Title: “A Comparative Study of Automated Refactoring Tools”

Authors: Maha Alharbi & Mohammad Alshayeb

Publication Date: February 1st, 2024

URL: https://ieeexplore.ieee.org/abstract/document/10418470

Group: IEEE Xplore

Group Member: Greta Leege 100559009

Summary: Due to the large number of automated refactoring tools, developers are struggling to decide upon the right one to use, which was the purpose of this article. The article compared MultiRefactor, JDeordorant, jSparrow, and Spartenizer using five-open source projects. They compared these tools based on tool specifications, approach specifications, approach performance, and tool useability. Their evaluation led to jSparrow outperforming the other refactoring tools because of its large amounts of both qualitative and quantitative features, which means it is preferred for broad projects. As well, jSparrow was the most effective at removing smell for all projects. The least preferred tool was the Spartenizer as it continued to produce code smells after refactoring. It is said that future evaluation could focus on a larger pool of refactoring tools or AI-driven approaches to broaden this scope. This article is important for us as students, specifically in this class, as we are now learning to manually refactor. If we are working in coding jobs in the future we may feel inclined to move towards automated refactoring, due to the lengthy process refactoring currently presents. Understanding this article, shows how important it is to critically evaluate online automated refactoring tools and to make sure one understands both their failings and strengths.

Title: “Code Refactoring for Software Reusability: An Experimental Study”

Authors: [Abdullah Almogahed](https://ieeexplore.ieee.org/author/37089589021); [Hairulnizam Mahdin](https://ieeexplore.ieee.org/author/37669678400); [Mazidah Mat Rejab](https://ieeexplore.ieee.org/author/37089281123); [Abdulwadood Alawadhi](https://ieeexplore.ieee.org/author/37089658761); [Samera Obaid Barraood](https://ieeexplore.ieee.org/author/37090033823); [Manal Othman](https://ieeexplore.ieee.org/author/731513949911929)

Publication Date: August 6th, 2024

URL: https://ieeexplore.ieee.org/abstract/document/10638872

Group: IEEE Xplore

Group Member: Greta Leege 100559009

Summary: This paper viewed ways refactoring techniques enhance reusability, to give developers more insight. The paper looked at five difference refactoring techniques including extract interface, extract class, inline class, inline method, and encapsulate field refactoring. It showed that extract interface, extract class, and encapsulate field refactoring raise code reusability, while the others damage it. The reason inlining of classes and methods hurts reusability is due to the fact that they increase the complexity and reduce the reusable software parts because the codebase is less modular. This paper showed techniques that can be used to bring about reusability, which can be incredibly helpful in large code building and partner programming which we focus on in this class.

Title: Semantic Code Refactoring for Abstract Data Types

Authors: [Shankara Pailoor](https://dl.acm.org/doi/abs/10.1145/3632870), [Yuepeng Wang](https://dl.acm.org/doi/abs/10.1145/3632870), [Işıl Dillig](https://dl.acm.org/doi/abs/10.1145/3632870)

Published By: Association for Computing Machinery

Date of Publication: 05 January 2024

Website: <https://dl.acm.org/doi/abs/10.1145/3632870>

In the article, the authors introduce a novel approach for automating the refactoring of ADTs by using a combination of counterexample-guided inductive synthesis and symbolic reasoning. This method takes an original ADT implementation and a relational invariant connecting the old and new data representations, and then automatically synthesizes an equivalent refactored version. The paper's approach is highly relevant to the principles outlined in our EG3 statement, as both emphasize the importance of code clarity, maintainability, and systematic design improvements. For example, the EG3 statement discusses addressing code smells such as mysterious names, duplicated code, long functions, and large classes by applying refactoring techniques like renaming, function extraction, and class extraction, which aligns with the paper’s goal of simplifying code structure while preserving functionality. Furthermore, the use of automated tools like Revamp in the article reflects the emphasis in EG3 on continuous testing and documentation with each incremental change—making sure that the refactored code passes all tests, is easy to understand, and is committed to version control step by step. Both the paper and the EG3 statement promote a disciplined and methodical approach to improving code through refactoring, while also incorporating design patterns (such as Singleton) to ensure architectural simplicity and reliability in software development​

Title: Automating Source Code Refactoring in the Classroom

Authors: [Eman Abdullah AlOmar](https://dl.acm.org/doi/abs/10.1145/3626252.3630787), [Mohamed Wiem Mkaouer](https://dl.acm.org/doi/abs/10.1145/3626252.3630787), [Ali Ouni](https://dl.acm.org/doi/abs/10.1145/3626252.3630787)

Published By: Association for Computing Machinery

Date of Publication: 07 January 2024

Website: <https://dl.acm.org/doi/abs/10.1145/3626252.3630787>

The article discusses how teaching students to use automated tools like the JDeodorant plugin helps them recognize and fix design-level antipatterns—such as God Class and Feature Envy—in their code. Through hands-on assignments in undergraduate and graduate classes, students first attempted manual refactoring and then used the tool to assist in identifying and correcting antipatterns. The study found that students not only improved code quality and reduced complexity but also became more aware of the importance of good design principles and the practical benefits of automated refactoring. This experience strongly relates to the EG3 statement’s emphasis on practicing refactoring, improving code clarity, and applying basic design patterns, like the Singleton. Both the article and EG3 highlight the value of identifying code smells, updating code according to best practices, and maintaining thorough documentation through step-by-step commits. Additionally, the article’s focus on using automated tools to both educate students and enforce refactoring best practices mirrors EG3’s advocacy for continuous testing and iterative development, reinforcing the importance of writing maintainable, high-quality software.