# Methods and Tools for the Analysis of Legacy Software Systems
# Report 2. Logical dependencies in practice.

Stana Adelina Diana

Computer Science and Engineering Department
"Politehnica" University of Timisoara

May, 2021

# Presentation of the research topic

The goal of the thesis is to develop methods for analyzing legacy software systems by using historical information extracted from the versioning systems. We divided our work into two main parts

- historical information collection and filtering
- **usage of the collected information in order to analyze the software systems**

# State of the art in key classes detection

### Definition

Classes that can be found in documents written to provide an architectural overview of the system or an introduction to the system structure. Also known as important classes.

# State of the art in key classes detection (cont.)

The key class identification can be done by using different algorithms:

- Osman et al., used a machine learning algorithm and class diagrams [2]
- Thung et al. builds on top of Osman et al.'s approach and adds network metrics and optimistic classification [3]
- Zaidman et al. use a webmining algorithm and dynamic analysis of the source code [4]
- Sora et al. use static analysis of the source code and a page ranking algorithm together with other class attributes [1]

# Results evaluation

To evaluate the quality of the approach and solution produced, the key classes found are compared with a reference solution. *The reference solution is extracted from the developer's documentation.*

# Metrics for results evaluation

For the comparison between both solutions is used a classification model and the Receiver Operating Characteristic Area Under Curve (ROC-AUC) metric to evaluate the performance.

This tool presents the result as a number between 0 and 1. For a classifier to be considered good, its ROC-AUC metric value should be as close to 1 as possible.

# Baseline approach

We took the research done by Sora et al. [1] as the baseline for our research.

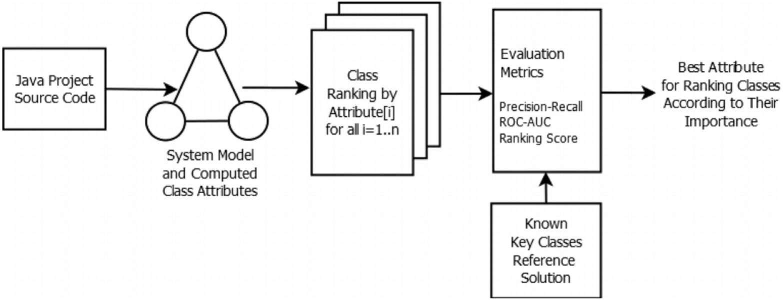



Figure 1: Overview of the baseline approach. From "Finding key classes in object-oriented software systems by techniques based on static analysis." by Ioana Sora and Ciprian-Bogdan Chirila, 2019, Information and Software Technology, 116:106176.

# Data set used

Highlighted with green are the systems used in the baseline research that can also be used in the current research.

Table 1: Found systems and versions of the systems in GitHub.

| ID | System | Version | Release Tag name | Commits number |
|---|---|---|---|---|
| SI | Apache Ant | 1.6.1 | rel/1.6.1 | 6713 |
| S2 | Argo UML | 0.9.5 | not found | 0 |
| S3 | GWT Portlets | 0.9.5 beta | not found | 0 |
| S4 | Hibernate | 5.2.12 | 5.2.12 | 6733 |
| S5 | javaclient | 2.0.0 | not found | 0 |
| S6 | jEdit | 5.1.0 | not found | 0 |
| S7 | JGAP | 3.6.3 | not found | 0 |
| S8 | JHotDraw | 6.0b.1 | not found | 149 |
| S9 | JMeter | 2.0.1 | v2_1_1 | 2506 |
| S10 | Log4j | 2.10.0 | v1_2_10-recalled | 634 |
| S11 | Mars | 3.06.0 | not found | 0 |
| S12 | Maze | 1.0.0 | not found | 0 |
| S13 | Neuroph | 2.2.0 | not found | 0 |
| S14 | Tomcat Catalina | 9.0.4 | 9.0.4 | 19108 |
| S15 | Wro4J | 1.6.3 | v1.6.3 | 2871 |

# Comparison with the baseline approach

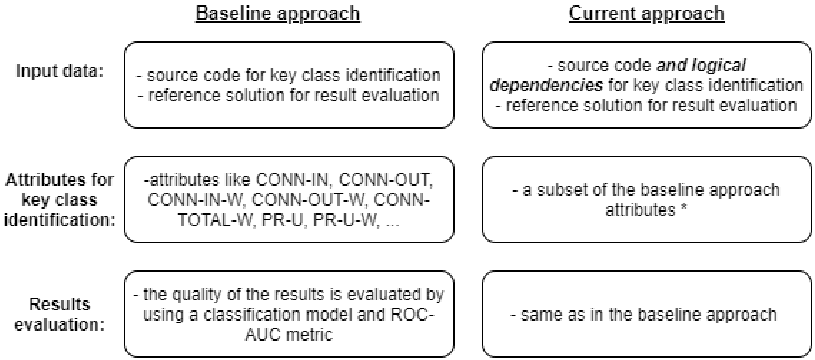



Figure 2: Comparison between the new approach and the baselines

** The reason why we are not using all the attributes is that the extracted logical dependencies are undirected.*

# Logical dependencies collection

Two filters are applied to co-changing pairs: the commit size filter with threshold 10, and the connection strength filter with a variable threshold.

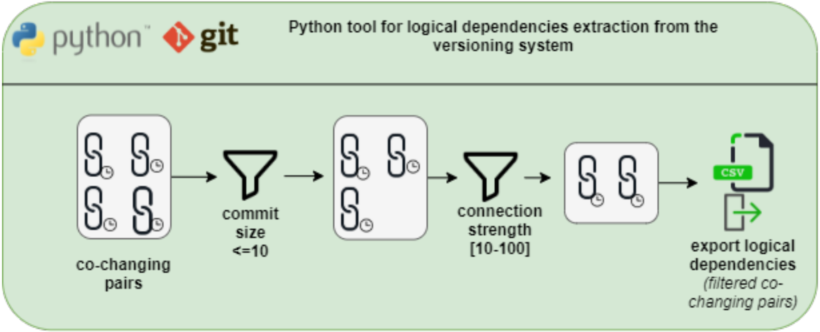



Figure 3: Logical dependencies extraction workflow and filters used

# Current workflow

We couple the tool that extracts logical dependencies and the tool that identifies key classes and evaluates the results.

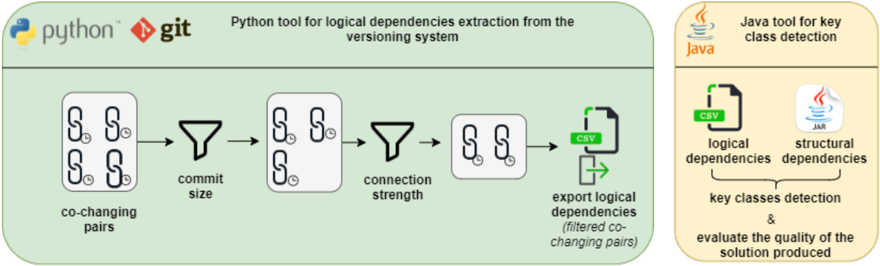



Figure 4: Workflow for key classes detection and evaluation of results

# Measurements using only the baseline approach

In table are presented the ROC-AUC values for different attributes computed for the systems Ant, Tomcat Catalina, and Hibernate by using the baseline approach.

Table 2: ROC-AUC metric values extracted.

| Metrics | Ant | Tomcat Catalina | Hibernate |
|---|---|---|---|
| PR_U2_W | 0.95823 | 0.92341 | 0.95823 |
| PR | 0.94944 | 0.92670 | 0.94944 |
| PR_U | 0.95060 | 0.93220 | 0.95060 |
| CONN_TOTAL_W | 0.94437 | 0.92595 | 0.94437 |
| CONN_TOTAL | 0.94630 | 0.93903 | 0.94630 |

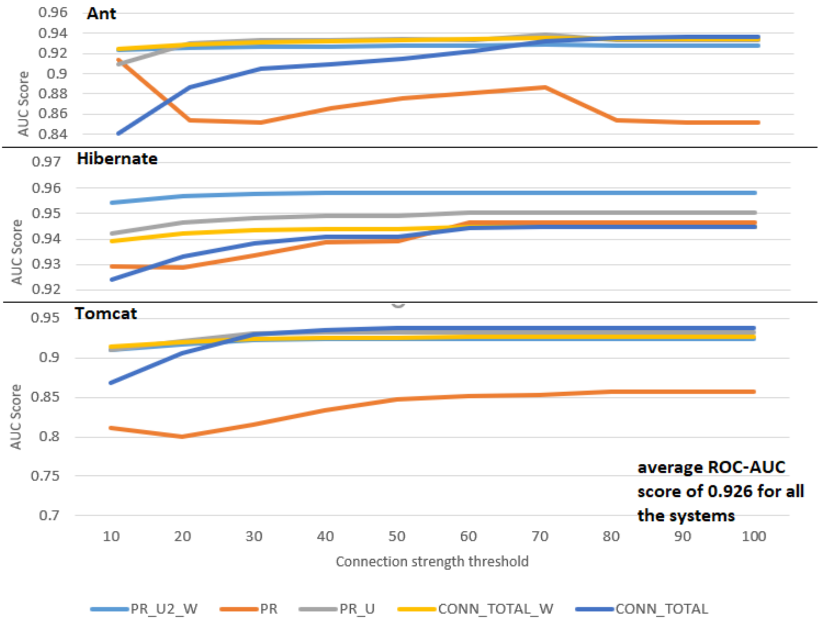
# Measurements using combined SD and LD

The tool used by the baseline approach builds a graph that contains the structural dependencies extracted from static source code analysis. We modified the tool to read also logical dependencies and add them to the graph.

Table 3: Measurements for Ant using structural and logical dependencies combined

| Metrics | $\geq 10\%$ | $\geq 20\%$ | $\geq 30\%$ | $\geq 40\%$ | $\geq 50\%$ | $\geq 60\%$ | $\geq 70\%$ | $\geq 80\%$ | $\geq 90\%$ | $\geq 100\%$ | Baseline |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PR_U2_W | 0.924 | 0.925 | 0.926 | 0.927 | 0.927 | 0.927 | 0.929 | 0.928 | 0.928 | 0.928 | 0.929 |
| PR | 0.914 | 0.854 | 0.851 | 0.866 | 0.876 | 0.882 | 0.887 | 0.854 | 0.852 | 0.852 | 0.855 |
| PR_U | 0.910 | 0.930 | 0.933 | 0.933 | 0.935 | 0.934 | 0.939 | 0.933 | 0.933 | 0.933 | 0.933 |
| CON_T_W | 0.924 | 0.928 | 0.931 | 0.932 | 0.933 | 0.934 | 0.936 | 0.934 | 0.934 | 0.934 | 0.934 |
| CON_T | 0.840 | 0.886 | 0.904 | 0.909 | 0.915 | 0.923 | 0.932 | 0.935 | 0.936 | 0.936 | 0.942 |

# Measurements using combined SD and LD (cont.)



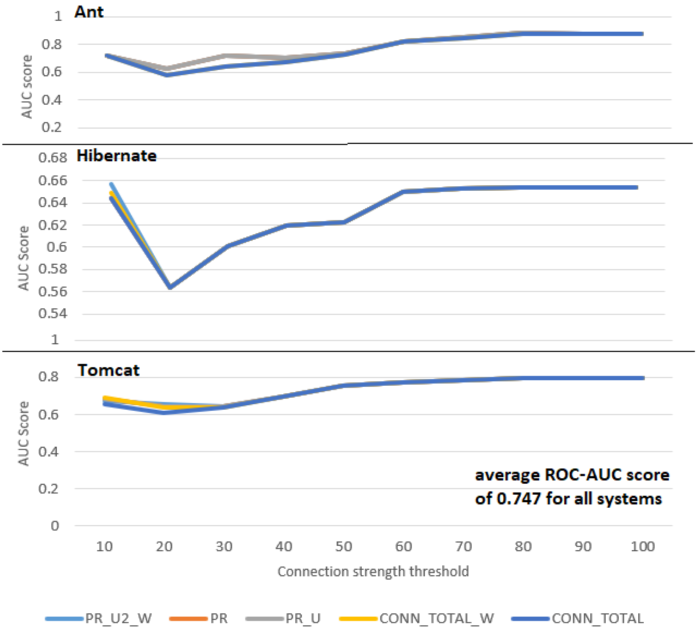
# Measurements using only logical dependencies

For this type of measurement, we add only the logical dependencies to the graph.

Table 4: Measurements for Ant using only logical dependencies

| Metrics | $\geq 10\%$ | $\geq 20\%$ | $\geq 30\%$ | $\geq 40\%$ | $\geq 50\%$ | $\geq 60\%$ | $\geq 70\%$ | $\geq 80\%$ | $\geq 90\%$ | $\geq 100\%$ | Baseline |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PR_U2_W | 0.720 | 0.627 | 0.718 | 0.703 | 0.732 | 0.824 | 0.852 | 0.881 | 0.876 | 0.876 | 0.929 |
| PR | 0.720 | 0.627 | 0.718 | 0.703 | 0.732 | 0.824 | 0.852 | 0.881 | 0.876 | 0.876 | 0.855 |
| PR_U | 0.720 | 0.627 | 0.718 | 0.703 | 0.732 | 0.824 | 0.852 | 0.881 | 0.876 | 0.876 | 0.933 |
| CON_T_W | 0.722 | 0.581 | 0.644 | 0.676 | 0.727 | 0.819 | 0.842 | 0.874 | 0.876 | 0.876 | 0.934 |
| CON_T | 0.722 | 0.581 | 0.644 | 0.676 | 0.727 | 0.819 | 0.842 | 0.874 | 0.876 | 0.876 | 0.942 |

# Measurements using only logical dependencies



# Comparison with fan-in and fan-out metric

### Definition

- The fan-in of entity A is the total number of entities that call functions of A.
- The fan-out of A is the total number of entities called by A.

# Comparison with fan-in and fan-out metric (cont.)

By looking at the comparisons between FAN IN, FAN OUT, FAN TOTAL, and the logical dependencies in which a class is involved we could not determine a direct connection between them.

Table 5: Top 10 LD measurements for Ant.

| Nr. | Classname | FAN_IN | FAN_OUT | FAN_TOTAL | LD_NUMBER |
|---|---|---|---|---|---|
| 1 | Project | 191 | 23 | 214 | 157 |
| 2 | Project$AntRefTable | 1 | 2 | 3 | 157 |
| 3 | Path | 39 | 13 | 52 | 147 |
| 4 | Path$PathElement | 3 | 2 | 5 | 147 |
| 5 | IntrospectionHelper | 18 | 24 | 42 | 143 |
| 6 | IntrospectionHelper$AttributeSetter | 8 | 1 | 9 | 143 |
| 7 | IntrospectionHelper$Creator | 3 | 5 | 8 | 143 |
| 8 | IntrospectionHelper$NestedCreator | 7 | 1 | 8 | 143 |
| 9 | Ant | 2 | 15 | 17 | 136 |
| 10 | Ant$Reference | 3 | 1 | 4 | 136 |

# Conclusions

The advantage of using *only logical dependencies* in key class detection is that it only uses data extracted from the versioning system and can be generalized to various programming languages.

- by using both dependencies combined, we can obtain a slightly better ROC-AUC score than the one obtained by the baseline approach
- by using only logical dependencies we obtain comparable results with the ones obtained by other researchers

| | Thung et al. | Osman et al. | Baseline | Current approach (SD+LD) | Current approach (LD) |
|---|---|---|---|---|---|
| ROC-AUC score | 0.825 | 0.750 | 0.894 | 0.926 | 0.747 |

- the logical dependencies number can be used complementary with fan-in and fan-out metric.

Ioana Şora and Ciprian-Bogdan Chirila.
Finding key classes in object-oriented software systems by techniques based on static analysis.
*Information and Software Technology*, 116:106176, 2019.

M. H. Osman, M. R. V. Chaudron, and P. v. d. Putten.
An analysis of machine learning algorithms for condensing reverse engineered class diagrams.
In *2013 IEEE International Conference on Software Maintenance*, pages 140–149, 2013.

Ferdian Thung, David Lo, Mohd Hafeez Osman, and Michel R. V. Chaudron.
Condensing class diagrams by analyzing design and network metrics using optimistic classification.
In *Proceedings of the 22nd International Conference on Program Comprehension*, ICPC 2014, page 110–121, New York, NY, USA, 2014. Association for Computing Machinery.

Andy Zaidman and Serge Demeyer.
Automatic identification of key classes in a software system using webmining techniques.
*Journal of Software Maintenance and Evolution: Research and Practice*, 20(6):387–417, 2008.