Integrating Logical Dependencies in Software Clustering: A Case Study on Apache Ant

Adelina Stana Ioana Șora

Computer Science and Engineering Department
"Politehnica" University of Timișoara, Romania

Outline

Introduction

- Software clustering organizes software entities into meaningful modules. Most existing clustering methods rely on structural dependencies extracted from code.
- We propose using logical dependencies extracted from versioning systems for software clustering.
- Our goal is to assess the impact of integrating logical dependencies into software clustering by conducting a case study on Apache Ant, evaluated using the Modularization Quality (MQ) metric.

Co-changes

- Instances where multiple software entities are changed together in the same commit in the versioning system.
- Co-changes represent raw data indicating potential relationships.
- However, not all co-changes can be meaningful dependencies.

From Co-changes to Logical Dependencies

Definition

Logical dependencies are co-changes that meet specific criteria, indicating a reliable relationship between software entities.

- Logical dependencies are extracted from co-changes extracted from the versioning system.
- They are more reliable than raw co-changes due to the filtering process.
- **Co-changes are not necessarily logical dependencies**; only filtered co-changes that pass the criteria become logical dependencies.

Filtering Co-changes

- Filtering is applied to co-changes to extract meaningful relationships.
- Filters help remove noise caused by:
  - Large commits with many files (e.g., formatting changes).
  - Rare co-changes that may not indicate a dependency.
- Criteria for filtering include:
  - **Commit Size Threshold**: Exclude commits that change more than a threshold number of files.
  - **Strength Metric Threshold**: Only consider co-changes with a strength above a certain level.

Commit Size Filter

- Excludes commits changing too many files.
- Large commits may introduce noise.
- We set a threshold of max 10 files per commit for this filter.

Strength Filter (1/2)

- This filter ensures only strong dependencies are considered.

Support and Confidence Metrics

- **Support** measures how often two entities change together:

$$\text{support}(A \rightarrow B) = \text{freq}_{\text{total commits}}(A \cup B)$$

- **Confidence**:

$$\text{confidence}(A \rightarrow B) = \frac{\text{support}(A \rightarrow B)}{\text{freq}_{\text{total commits}}(A)}$$

Strength Filter (2/2)

Strength Metric

- **System Factor**:

$$\text{system factor for } (A \rightarrow B) = \frac{\text{support}(A \rightarrow B)}{\text{system mean}}$$

  The *system mean* is the mean value of all the support values for all the association rules from the system.

- **Strength** between entities A and B:

$$\text{strength}(A \rightarrow B) = \frac{\text{support}(A \rightarrow B) \times 100}{\text{freq}_{\text{total commits}}(A)} \times \text{system factor}$$

The strength metric ranges from 0 to 100, where 100 represents the best possible score.

Methodology Overview

- Case study on Apache Ant.
- Three scenarios:
  1. Clustering using structural dependencies only.
  2. Clustering using logical dependencies only.
  3. Clustering using both logical and structural dependencies.
- Use of Louvain Clustering algorithm.
- Evaluation using Modularization Quality (MQ) metric.

Clustering Generation Process

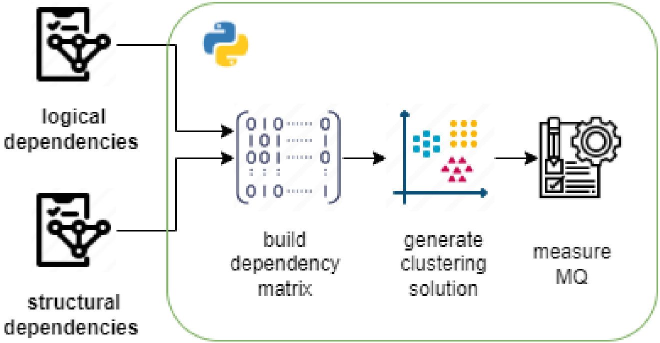

Figure 1: Clustering solution creation process diagram

Louvain Clustering Algorithm

- Community detection algorithm for complex networks.
- Optimizes modularity by moving nodes between clusters.
- Suitable for large-scale clustering.

Clustering Results

In our experiments, for logical dependencies filtering, we started with a strength metric threshold of 10, incrementing in steps of 10 up to 100.

Table 1: Louvain Clustering Results - Highlights

Dataset	**Entities**	**Clusters**	**MQ Metric**
SD only	517	12	0.08
LD only (Strength 100%)	64	19	0.611
SD + LD (Strength 30%)	517	15	0.227

Clustering Results: Logical Dependencies Only

- As the strength threshold increases, the number of entities decreases. Higher thresholds filter out more dependencies, leading to fewer entities known by the system.

Table 2: MQ Results for Logical Dependencies Only

Strength	**Entities**	**Clusters**	**MQ**
SD only	517	12	0.08
10%	320	56	0.506
20%	215	53	0.547
30%	174	44	0.558
40%	152	40	0.580
50%	138	35	0.604
60%	120	34	0.587
70%	106	32	0.577
80%	92	29	0.576
90%	79	24	0.606
100%	64	19	**0.611**

Clustering Results: Combined Dependencies

Table 3: MQ Results for Logical + Structural Dependencies

Strength	**Entities**	**Clusters**	**MQ**
SD only	517	12	0.08
10%	517	13	0.191
20%	517	13	0.176
30%	517	15	**0.227**
40%	517	16	0.214
50%	517	15	0.213
60%	517	16	0.211
70%	517	16	0.211
80%	517	15	0.210
90%	517	12	0.124
100%	517	13	0.137

Analysis of Results

- MQ scores for **LD only** are higher than for **SD + LD** or **SD only**.
- **LD only** achieves MQ up to 0.611 at 100% strength.
- However, **LD only** covers fewer entities (e.g., 64 entities at 100% strength).
- **SD + LD** covers the entire system (517 entities), a complete coverage.
- Balance between clustering quality and system coverage:
  - **LD only**: Better MQ scores but limited coverage.
  - **SD + LD**: Lower MQ scores but full system coverage.
- Including logical dependencies improves MQ compared with **SD only** (MQ: 0.08).

Conclusion

- Incorporating logical dependencies improves clustering quality.
- Logical dependencies provide additional insights not available from code analysis alone.
- The combined approach leads to clusters with an improved MQ metric and complete system coverage.

Future Work

- Expand analysis to more projects.
- Explore alternative evaluation metrics and clustering algorithms.