2. Duplicate Code
(Last Update: 2024-02-20)

The purpose of this laboratory session is to present tools for duplicate code detection. The removal or not removal of duplicated code is a decision for an expert. Although "Detecting Duplicated Code" has its own chapter in the book (OORP, p.223), you should also note "Setting Direction" chapter (OORP, p.19) as well. When dealing with duplicate code we often need to deal with the patterns "Fix Problems, Not Symptoms" (OORP, p.33), and "If It Ain't Broke, Don't Fixed it" (OORP, p.35). Duplicate code can be the symptom of the programming practices adopted by the developers; therefore, it might be better to address them before trying to remove the duplicates. Moreover, a duplicated code that is stable and not prone to change, may not need to be removed (or maybe its costs for removal do not worth it).

When is code duplication okay? If duplicating something hits a deadline and not duplicating does not, then one would rather deliver today and fix the tech debt tomorrow. One development paradigm where code duplication is preferred over abstraction is clone-and-own.

In clone-and-own development, a new variant of a software system is created by copying and adapting an existing variant (e.g., using the branching or forking capabilities of a version control system). This way, new variants are created ad-hoc and without requiring upfront investments. However, with an increasing number of variants, development becomes redundant and maintenance efforts rapidly grow. For example, if a bug is discovered and fixed in one variant, it is often unclear which other variants in the family are affected by the same bug and how this bug should be fixed in these variants. A more systematic way of developing variants is through software product lines, which consists of a set of similar software products with well-defined commonalities and variabilities. The-software-product-line strategy easily scales with many variants but is often difficult to adopt as it requires a large upfront investment of time and money. However, developers frequently create similar software products employing ad-hoc reuse of clone-and-own instead due to its inexpensiveness, flexibility, and developer independence. On social coding platforms like GitHub clone-and-own is prevalence mainline because of its inexpensiveness, flexibility, and developer independence.

Sample Projects and Tools

Projects
- pacman-python (https://github.com/kilincceker/pacman-python-ua-sre)
- django CMS (https://github.com/django-cms/django-cms)

Clone Detection Tools
- Text-Based: NiCAD (https://www.txl.ca/txl-nicaddownload.html)
- Token-Based: PMD's CPD (https://docs.pmd-code.org/latest/pmd_userdocs_cpd.html, https://pmd.github.io)
- Syntax-based: Clone Digger (https://clonedigger.sourceforge.net)
- For Other Tools and Resources
  - Valerio Maggio, "Clone detection in Python", SlideShare, 27 June 2013. (https://www.slideshare.net/valeriomaggio/clone-detection-in-python - Last Accessed: 9.2.2024)
  - Andrew Walker, Tomas Cerny, and Eungee Song. 2020. Open-source tools and benchmarks for code-clone detection: past, present, and future trends. SIGAPP Appl. Comput. Rev. 19, 4 (December 2019), 28–39. https://doi.org/10.1145/3381307.3381310. (https://dl.acm.org/doi/10.1145/3381307.3381310)
  - Yuekun Wang, Yuhang Ye, WeiWei Zhang, Yinxing Xue, Yang Liu, "Comparison and Evaluation of Clone Detection Techniques with Different Code Representations", 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 2023, pp. 332-344, doi: 10.1109/ICSE48619.2023.00039. (https://ieeexplore.ieee.org/abstract/document/10172753)

Task 1: Automated Small Scale Clone Detection

Download and install a text-based clone detection tool. A sample tool is listed above. You can use another one if you wish. Execute the tool and try it on one or more sample projects. Try to understand how the tool works.

What do you think? Does it identify the duplicates correctly?
Can you determine any refactoring candidates based on the outputs?

You can also use other tools of the same type to make comparisons.

Task 2: Large Scale Clone Detection

Now try a more sophisticated tool. Instead of a text-based one, use a token-based one. Once again, an example tool is given above.

What information do you have in the output?
How many clones are reported?
What types of clones does the tool detect?
Can you determine any refactoring candidates?
What types of clones are clearer and easy candidates for refactoring?

Task 3: Tweaking the Parameters on Clone Detection

Now, run the tools with different parameters. Refer to the documentations if needed.

How does the tool output change when the parameters are changed?
How do you get more/fewer false positives?
You are on the lookout for clones, which can be easily refactored. Which parameter values seem to lead to these clones?

Task 4: Comparing Clone Detection Tools

Compare and contrast the tools' usage and outputs.
How does the tools you used compare to each other?
Which tool finds the most clones?
Which tool seems the most useful?

You can refer to ICSE 2023 paper above (and the references therein) to learn about different comparison criteria.

Optional Task 1: Clone Detection on Other Projects

Now, run the clone detection on another project. Adjust the parameters as necessary.

Was it important to adjust the parameters to find clones?
How many clones do you find? Compare with your finding from the previous project(s).
Is it necessary to remove all the clones you found?

Optional Task 2: Clone Detection on other Large Systems

There are also large projects for you to experiment with. They are coded in different languages. Use the existing tools if possible or find alternatives to analyze them.

- MegaMek (https://github.com/MegaMek/megamek) (Java, Strategy Game, https://megamek.org)
- PostgreSQL (https://www.postgresql.org/docs/current/git.html) (C, Relational DB, https://www.postgresql.org)

- Quake 3 Arena (https://github.com/id-Software/Quake-III-Arena) (C, FPS, https://www.idsoftware.com)

Discussion and Conclusions

In this session, we used duplicate code detection tools to find code clones. Be sure to check the chapter "Detecting Duplicated Code" (OORP, p.223) for more information. As a final discussion let us consider the following questions.

Detection Quality:

- How many false positives (clones which you as a programmer would not name as such) have the detector found?
- Where the options offered by the detector enough to get to the true positives?
- Did you feel that the options also removed some true positives?

Reengineering:

- What were the reasons you could not refactor some clones?
- Could a tool detect the characteristics that are detrimental to the refactorability of clones?

Tool Support:

- What are the shortcomings of the tools you used?
- What feature do you miss most?

Duplication Awareness:

- If you look at your own code, would you find clones?
- Will you pay more attention to duplication in your own programming in the future?