Methods and Tools for the Analysis of Legacy Software Systems

Stana Adelina Diana

Computer Science and Engineering Department
"Politehnica" University of Timisoara

2019

Presentation of the research topic

The thesis will develop methods for the analysis of software systems using historical information from the versioning systems[1].

[1] Versioning systems keep track of every change to a file over time so early versions can be restored and used by software teams.

Structural dependencies

Definition

Structural dependencies are the result of *source code analysis and can be extracted from : members, call parameters, local variables.*

```
class ClassA{
    ...
    public void doSomething(ClassB b){
        b.doWork();
    }
}
```

```
class ClassB{
    ...
    public void doWork(){
        ....
    }
}
```

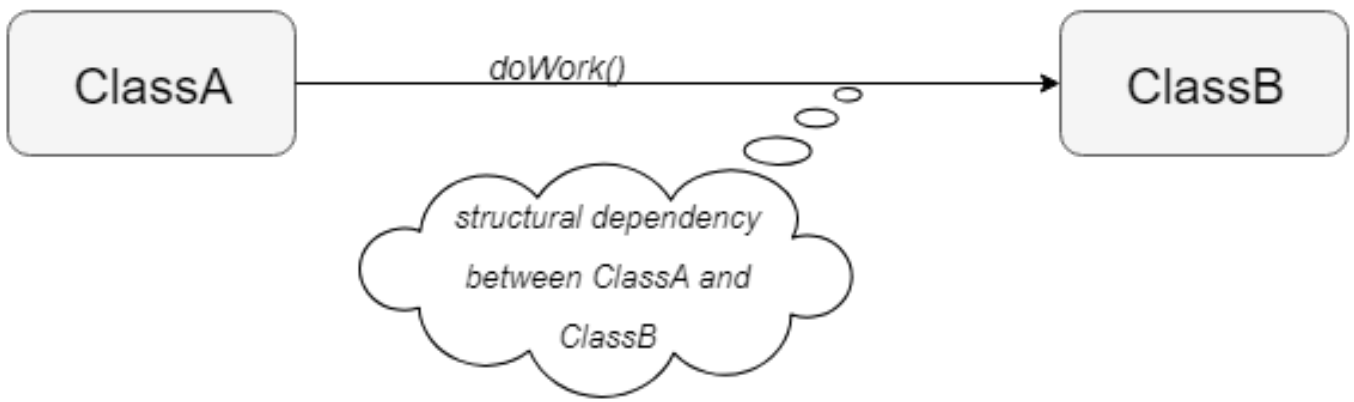


Figure 1: Example of structural dependency between two classes

Logical dependencies

Definition

Logical dependencies are the result of software history analysis and can reveal relationships that are not present in the source code code (structural dependencies).

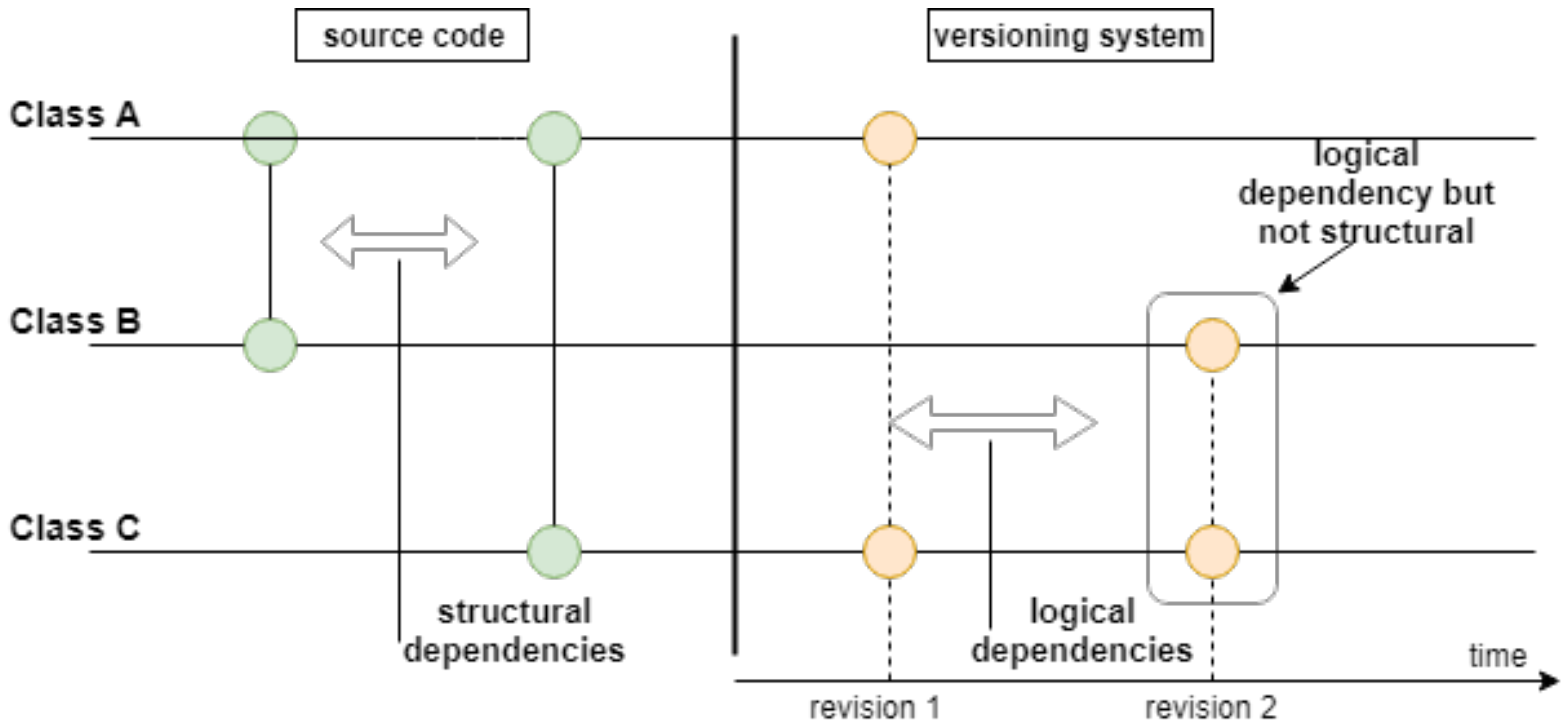


Figure 2: Example of logical and structural dependencies

Applications of software dependencies

- Reverse engineering
- Architecture reconstruction
- Identifying clones
- Code smells
- Comprehension
- Fault location
- Error proneness
- Empirical software engineering research

Current status of research in identifying logical dependencies

The current trend recommends that general dependency management methods and tools should also include logical dependencies besides the structural dependencies [2] [3].
But there are no strict rules to *filter co-changes into logical dependencies*. Other researches filtered co-changes only in order to decrease their number and not to increase their validity.

[2] Gustavo Ansaldi Oliva and Marco Aurelio Gerosa. On the interplay between structural and logical dependencies in open-source software.

[3] Nemitari Ajienka and Andrea Capiluppi. Understanding the interplay between the logical and structural coupling of software classes.

Current status of doctoral research: Introduction

We studied 20 open source systems written in Java and CSharp. Filters and Thresholds:

- commit size (cs): the maximum size of commit transactions which are accepted to generate logical dependencies. The values for this threshold were 5, 10, 20 and no threshold (infinity).
- number of occurrences (occ): the minimum number of repeated occurrences for a co-change to be counted as logical dependency. The values for this threshold were 1, 2, 3 and 4.
- with/without taking comments into consideration as valid change.

Open source projects studied

ID	Project	Nr. of classes	Nr. of commits	
1	urSQL	41	89	java
2	JavaCoder	5	11	java
3	jbandwidthlog	14	54	java
4	sjava-logging	18	62	java
5	daedalum	66	29	java
6	prettyfaces	236	207	java
7	jbal	102	113	java
8	guavatools	237	85	java
9	monome-pages	240	280	java
10	kryo	309	743	java
11	bitlyj	21	81	java
12	slema	276	368	java
13	bluecove	435	1679	java
14	gp-net-radius	25	28	java
15	aima-java	833	1181	java
16	powermock	966	1512	java
17	restfb	757	1545	java
18	Tensorflow	1104	2386	cpp
19	mangnum	143	1728	cpp

Tool for measuring software dependencies

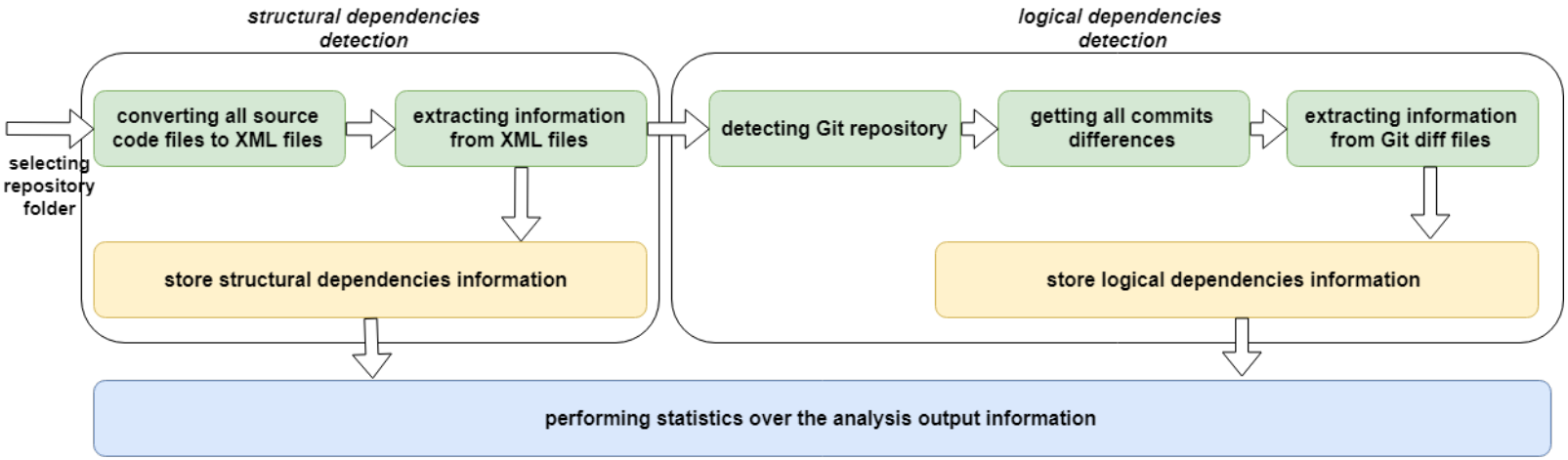


Figure 3: Workflow diagram of the tool

The workflow can be delimited by three major steps as it follows:

- Extracting structural dependencies.
- Extracting co-changes.
- Processing the information extracted.

Commit transactions size

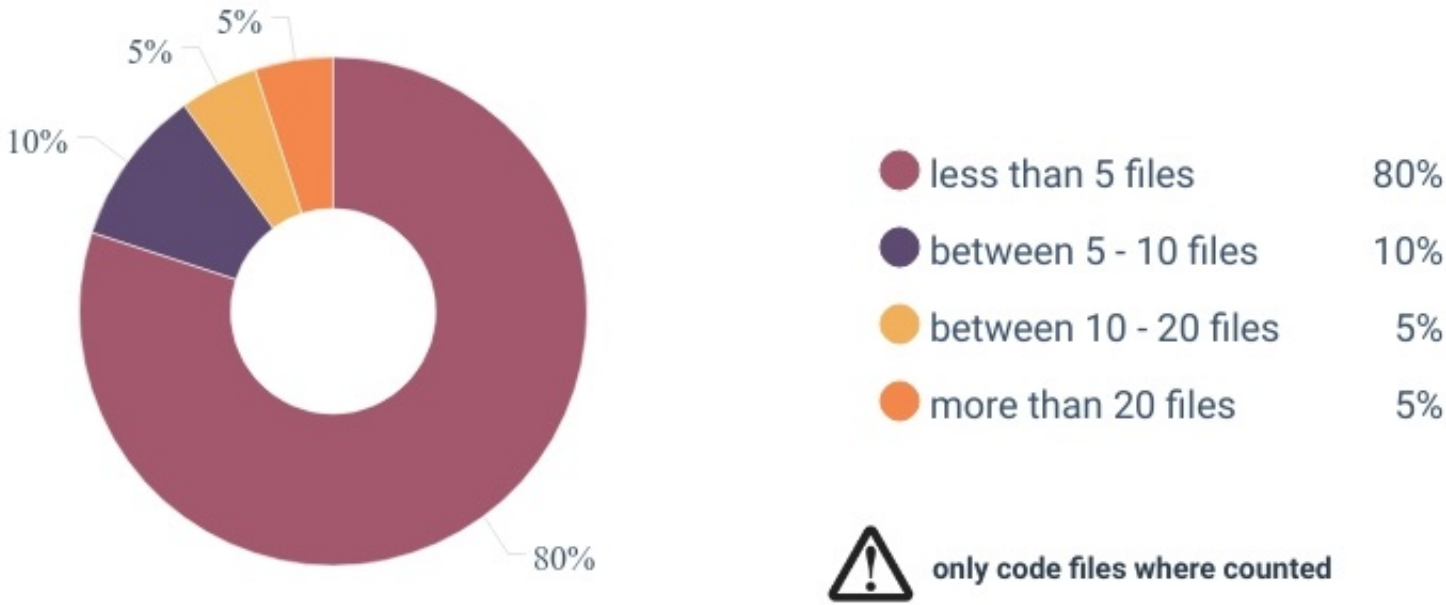


Figure 4: Commit transactions size: overview in percentages

Pairs of co-changes extracted

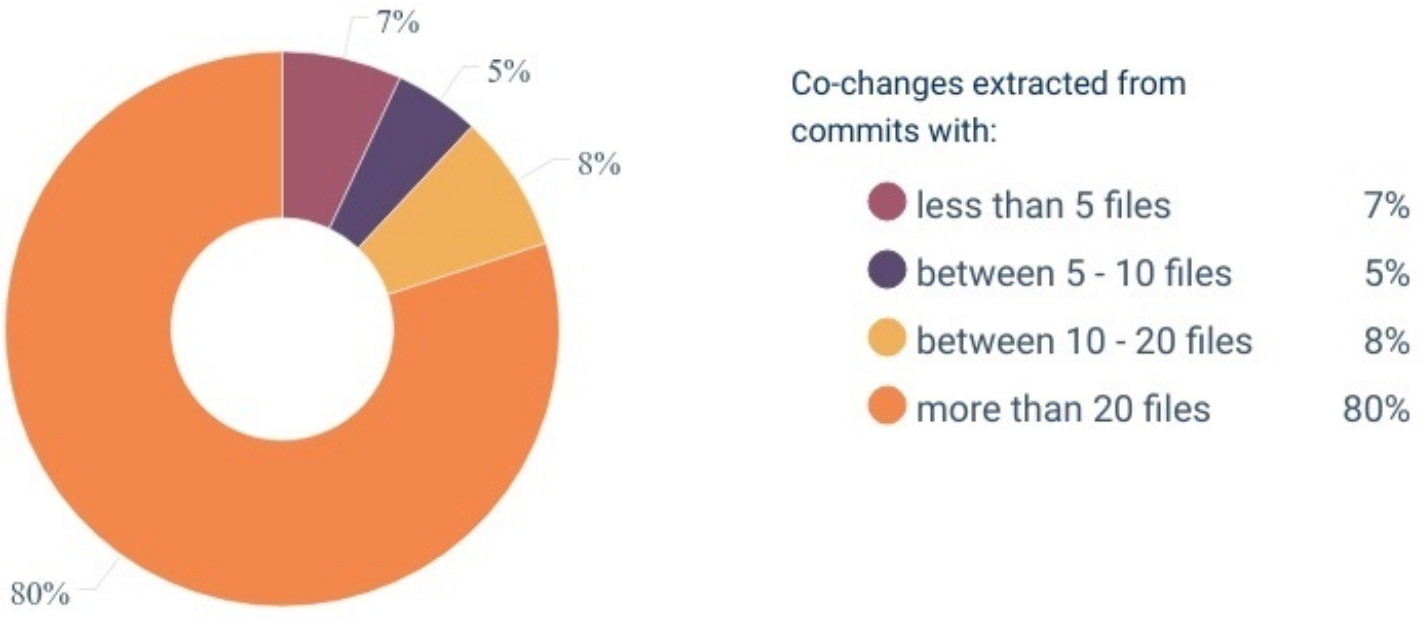
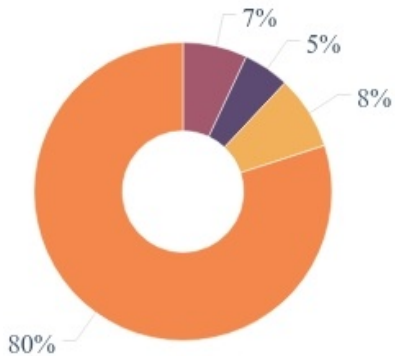


Figure 5: Pairs of co-changes extracted: overview in percentages

Pairs of co-changes extracted - observation



Co-changes extracted from commits with:

less than 5 files	7%
between 5 - 10 files	5%
between 10 - 20 files	8%
more than 20 files	80%

5% of total commits generate 80% from total co-changes extracted

Comments changes

```
5   5
6   6     /**
7       -  * this is a test
    7   +  * this is a comment
8   8      */
9   9     public class ApplicationTest extends Application
10  10        public ApplicationTest() {
```

- approx -5% from total co-changes extracted from all commit sizes
- approx -1% from total co-changes extracted from commits with less than 10 files

Setting thresholds for the number of occurrences

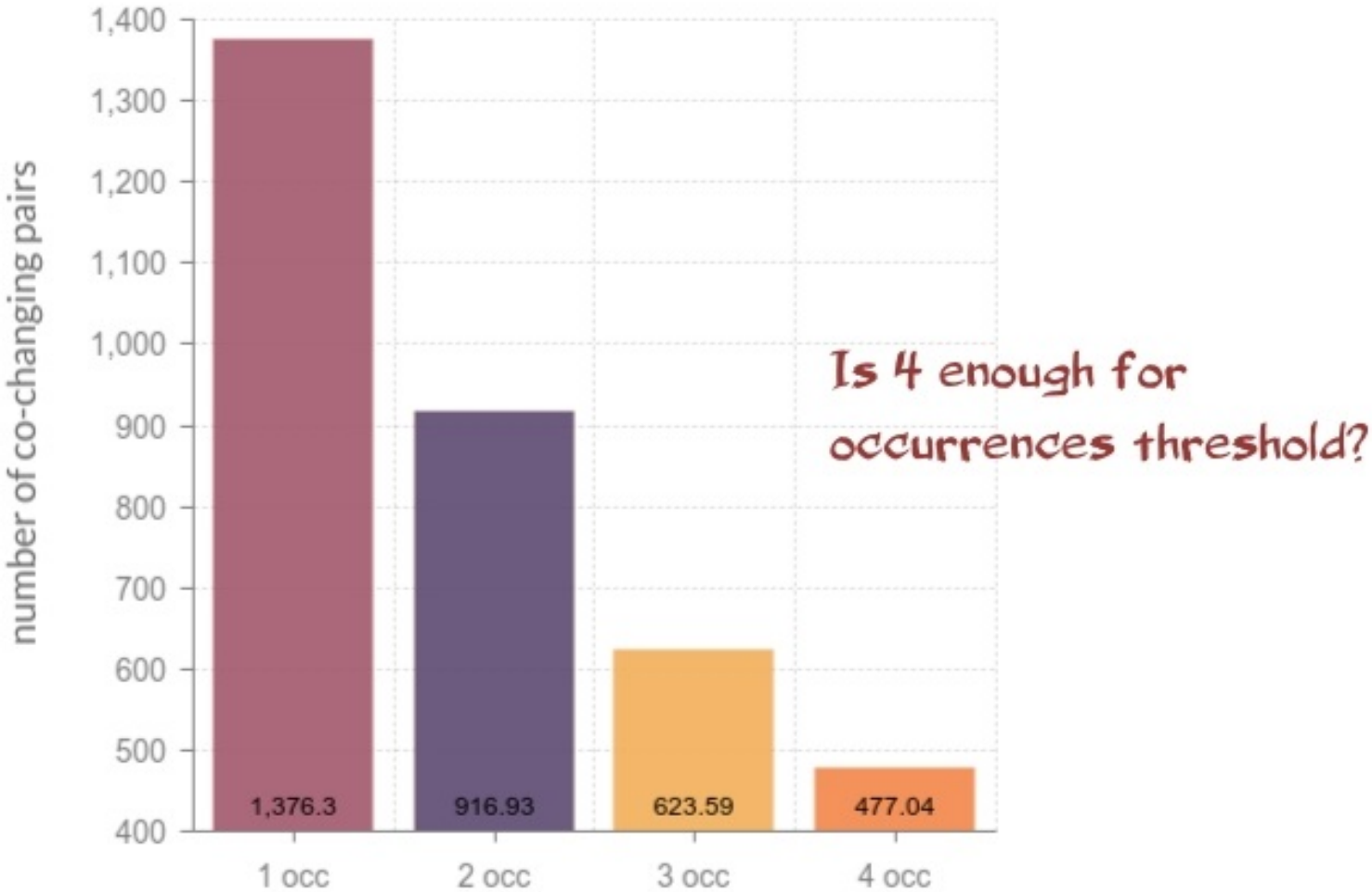


Figure 6: Pairs of co-changes extracted reported to occ threshold

Refine filter for occurrences of co-changing classes

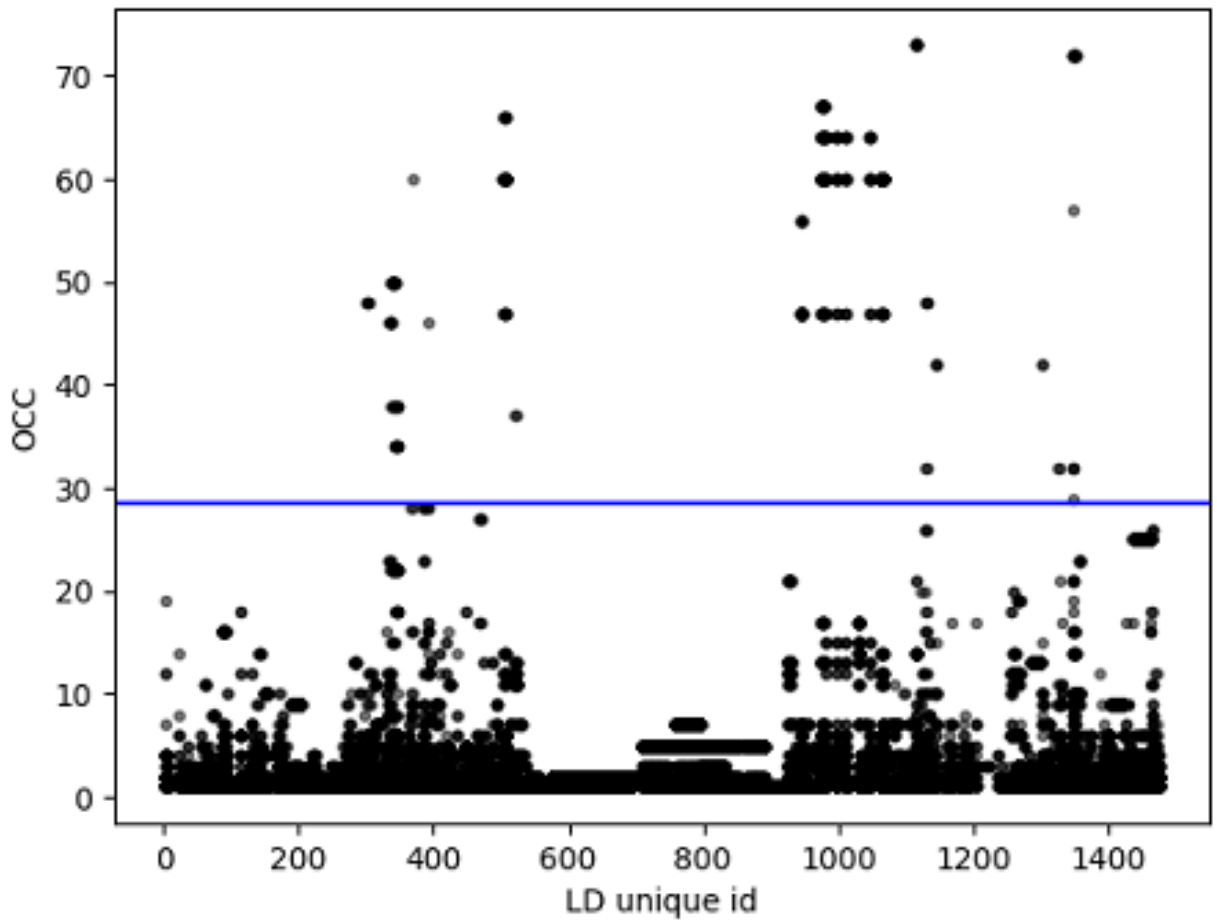
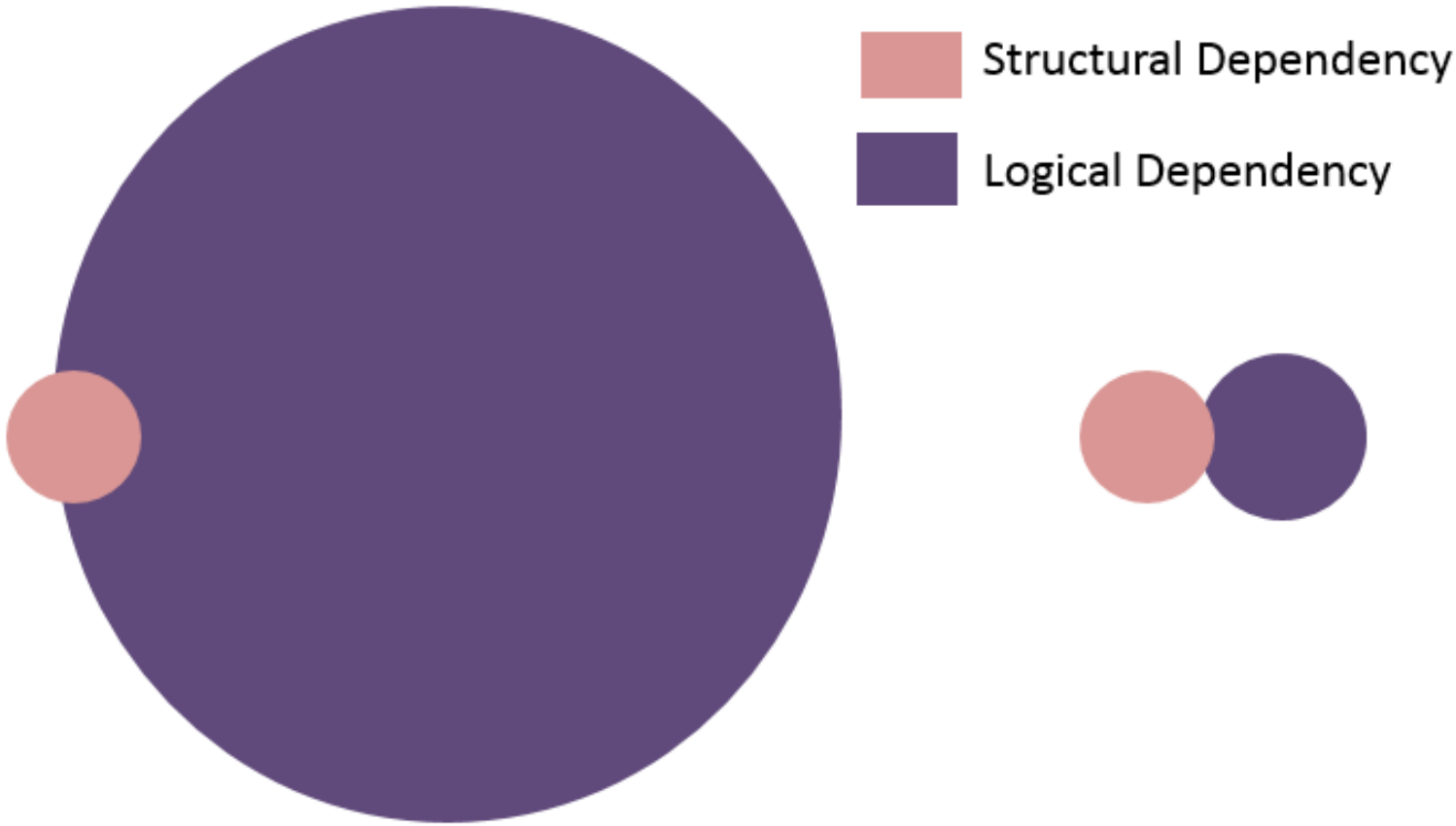


Figure 7: Occurrences rates of co-changing classes extracted from one system. Number of total commits: 6000.

Logical and structural dependencies overlapping



Case a.)
cs $\leq \infty$
occ ≥ 1
LD/SD=314.59

Case b.)
cs ≤ 5
occ ≥ 4
LD/SD=1.81

Conclusions

- The most important factors in co-changing classes filtering:
  - number of changed files taken into consideration (commit size)
  - number of occurrences
- Filtering thresholds shall be calculated according to the system size
- + - 5% for comment changes

Proposed research stages: Development of content and tools

Stage 1: Build tool to extract structural dependencies from code and co-changes from git for a given set of projects.
Stage 2: Find filters for the co-changes extracted and establish different thresholds for those filters.
Stage 3: Study the impact of those filters and the corresponding thresholds on the remaining quantity of co-changes for each system. Study the overlapping between the remaining pairs of co-changing entities and the structural dependencies extracted [4] [5].

[4] Adelina Stana. and Ioana Sora. Identifying logical dependencies from co-changing classes. ENASE 2019.

[5] Adelina Stana. and Ioana Sora. Analyzing information from versioning systems to detect logical dependencies in software systems. SACI 2019.

Proposed research stages: Development of content and tools

Stage 4: Establish a dynamic way to determine the thresholds for filters in order to fit the best each studied system. Main focus on the threshold for number of occurrences of co-changing pairs. Use plots or other visual instruments in order to see the highest and the lowest rates for the numbers of occurrences among co-changing pairs.

Proposed research stages: Development of content and tools

Stage 5: Take into account also structural dependencies from all the revisions of the system to filter out the old, out-of-date logical dependencies. Here an extra check is needed, it can be a case in which old structural dependencies that were also logically linked to continue to be logically linked even after the structural dependency was removed.

Proposed research stages: Usage

Stage 6: Use logical dependencies among structural dependencies in tools for architectural reconstruction to evaluate the improvement.

Stage 7: Compare the number of logical dependencies with metrics like Fan Out, Fan In and study their connections.

- Fan Out - number of other classes referenced by a class.
- Fan In - number of other classes that reference a class.

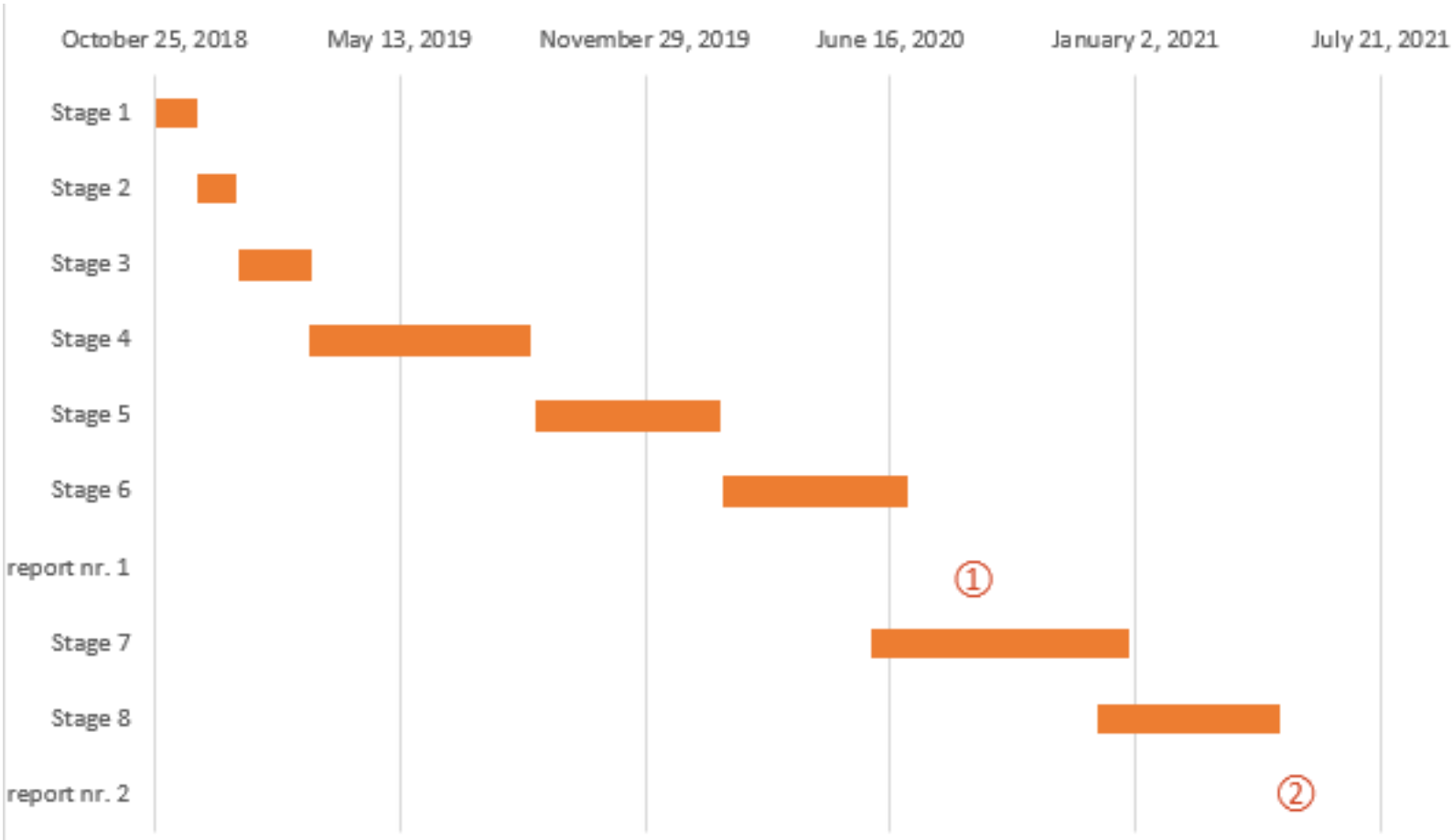
Stage 8: Identify other tools that use structural dependencies and evaluate the impact of co-changes filtering into logical dependencies for them.

Scientific reports and deadlines of stages

Gantt Chart

START DATE	END DATE	DESCRIPTION	PAPER OUTPUT	DURATION (days)
10/25/2018	11/30/2018	Stage 1	No	35
11/30/2018	12/30/2018	Stage 2	No	30
1/1/2019	3/1/2019	Stage 3	Yes: ENASE, SACI	60
3/1/2019	9/1/2019	Stage 4	No	180
9/1/2019	2/1/2020	Stage 5	Yes: ICSME	150
2/1/2020	7/1/2020	Stage 6	Yes: ENASE	150
7/1/2020	scientific report nr. 1 with the intermediate results of the research			
6/1/2020	1/1/2021	Stage 7	Yes: ICSE	210
12/1/2020	5/1/2021	Stage 8	No	150
5/1/2021	scientific report nr. 2 with the intermediate result of the research			
9/1/2021	doctoral thesis completion			

Gantt chart


October 25, 2018
May 13, 2019
November 29, 2019
June 16, 2020
January 2, 2021
July 21, 2021
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Stage 6
report nr. 1
①
Stage 7
Stage 8
report nr. 2
②