# Creating evolving project data sets in software engineering

Tomasz Lewowski and Lech Madeyski

Wrocław University of Science and Technology
Faculty of Computer Science and Management
Department of Software Engineering

11 September 2019



Politechnika Wrocławska

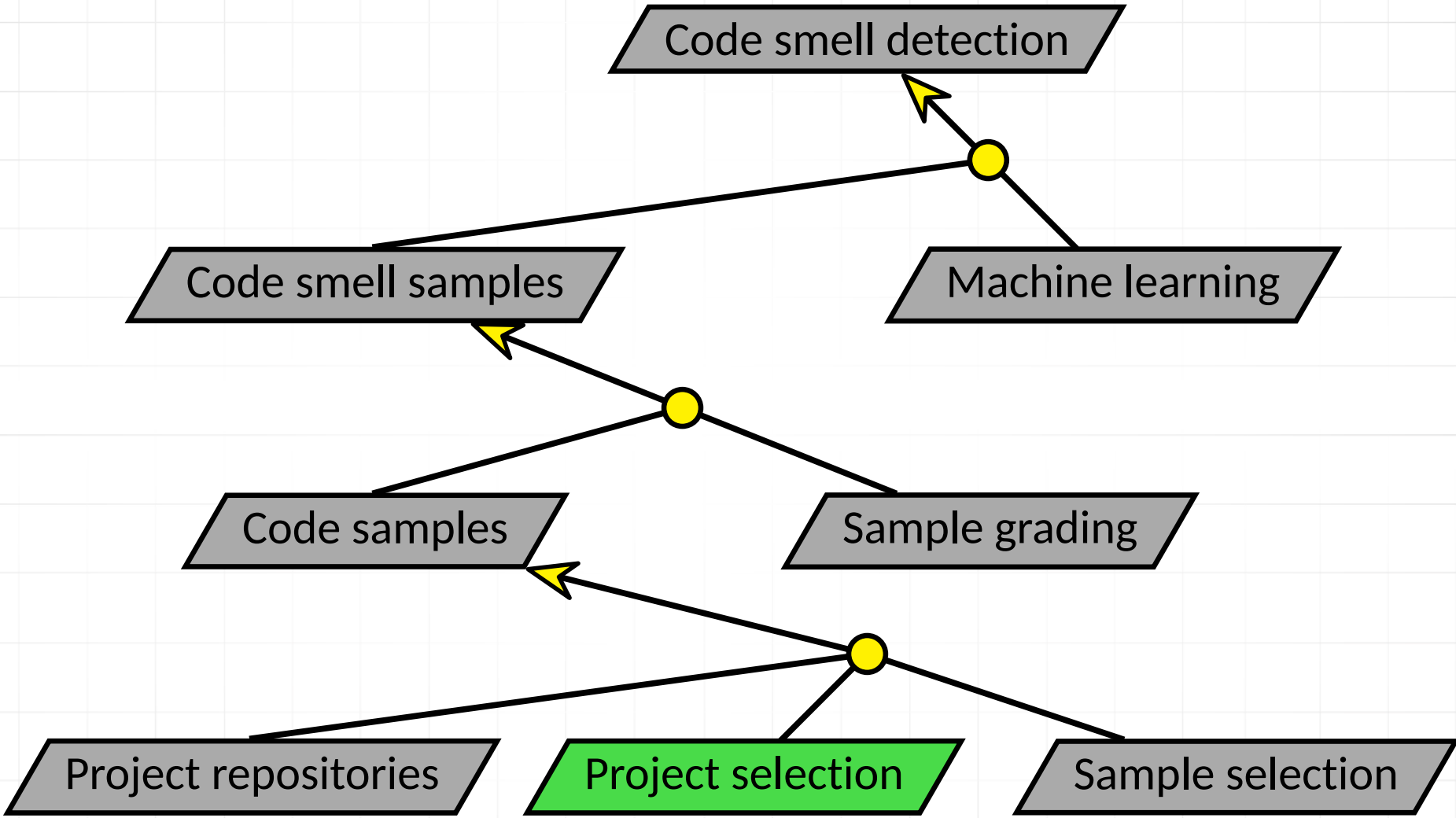
# Acknowledgement

This work has been conducted as a part of research and development project POIR.01.01.01-00-0792/16 supported by the National Centre for Research and Development (NCBiR).
We would like to thank Tomasz Korzeniowski and Marek Skrajnowski from *code quest sp. z o.o.* for all of the comments and feedback from the real-world software engineering environment.

# Agenda

1. Research context
2. Software evolution
3. What are the options?
4. Reproducibility

# Research context



# Evolution of software environments

- new C++ standard every 3 years,
- new LLVM major version each year,
- new ECMA Script edition each year,
- new Java release every 6 months,
- new Golang minor version every 6 months

# Data set lifetime to first published reference

- preparing publication with data set,
- peer-review,
- publishing,
- preparing publication that refers data set,
- peer-review,
- publishing

Total: **12+ months** to first published reference

2 Java releases, 1 ECMA Script version, 1 LLVM version

# Automation

There are places where automating data acquisition is not possible—these are the ones that require explicit human input.

What about others? Why publish explicit data set instead of the procedure used for its acquisition?

# Cons of automation

- paradigm switch,
- code needs to be shared (either maintained or will eventually go out of date),
- dependence on external databases,
- obtaining data set becomes harder,
- concept of data set is less clear, as subsequent researchers work on effectively different data, and only acquisition technique is preserved,
- search criteria need to be thought through in detail.

# Pros of automation

- **data sets are up-to-date with industry trends (concept drift)**,
- code needs to be shared—can be peer-reviewed and checked for correctness,
- no need to host large files,
- no licensing issues for data set storage,
- search criteria are thought through.

# Types of conditions

- popularity (**stargazers, watchers, forks**, dependants...),
- maturity (**project age, number of commits**, documentation quality),
- scale (**code base size**, repository size, number of committers, number of commits),
- activity (**number of recent commits**, number of active committers, number of dependants)
- relevance (**language**, frameworks)

All numeric attributes should have a cutoff based on percentile, not on fixed value. We selected a total of 792 projects.

# Industry-relevance

- documentation of reasonable quality,
- automated installer or detailed instructions,
- bug tracker or support list,
- an actual project, not set of samples, exercises etc.

517 projects fulfilled all requirements and 118 fulfilled them partially, which yields precision between 0.65 and 0.80

# Reproducible research

- Data sets used in specific publications still need to be included for reproducibility,
- they are immutable snapshots of results for given time, with no external dependencies,
- their goal is only to provide reproducibility, not to be used in further studies (except maybe benchmarks)

# Conclusion

- Publishing is too slow to keep up with creating data sets for software engineering, if the data sets are to represent current state-of-the-art,
- whenever data set can be generated automatically—e.g. by querying GitHub, GitLab, BitBucket, Jira or any other online service—querying script should be the preferred form of data set,
- used snapshots still should be published together with papers to for reproducibility