Large Patch Integration among Divergent Variants

Re-using whole repositories as a starting point for new projects is often done by maintaining a variant fork parallel to the original. However, the common artifacts between both are not always kept up to date (in terms of patches). We define a patch as a pull request that fixes a bug, which we identify using keywords (e.g., resolves, fixes) from the pull request title. As a result, patches are not optimally integrated across the two repositories, which may lead to sub-optimal maintenance between the variant and the original project. A bug in both repositories can be patched in one but not the other (we see this as a missed opportunity) or manually patched in both, probably by different developers (we see this as effort duplication). In a recent study, we analyzed 364 (source->target) variant pairs and discovered 1,116 cases of effort duplication and 1,008 cases of missed opportunities [1]. In another preliminary study, we observed patches that have many changes in terms of number of commits, code churn, number of file changes, number of review comments, length of discussion, and time to merge, pose engineering challenges in terms of frequent merge conflicts when one tries to integrate them in the target variant. In another study [2], the authors report large pull requests in terms of the number of commits, code churn, and the number of file changes usually have at least one or more code refactorings.

In this study, we would like to explore avenues for solving the engineering challenges caused by merge conflicts while integrating the patches by following steps:

1. Record the merge conflicts
2. Detect and simplify refactorings with the help of RefactoringMiner [3]
3. Invert refactorings with the help of RefMerge [5]
4. Cherry-pick the patches
5. Replay the refactorings. RefMerge [5]
6. Detect refactoring conflicts
7. Perform regression testing in the target variant after applying the patch.
8. Measure the effort involved in integrating the patch before (step 1) and after applying the refactorings to the target (step 6).

Required Skills

Good in any programming and willing to adapt to any new programming language (Java, Python)

References

1. Poedjadevie Ramkisoen, John Businge, Brent van Bladel, Alexandre Decan, Serge Demeyer, Coen De Roover, and Foutse Khomh. PaReco: Patched Clones and Missed Patches among the Divergent Variants of a Software Family. FSE 2022.

2. Flávia Coelho, Nikolaos Tsantalis, Tiago Massoni, Everton L. G. Alves. An Empirical Study on Refactoring-Inducing Pull Requests. ESEM 2021
3. Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. RefactoringMiner 2.0. TSE 2020. https://github.com/tsantalis/RefactoringMiner/tree/intellij-psi
4. https://sarahnadi.org/resources/pubs/MahmoudiSANER19.pdf
5. https://arxiv.org/pdf/2112.10370.pdf
6. https://dl.acm.org/doi/pdf/10.1145/3377813.3381362
7. https://dl.acm.org/doi/pdf/10.1145/3530786