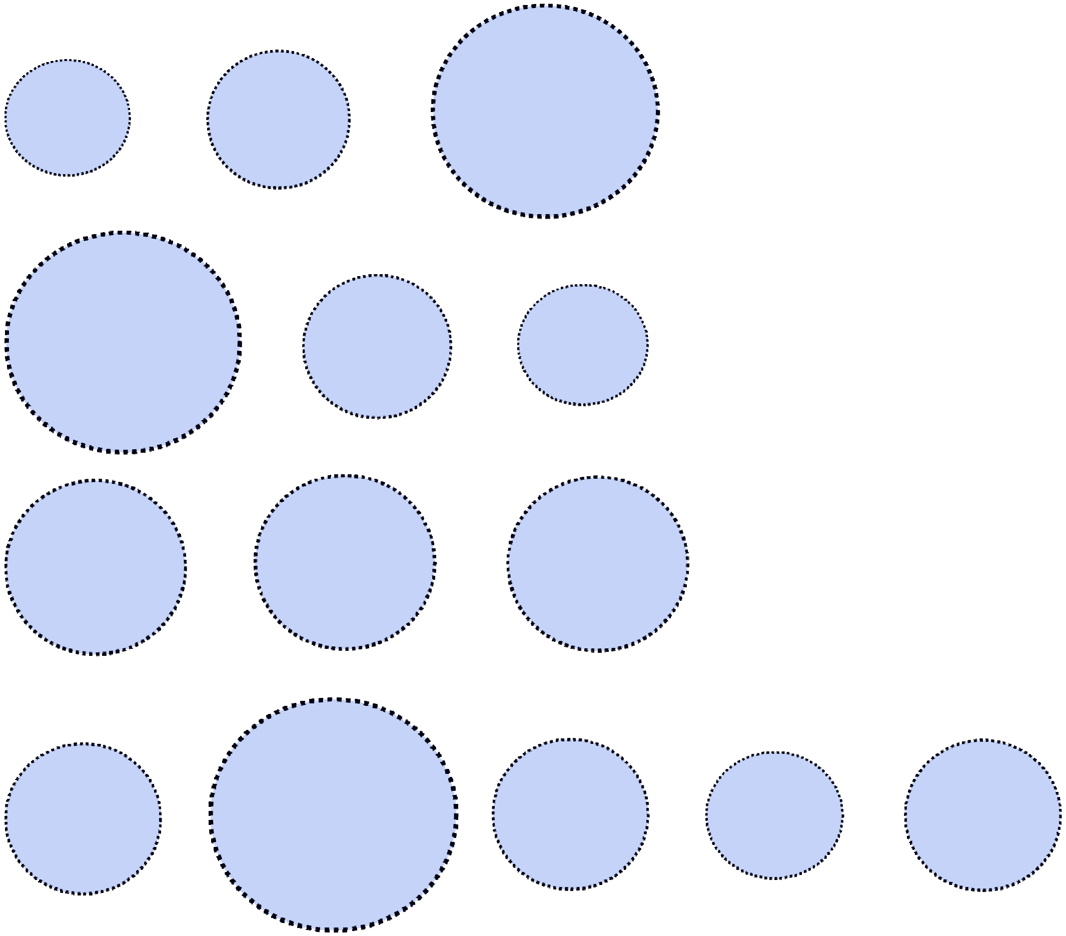


Growing Problem in OODT

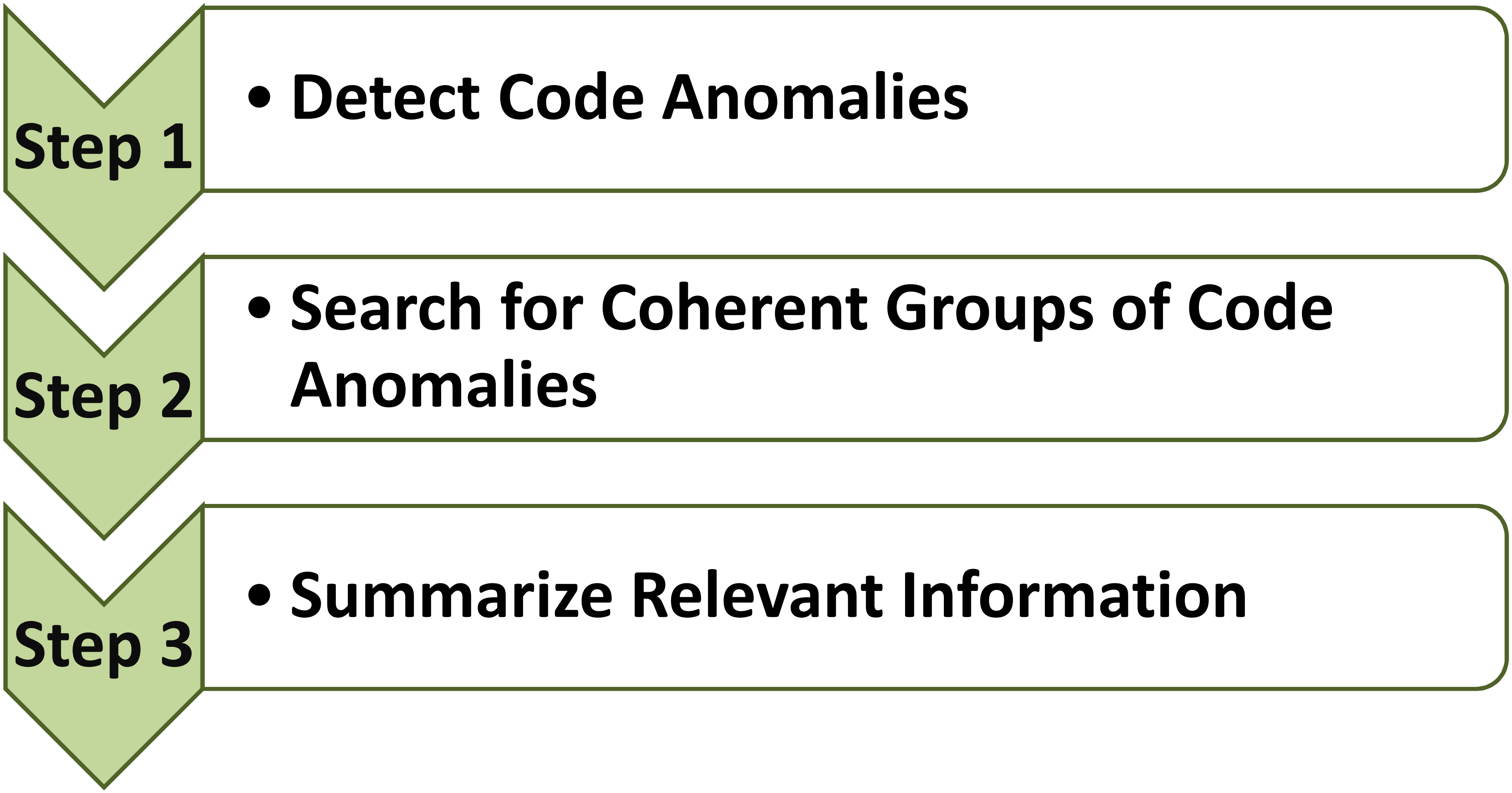


History Information

- Allows developers to identify different changing patterns:
  - Growing
  - Shrinking
  - Idle
  - Waving



Synthesis of Code Anomalies



Step 1

- Detect Code Anomalies

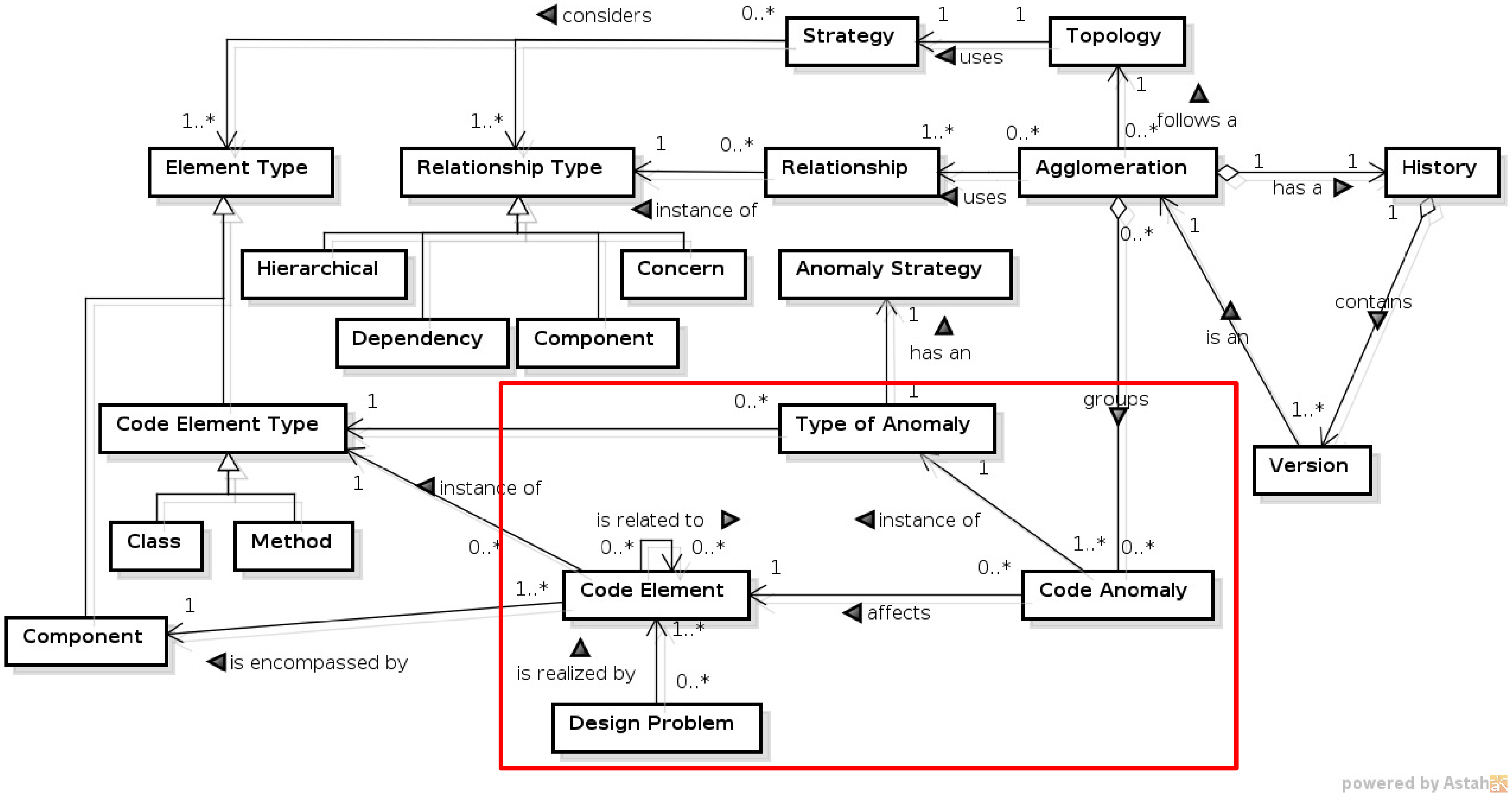
Step 2

- Search for Coherent Groups of Code Anomalies

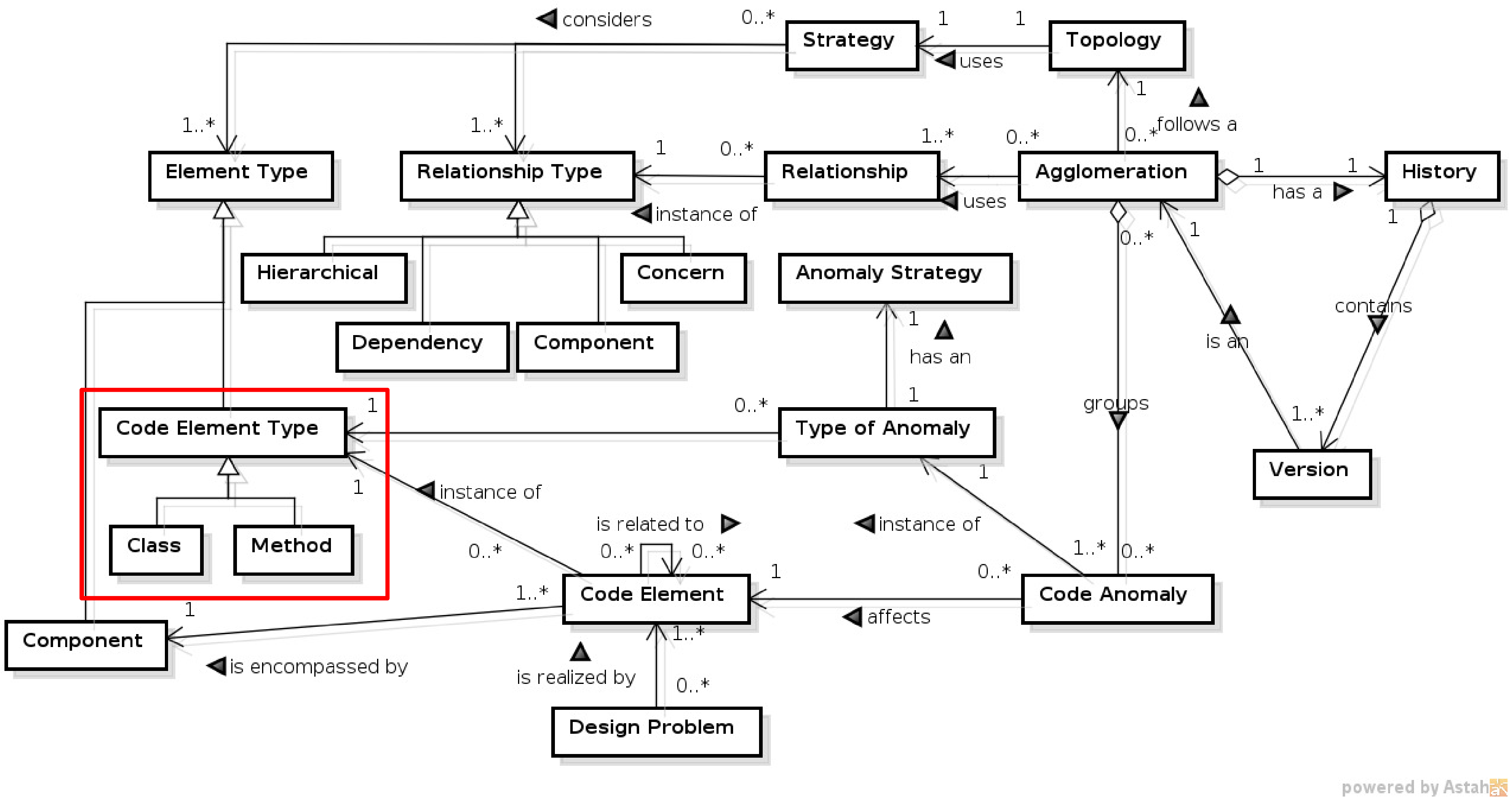
Step 3

- Summarize Relevant Information

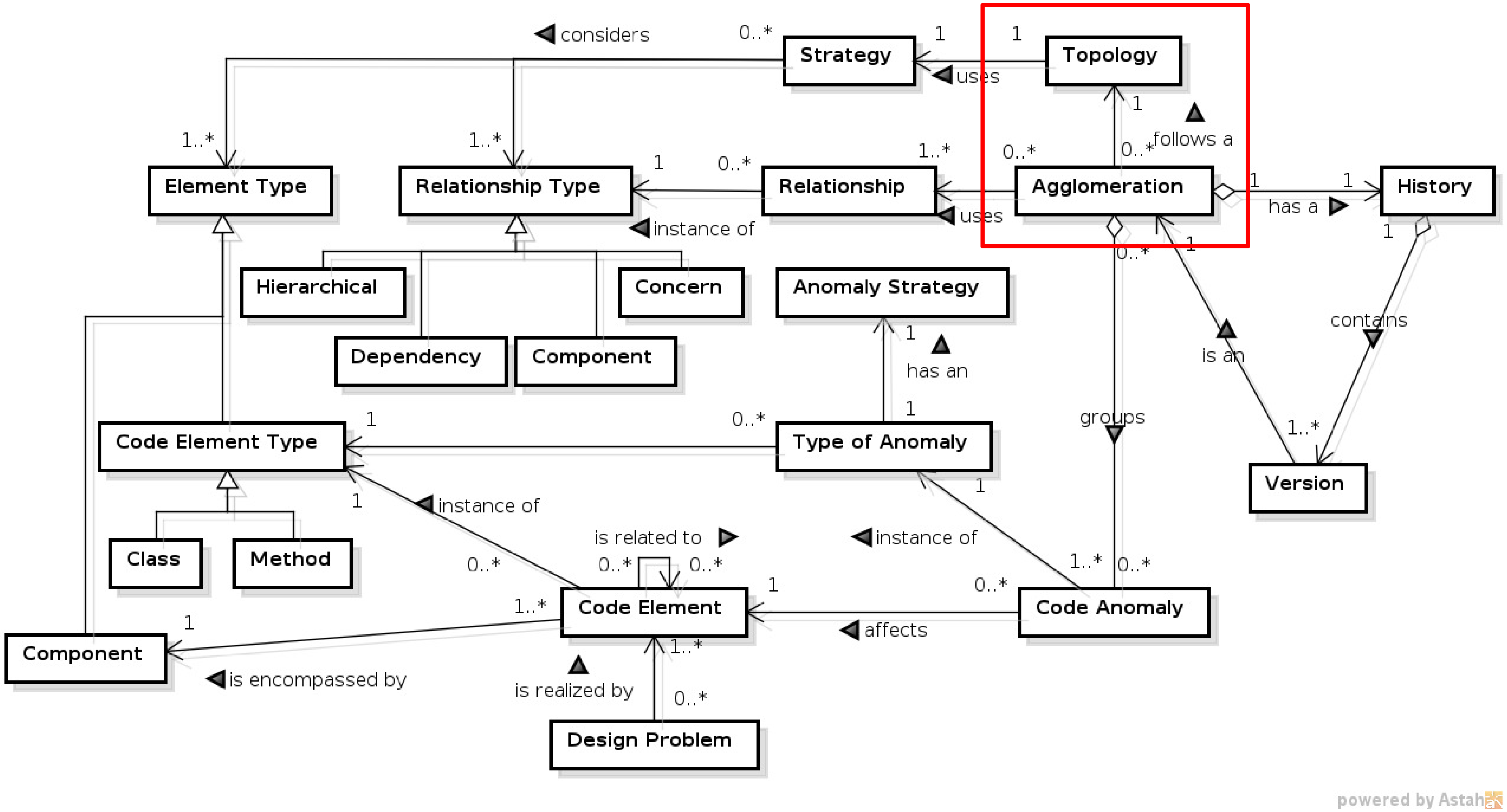
Synthesis Technique



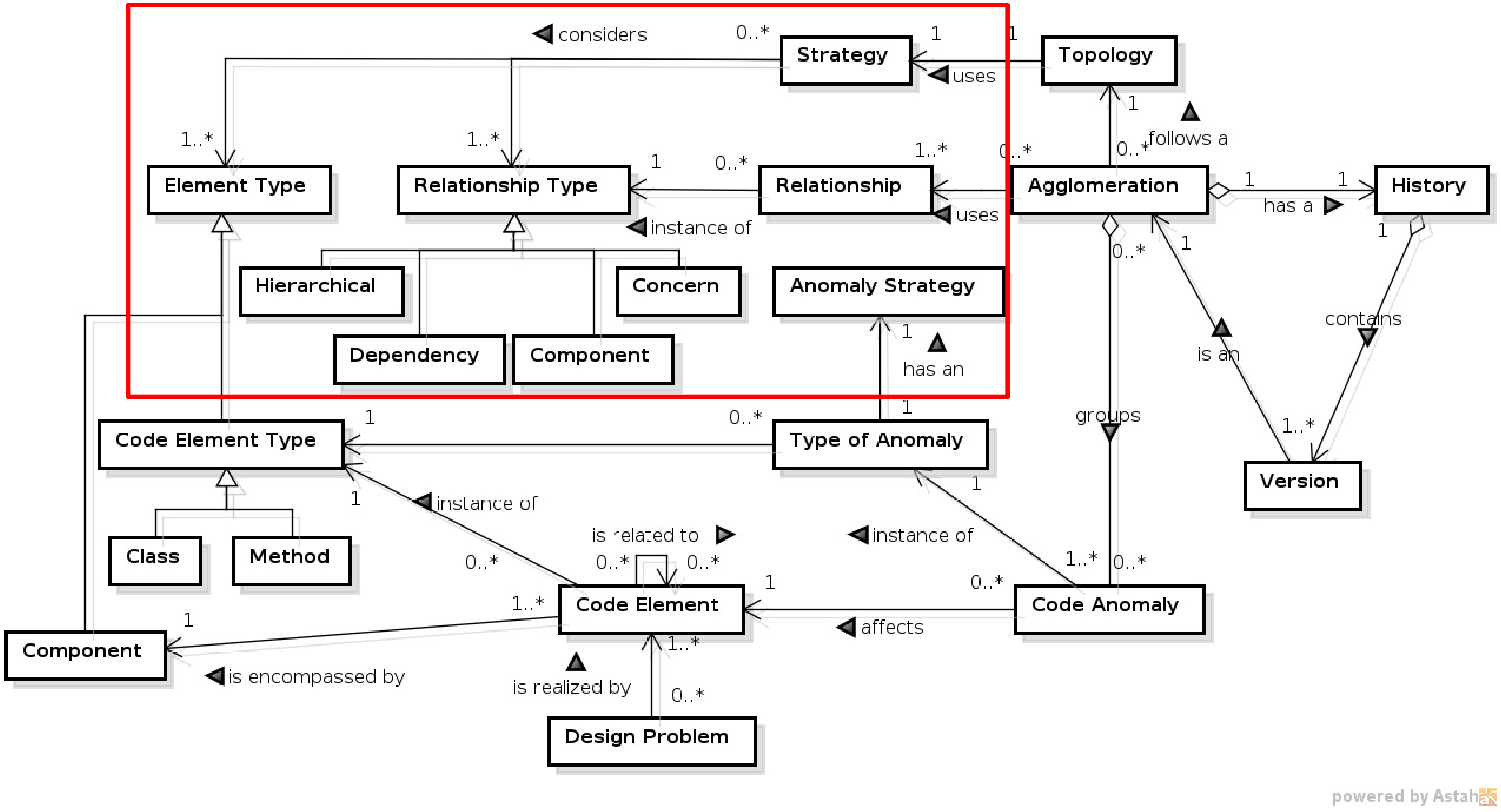
Synthesis Technique



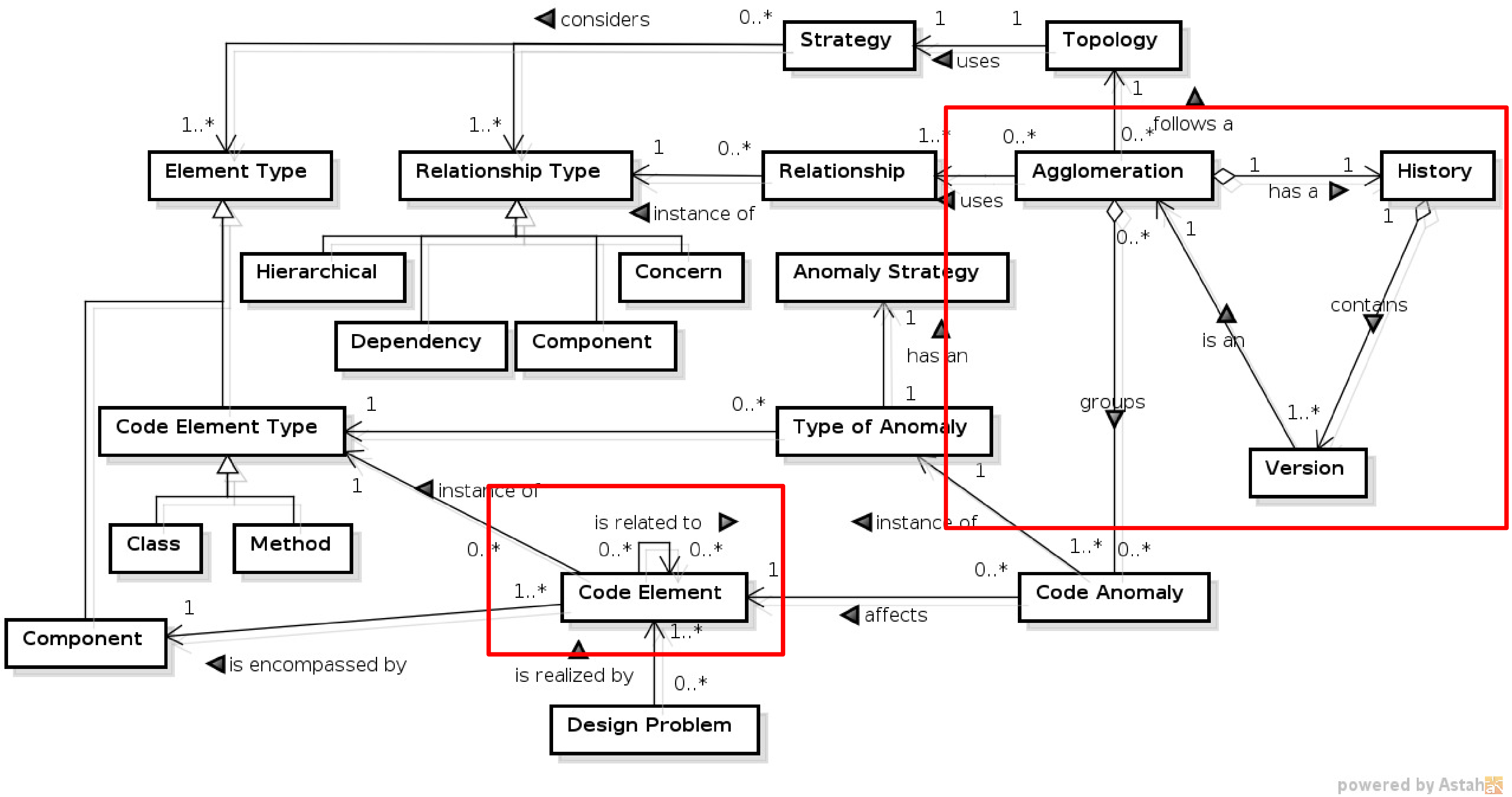
Synthesis Technique



Synthesis Technique



Synthesis Technique



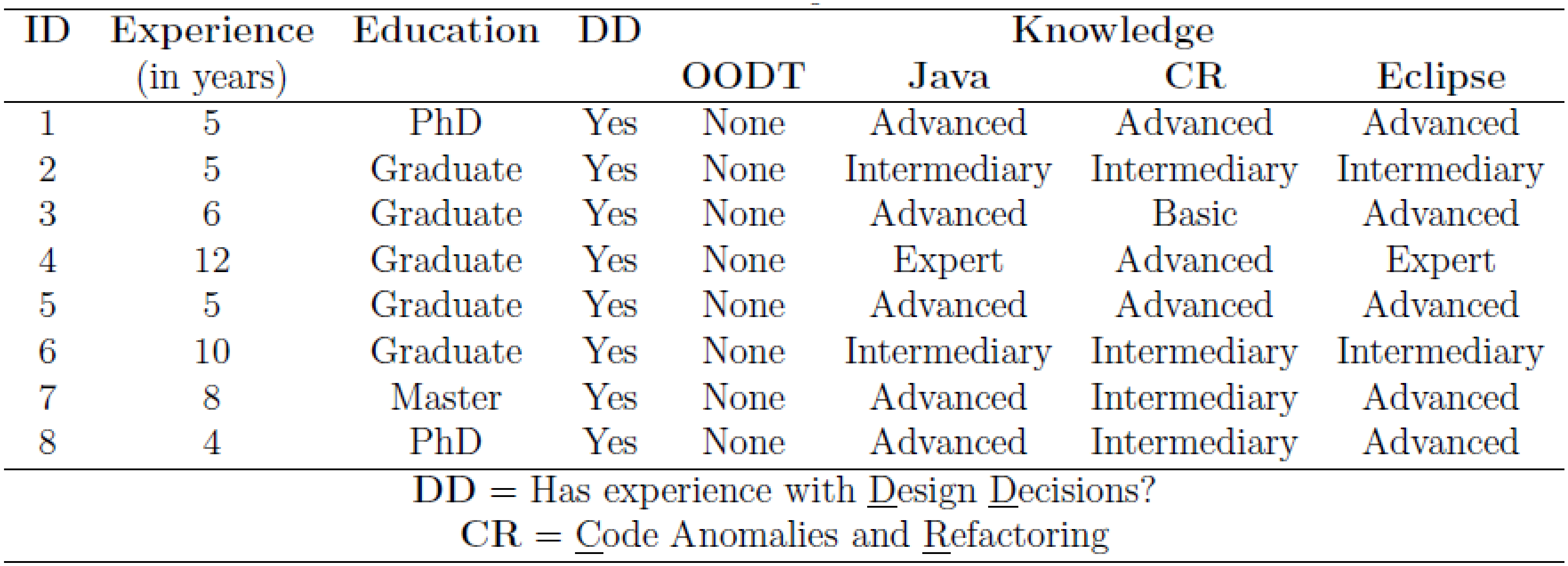
Evaluation

- **RQ1:** Which is the most accurate technique regarding the identification of design problems?
  - Synthesis or Conventional?

- **RQ2:** What are the most useful agglomeration topologies?

Evaluation

- We conducted two empirical studies:
  - Multi-case study with 7 systems
  - Quasi-experiment with 6 industry professionals and 2 PhD students



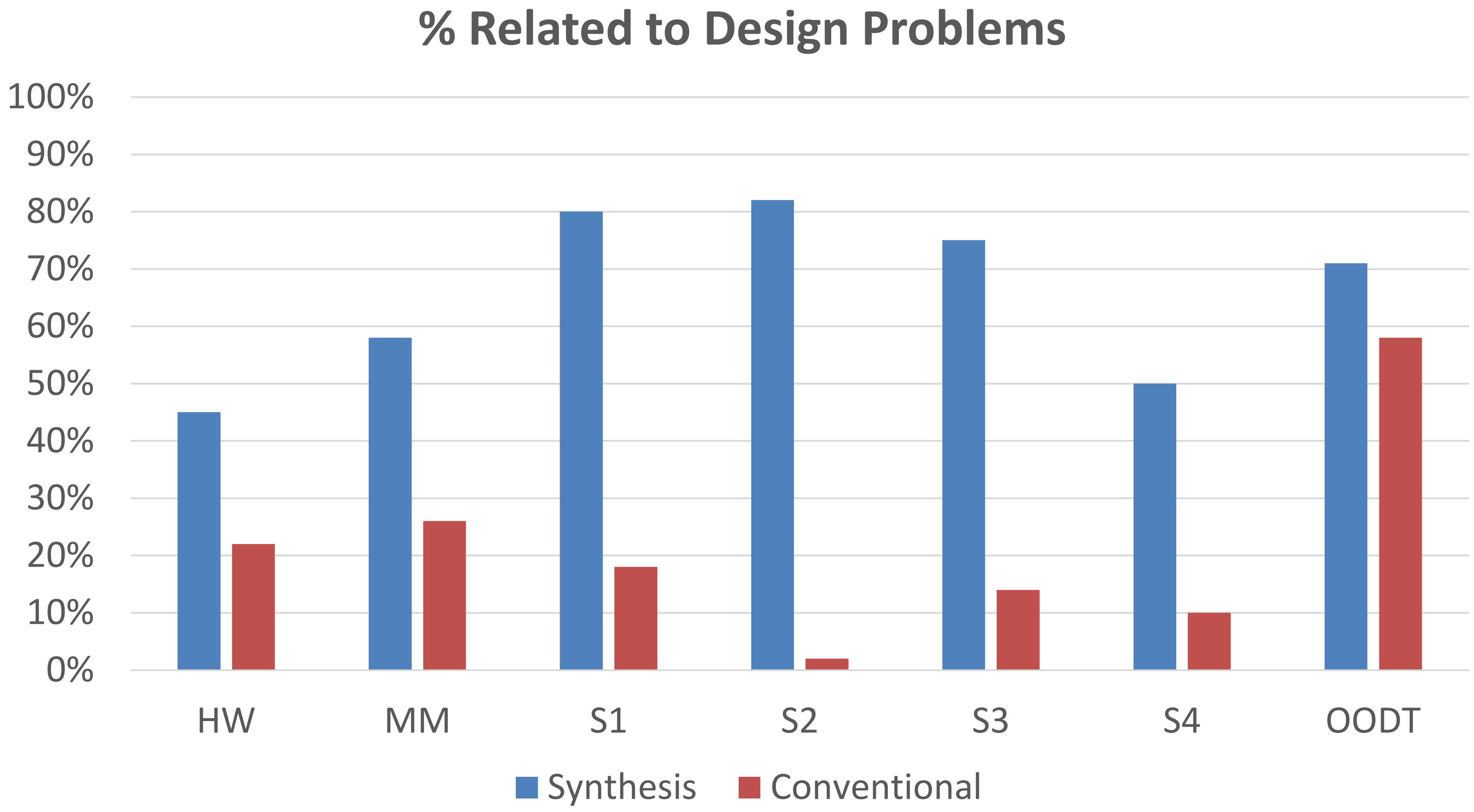
ID	Experience	Education	DD	Knowledge			
	(in years)			OODT	Java	CR	Eclipse
1	5	PhD	Yes	None	Advanced	Advanced	Advanced
2	5	Graduate	Yes	None	Intermediary	Intermediary	Intermediary
3	6	Graduate	Yes	None	Advanced	Basic	Advanced
4	12	Graduate	Yes	None	Expert	Advanced	Expert
5	5	Graduate	Yes	None	Advanced	Advanced	Advanced
6	10	Graduate	Yes	None	Intermediary	Intermediary	Intermediary
7	8	Master	Yes	None	Advanced	Intermediary	Advanced
8	4	PhD	Yes	None	Advanced	Intermediary	Advanced

DD = Has experience with Design Decisions?

CR = Code Anomalies and Refactoring

RQ1: Synthesis vs Conventional

Multi-case study



RQ1: Synthesis vs Conventional

Quasi-experiment

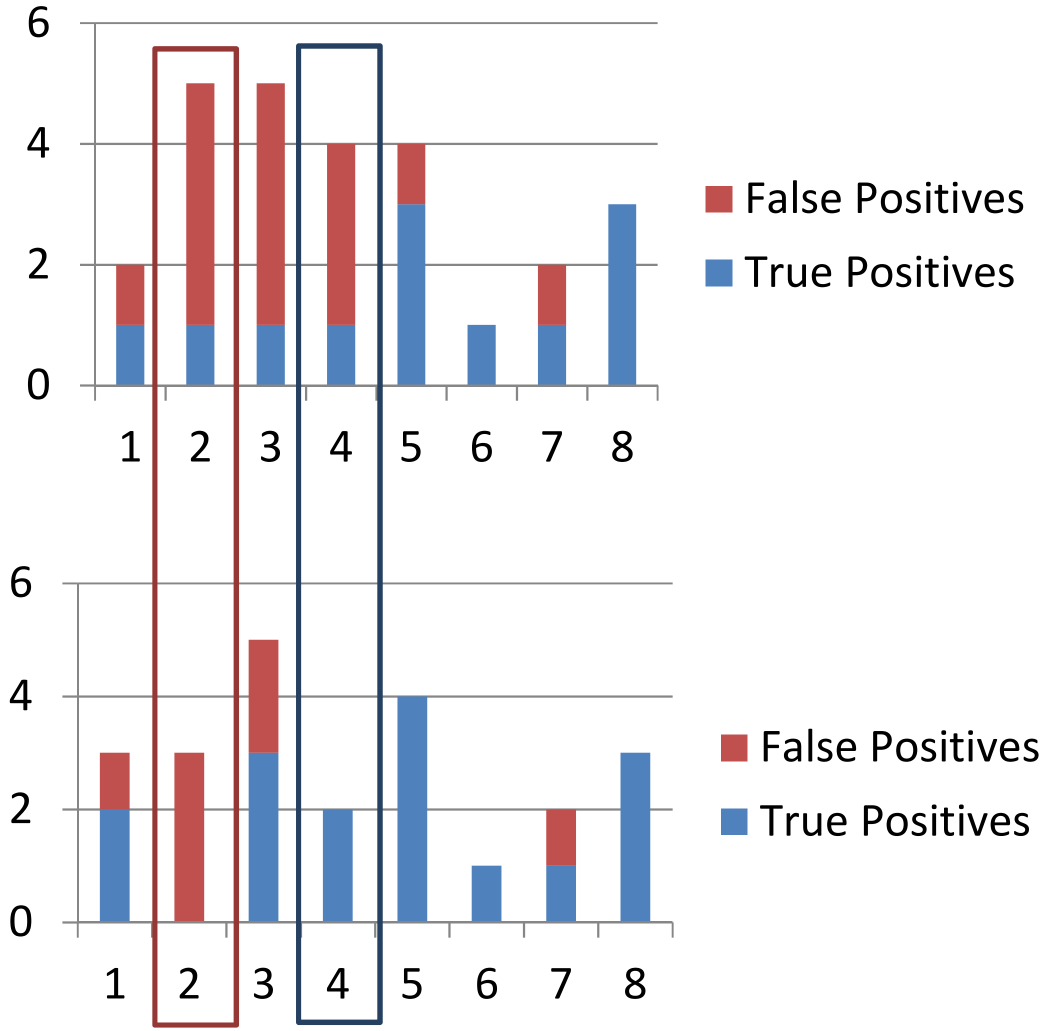
Conventional Technique

- Higher number of guesses (26)
- More false positives (53%)

Synthesis Technique

- Lower number of guesses (21)
- Less false positives (33%)

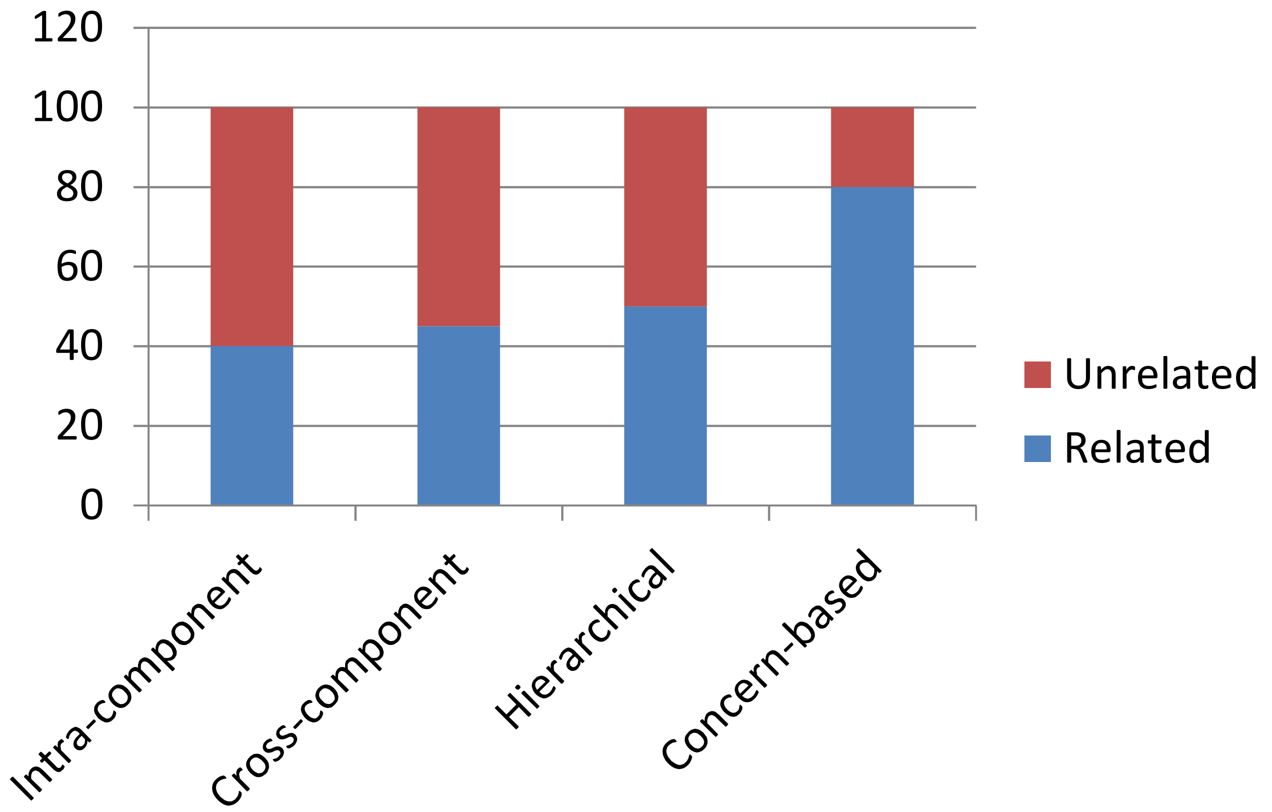
RQ1: Strong evidence that **Synthesis** is **better** than **Conventional**



RQ2: Which is the better topology?

Multi-case study

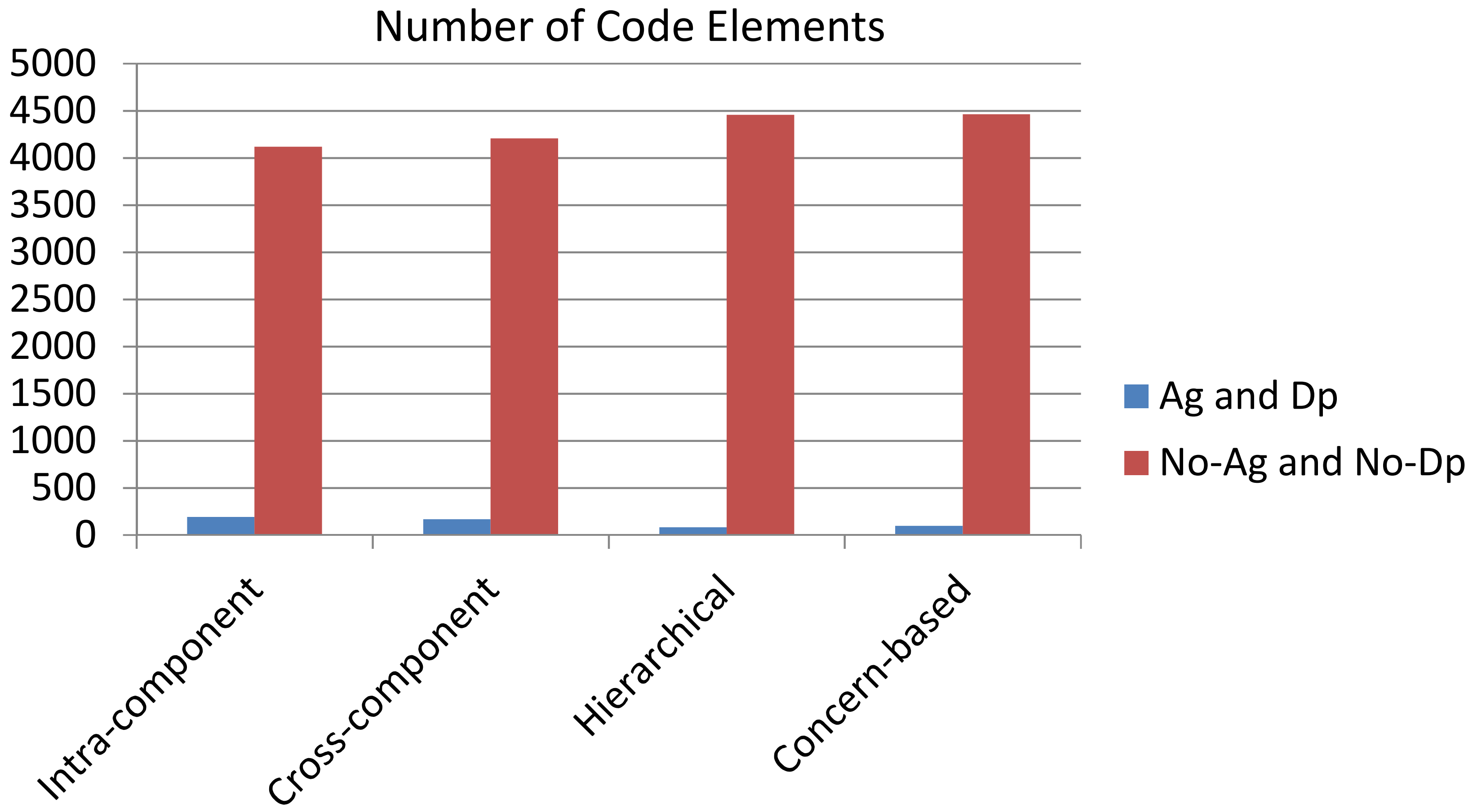
- **Concern-based** topology presented the **lower** number of **false positives** (i.e., agglomerations unrelated to design problems)



RQ2: Which is the better topology?

Multi-case study

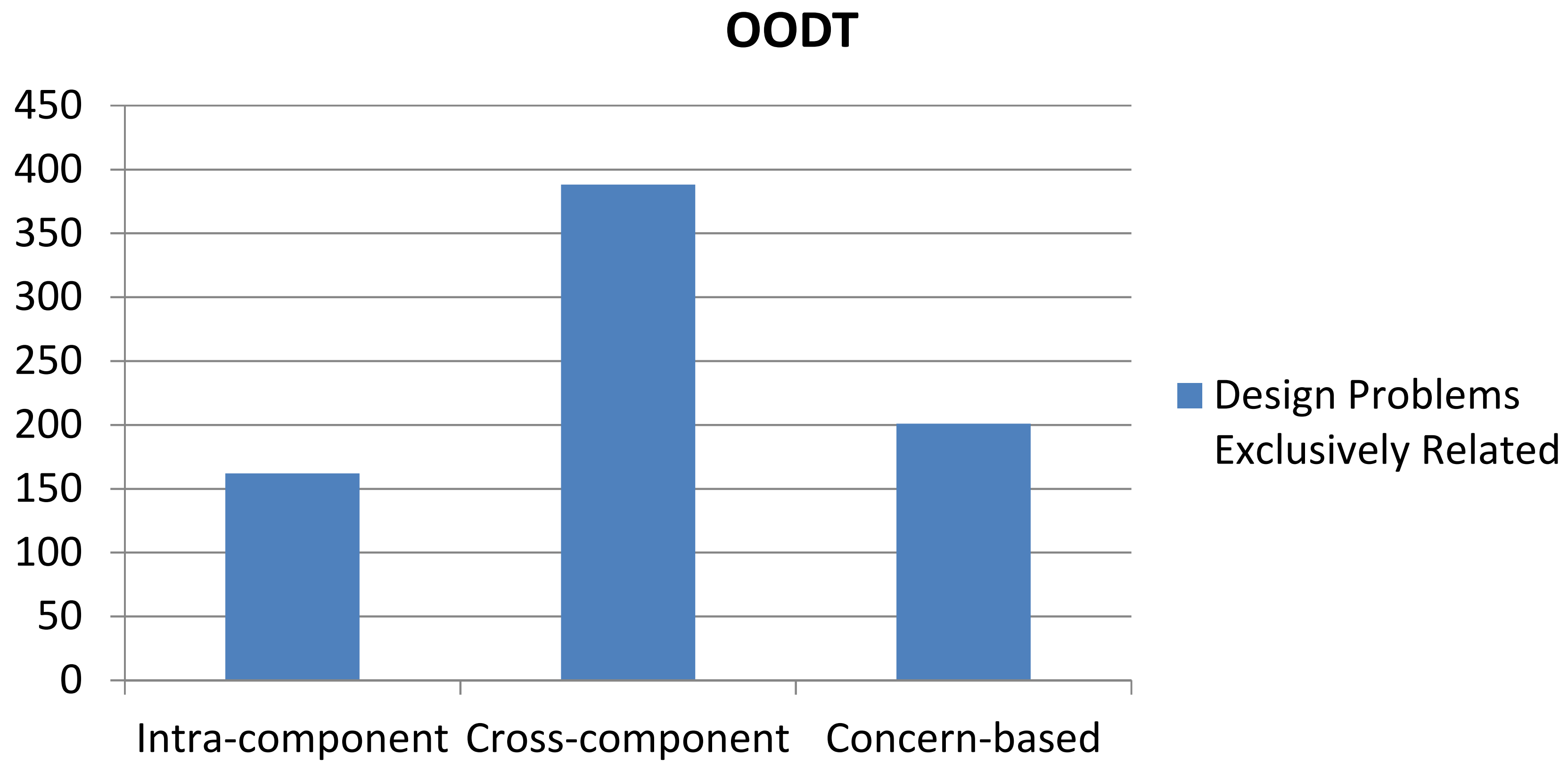
- All of them help developers to discard irrelevant anomalies



RQ2: Which is the better topology?

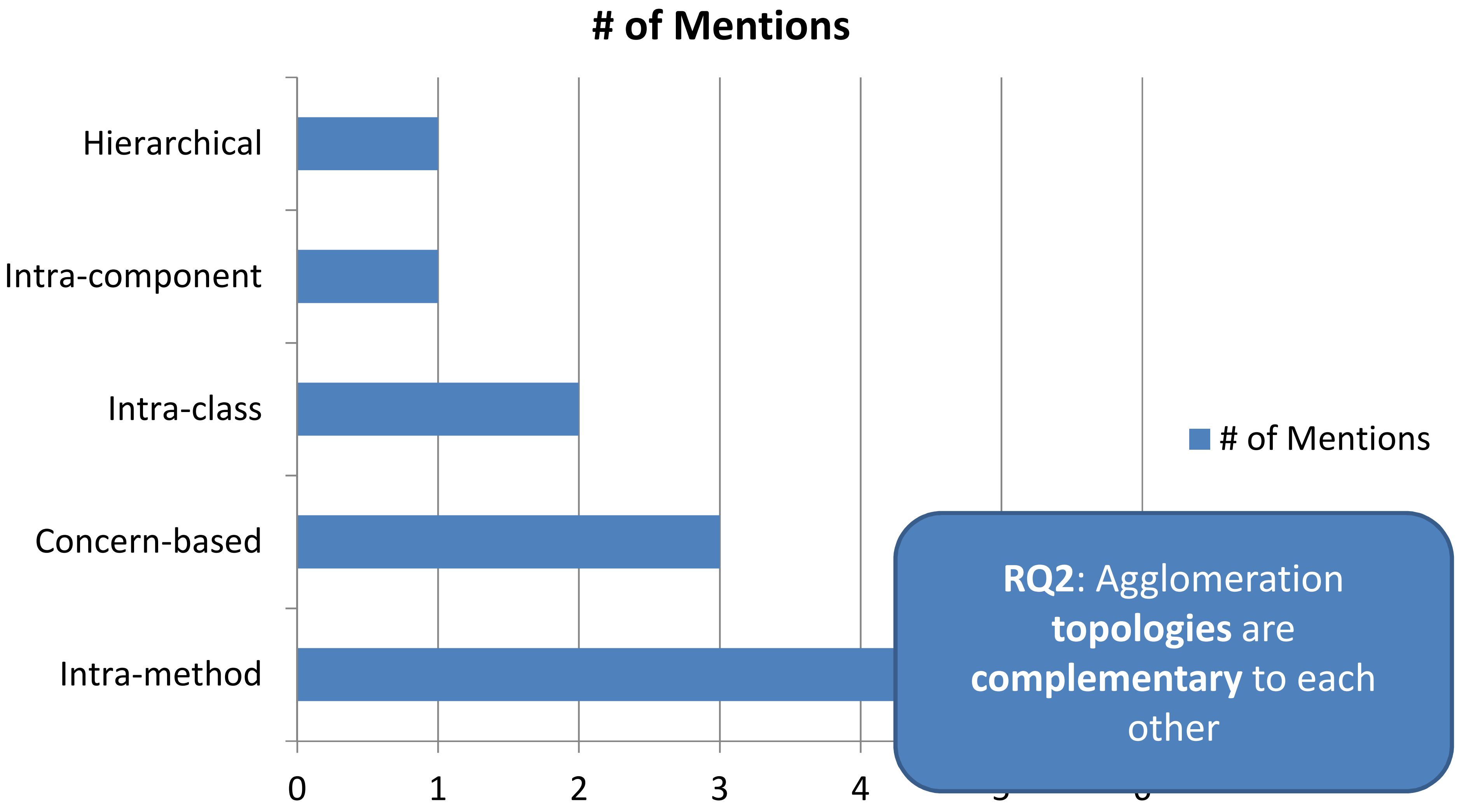
Multi-case study

- Each topology reveals problems not revealed by other topologies
- Example:



RQ2: Which is the better topology?

Quasi-experiment



RQ2: Agglomeration **topologies** are **complementary** to each other

Conclusion

- Design problems are caused by design decisions that negatively impact the resulting system’s quality

- They may be responsible for the reengineering or even the discontinuation of a system

- However, state-of-art techniques are not effective

Contributions

- Synthesis Technique
  - **Detects** code anomalies using detection strategies
  - **Searches** for code-anomaly **agglomerations**
  - **Summarizes contextual** and **history** information
- Tool Support
  - **Organic:** Eclipse plugin for java programs
- Empirical Evaluations
  - **Synthesis technique** is **better** than conventional techniques
  - Agglomeration **topologies** are **complementary** to each other

Publications

- Oizumi, Willian, *et al.* "Towards the synthesis of architecturally-relevant code anomalies.“, **WMod**, 2014 [(1st) **Best Paper Awards**]
- Oizumi, Willian, et al. "When Code-Anomaly Agglomerations Represent Architectural Problems? An Exploratory Study." **SBES**, 2014 [(3rd) **Best Paper Awards**]
- Oizumi, Willian, *et al.* "On the relationship of code-anomaly agglomerations and architectural problems.”, **JSERD**, 2015
- Oizumi, Willian et al. “Code Anomalies Flock Together: Exploring Code Anomaly Agglomerations for Locating Design Problems”, **ICSE**, 2016 (Accepted)

Future Work

- Propose a semi-automated technique for the removal of design problems
  - Tips of possible design problems
  - Prioritization of agglomerations
  - Proposal of refactoring strategies

- Improve the **visualization mechanism** provided by Organic

- Improve techniques for the identification of concerns

Continuous Detection of Code Anomalies: Synthesis of Code Anomalies

Towards Revealing Design Problems in Source Code

Alessandro Garcia – afgarcia@inf.puc-rio.br

Willian Oizumi – woizumi@inf.puc-rio.br

LES | DI |PUC-Rio - Brazil

OPUS Research Group