

Live Software Inspection and Refactoring

Sara Fernandes^a, Ademar Aguiar^{a,b} and André Restivo^{a,c}

^aFaculty of Engineering, University of Porto, Porto Portugal

^bInstitute for Systems and Computer Engineering, Technology and Science, Porto, Portugal

^cArtificial Intelligence and Computer Science Laboratory, Porto, Portugal

Abstract

With the increasing complexity of software systems, software developers would benefit from instant and continuous guidance about the system they are maintaining and evolving. Despite existing several solutions providing feedback and suggesting improvements, many tools require explicit invocation, leading to developers missing improvement opportunities, such as important refactorings, due to lost of train of thought. Therefore, to address these limitations, we propose an approach where developers receive instant and continuous feedback about their software systems. This guidance would include the detection of code smells and the suggestion of refactorings to improve the system, justified by relevant software quality metrics related to the recommendations. This research aims to improve the experience of developing and maintaining software systems by providing a live environment for continuous inspection and refactoring of software systems, that is informative, responsive, and tactically predictive, and thus helping developers to identify and solve quality problems in a quicker and better way.

Keywords

Software and its engineering → Software maintenance tools

1. Introduction

Software systems are increasingly complex and extensive, and, as a result, approximately 75% of the costs associated with software development occur after finishing a particular system, or after inspecting it. Sometimes, simple tasks, like adding or modifying a feature, are generally hard and time-consuming due to the size and complexity of the system. Refactoring aims to mitigate existing code smells, usually a result of bad programming practices, without changing the current results while, at the same time, improving the maintainability and comprehensibility [1].

Several approaches are capable of detecting several code smells by analyzing specific software quality metrics. Also, there are already tools that suggest refactoring techniques to improve the overall quality of a program. However, the feedback provided to developers is not always detailed, fast enough, and optimal to effectively enhance the software systems. Then, we are exploring the introduction of liveness to enhance this guidance loop to provide real-time information that will help them change their programs beforehand so that when it comes to adding or modifying features, it will not be as hard for them as usually is [2].

The main objective of this research is to tighten the feedback loop associated with refactoring, making it immediate and continuous, during development, through the analysis of quality


8th SEDES, Software Engineering Doctoral Symposium, September 08–11, 2020, Algarve, Portugal

EMAIL: up201405955@fe.up.pt (S. Fernandes); aaguiar@fe.up.pt (A. Aguiar); arestivo@fe.up.pt (A. Restivo)

ORCID:



© 2020 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

metrics, detection of code smells, identification of refactoring opportunities, and visualization of metrics related to each recommendation. Therefore, we aim to increase, improve, and speed up the feedback given to developers regarding their software systems. The main research question is to find the best combination of features to combine and provide in a live development environment, so that (1) developers can become aware of existing code smells, (2) refactoring opportunities can be appropriately suggested to developers, together with their expected quality impact, without being intrusive, (3) the developer experience can motivate them to quickly fix code issues, without strictly needing to compile or execute the system under development.

2. State-of-the-art

Liveveness is the possibility to shorten the edit-compile-run loop by providing real-time information to developers about their systems. Bringing liveness to the development of software systems makes them easier and simpler to understand, visualize, maintain, and evolve [2].

In 1990 and later in 2013, Tanimoto proposed a hierarchy composed of several liveness levels. The first level is merely an informative level, where developers had access to information using different visual methods such as flowcharts. Level two was a more evolved level where developers needed to ask for a response from their systems, and shortly afterward, they could check the results obtained. In level three, computers would wait for programmers to stop developing their systems showing the results of the respective program automatically. In the fourth level, the computer did not wait for the developer to finish programming since it was executing the program while it was being created, instantaneously showing its results. In level five, the computer not only runs the software while it is being developed but also forecasts a group of actions that the programmer may be interested in, executing one or more versions of these predicted actions. Finally, in level six, the forecasted steps are strategically selected to cover the majority of the software behaviors [3].

Currently, there are some live tools like Quokka, which is a live visualization tool of Kaluta emulations (enterprise software emulator able to represent large-scale environments), enabling a developer to see the emulation's endpoints and the respective interactions with external software systems while they are occurring [4].

We also have the concept of *Code Smells*. A code smell is any bad implementation characteristic present in the source code of any software system that can eventually represent a more profound problem [1]. In contrast, *Refactoring* is the process of modifying the design of a software system to improve its quality and reduce the code smells, without changing the behaviors previously implemented. These kinds of actions make the program cleaner and self-explanatory [1, 5].

Tarwani et al. [6] used a *Greedy algorithm* to determine the optimal refactoring sequence to be executed in a program. Each refactoring produces a specific maintainability degree, and the Greedy algorithm tries to find the chain of refactorings that will lead to the higher maintainability value. It supports refactorings like “*Extract Method*”, “*Move Method*”, “*Extract Class*”, etc. Meananeatra [7] used a methodology to obtain the best refactorings to solve “*Long Method*” smells. He used a four criteria method, to select the optimal sequence, composed by (i) the number of code smells removed, (ii) the maintainability degree of the solution, (iii) the size of the created sequence, and (iv) the number of elements of the source code changed.

This approach includes refactorings such as “*Decomposed Conditional*”, or “*Replace Method with Method Object*”.

Although there are already advances in detecting code smells, identifying refactoring opportunities, and developing live tools, they are not well integrated yet. They do not capitalize on all the existing potential, such as giving immediate feedback and guidance to programmers. By combining these approaches, first, and to explore their synergies, after, we are convinced that it will be possible to tighten and improve the support provided to developers. They would have instant and continuous access to their code quality through the detection of specific smells and identification of refactoring opportunities, while coding, and they could choose the most appropriate actions based on the impacted metrics (i.e., before versus after refactoring).

3. Research objectives and methodological approach

This section describes all of our main research objectives, stating the hypothesis and research questions that we aim to answer with this project. Besides, we summarize the approach that we intend to use to develop and validate our solution.

3.1. Objectives

This project aims to tighten the refactoring feedback loop, making it instantaneous and continuous, while developers are programming. We expect to achieve this objective by analyzing software quality metrics and suggesting and applying refactoring methods to solve several code smells. Besides, we want to provide information about the evolution of specific metrics related to the refactoring recommended to complement the feedback given to the programmers.

Therefore, this research project is based on the hypothesis that “*A higher level of liveness combined with the constant exposure to refactoring suggestions can increase the overall quality of software systems and reduce their development time.*”

Based on the previous statement, the main research questions of our project are:

- RQ1** “*Are the identified refactorings capable of mitigating the code smells detected?*” We aim to verify if the refactorings suggested by our approach are able to extinguish the code smells detected on the source code.
- RQ2** “*When developers constantly receive refactoring suggestions, can they reach better programming solutions, with more quality?*” We aim to analyze if developers can converge to better programming solutions when they are constantly exposed to refactoring suggestions complemented by information about specific metrics that will be increased when a refactoring is applied.
- RQ3** “*When we apply a higher level of liveness in a refactoring suggestion approach, are developers able to shorten their total development time?*” We aim to verify if it is possible to reduce the development time when developers know which refactorings they should apply on their code.

3.2. Proposed Approach

Our research project will mainly follow the *Engineering Research Method* [8]. Therefore, we expect to develop a tool for VSCode capable of providing live feedback to developers about their software systems. This feedback will be presented as refactoring recommendations that aim to mitigate specific code smells implemented in the code. Besides, each refactoring will be complemented by information related to the quality metrics improved by its execution. Therefore, developers will be able to make more informed decisions when they are refactoring their programs.

The development of our solution will be divided into four implementation stages. In the first stage, we want to analyze a set of quality metrics that will be used to detect specific code smells on a project. To calculate each metric, we will analyze the program's abstract syntax tree and, when necessary, it will apply mathematical formulas to measure specific metrics like the volume of a method [9]. Then, in the second phase, we will analyze each metric previously calculated to detect specific code smells like "God Class" or "Long Method" [1]. To help in this task, we will create a catalog that associates the software quality metrics with possible code smells. Besides, we will try to apply some machine learning or data mining algorithms to make the detection mechanism more efficient and informed. After that, in the third main step, we will use the detected smells to suggest several refactoring techniques that aim to solve them. As it happens in the previous phase, we will create a catalog that will associate the code smells and metrics with the possible refactorings that can be applied. For each of the recommended refactorings, the developers will have access to the evolution of specific metrics to decide which is the best refactoring to be executed on their systems. In the fourth phase, we will try to increase the liveness level of this entire process, up to level five. With the introduction of this subject, the proposed tool will be capable of reducing the "edit-compile-run" cycle. Then, the feedback will be faster and more fluid. Thus, we will create a "live refactoring recommendation" tool that assists any programmer, since it allows implementing the best programming practices in a faster and more informed way.

We also expect to execute different validation tests, using the *observational* and *controlled methodologies* to validate the proposed approach. We expect to execute different experiments, using academic and industrial case studies, to achieve the main goals on the validation of our solution. Each participant will have to complete a list with several programming tasks, with and without the developed tool. We will then use hypothesis tests with the data collected to validate our research questions and the main hypothesis. Besides, we aim to create robotic developers, who guided by public git repositories, will use the proposed tool in both a full and semi-automated way to change code driven by commits and assess the expected quality with and without doing the recommended refactorings.

4. Past work and preliminary results

During the development of the master's thesis, we implemented a plugin for Visual Studio Code, for JavaScript and TypeScript programs, which analyzed several software quality metrics, while developers were programming their systems. In this project, we calculated metrics like the number of lines of code, cyclomatic complexity, or maintainability degree. The measurement of

each metric was done using the abstract syntax tree (AST) or specific mathematical formulas [9]. Then, the information about these values was given through bar charts on the IDE's side menu.

After developing that tool, we tried to validate our hypothesis by doing an empirical study with programming students. However, we used a very simple case study, with a low complexity degree, which did not allow us to collect enough relevant data to validate any of the research questions of that project or even its main hypothesis.

Regarding the current doctoral research project, we already analyzed the problem that this Ph.D. will solve. We proposed the respective, and we also studied the current state-of-the-art on the main topic of this project. Besides, we already started our tool by analyzing some software quality metrics that were not included in the master thesis' project.

5. Future work and expected results

As mentioned before, we already started developing the proposed tool, and currently, we are analyzing more quality metrics, and we are also cataloging the metrics and associated code smells.

As future work, after finishing the catalog, we expect to start the mechanism to detect code smells using the measured metrics and the developed catalog. We also intend to use machine learning and data mining algorithms to predict parts of the code that can suffer from some bad programming practices. Then, after detecting the code smells, we want to create a catalog that associates each code smell and respective metrics with the refactoring techniques that can solve them. We will use Fowler's refactorings catalog [1] for that. However, we also want to investigate more code smells and refactorings beyond the ones already known. After finishing this catalog, we intend to apply intelligent algorithms to suggest the optimal sequence of refactorings for a given programming context. In each refactoring indicated, we also expect to present the evolution of specific software quality metrics to complement the information provided to developers. After this last development step, we will implement the mechanism that will automatically execute a refactoring when a developer accepts the refactoring suggestion provided by our tool.

Next, we plan to confirm the main hypothesis using hypothesis tests with the data collected through the controlled experiment. In this experiment, we pretend to use academic and industrial case studies. Finally, in the last step, we intend to write the respective dissertation.

With this, we expect to achieve three different results. The first one is the state-of-the-art analysis, which aims to introduce the current approaches, techniques, or tools related to the most relevant topics of this thesis. Secondly, we aim to develop a live refactoring suggestion tool for Visual Studio Code that will allow developers to receive feedback – in the form of restructuring recommendations – about their software systems, while they are programming.

6. Conclusions

More and more, developers need to have support tools that help them understand and change their software systems. When developers have to maintain their programs, sometimes they

have great difficulty in doing precisely that, because their code is hard to read. Then, they have to refactor the code turning it more clean and self-explanatory, increasing its overall quality.

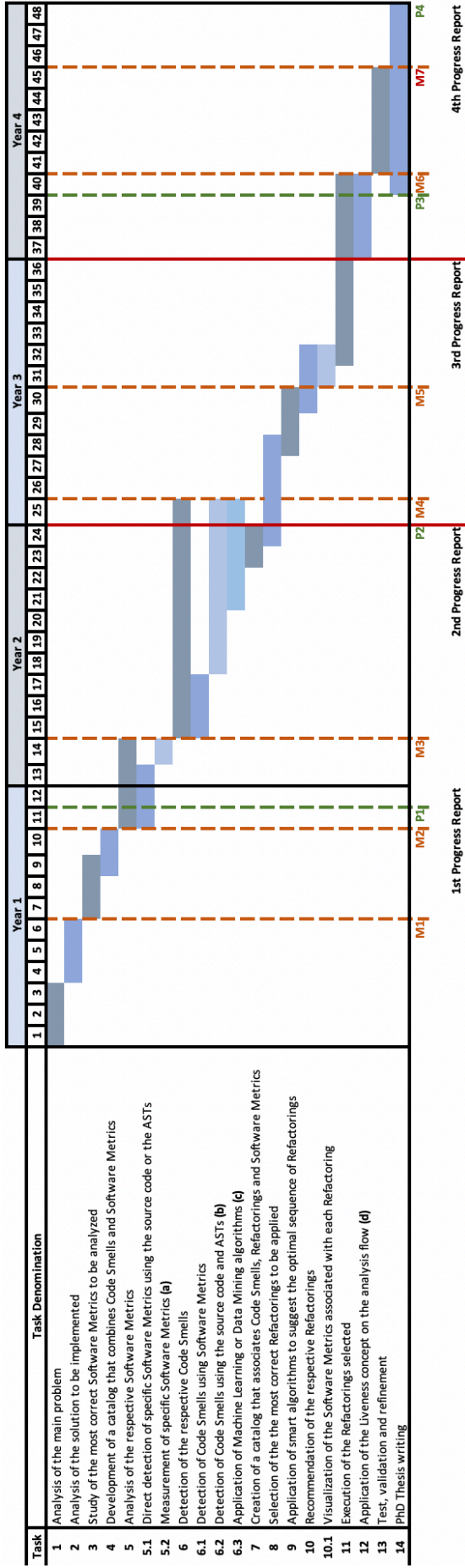
Therefore, we proposed a new tool capable of doing exactly that. This tool has its merit and respective impact, mainly regarding the software development cycle, since it focuses on the development and maintenance stages, inspecting the code to find possible problems. We think that this project has several benefits: (1) efficiency, as it is expected that through this approach the software development time could be reduced significantly; (2) automation, because the suggestions will be made automatically while the developer is programming; and (3) source code quality since it is expected that at the end, the source code would have better quality. We also believe that our approach has an impact in the area of Software Engineering, because (1) our approach helps developers to understand and correct their systems, if necessary, in early development stages; and (2) can be used in the industry, simplifying their systems that usually are complex, with millions of lines of code.

References

- [1] M. Fowler, K. Beck, J. Brant, W. Opdyke, don Roberts, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
- [2] A. Aguiar, A. Restivo, F. F. Correia, H. S. Ferreira, J. P. Dias, Live software development – tightening the feedback loops, in: *Proceedings of the 5th Programming Experience (PX) Workshop*, 2019, pp. 1–6.
- [3] S. L. Tanimoto, A perspective on the evolution of live programming, in: *Proceedings of the 1st International Workshop on Live Programming, LIVE '13*, IEEE Press, Piscataway, NJ, USA, 2013, pp. 31–34. URL: <http://dl.acm.org/citation.cfm?id=2662726.2662735>.
- [4] C. Hine, J.-G. Schneider, J. Han, S. Versteeg, Quokka: visualising interactions of enterprise software environment emulators, in: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012*, ACM Press, New York, New York, USA, 2012, p. 370. URL: <http://dl.acm.org/citation.cfm?doid=2351676.2351750>. doi:10.1145/2351676.2351750.
- [5] K. Cassell, C. Anslow, L. Groves, P. Andreae, Visualizing the refactoring of classes via clustering, *Conferences in Research and Practice in Information Technology Series 113* (2011) 63–72.
- [6] S. Tarwani, A. Chug, Sequencing of refactoring techniques by Greedy algorithm for maximizing maintainability, *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (2016)* 1397–1403. URL: <http://ieeexplore.ieee.org/document/7732243/>. doi:10.1109/ICACCI.2016.7732243.
- [7] P. Meananeatra, Identifying Refactoring Sequences for Improving Software Maintainability Categories and Subject Descriptors The Scope of Our Approach, *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (2012)* 406–409.
- [8] M. V. Zelkowitz, D. R. Wallace, Experimental models for validating technology, *Computer* 31 (1998) 23–31.
- [9] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*, Elsevier Science Inc., New York, NY, USA, 1977.

A. Work Plan

In this appendix, we present the Gantt Chart of our work plan with the idealized tasks and main objectives of our Ph.D. research project. We also describe the different tasks, milestones, and papers that we aim to publish about the results we achieve over this project.



Milestones

- M1 Milestone 1 - 6th month
- M2 Milestone 2 - 10th month
- M3 Milestone 3 - 14th month
- M4 Milestone 4 - 25th month
- M5 Milestone 5 - 30th month
- M6 Milestone 6 - 40th month
- M7 Milestone 7 - 45th month

Papers

- P1 Paper 1 about the relationship between Code Smells and Software Metrics
- P2 Paper 2 about the detection of Code Smells using the different techniques
- P3 Paper 3 about the recommendation and execution of Refactorings and its impact on the Software Metrics
- P4 Paper 4 summarizing the PHD project and thesis

(a) Measurement of specific Software Metrics using mathematical formulas

(b) Detection of Code Smells directly through the source code and ASTs, without using the Software Metrics previously calculated

(c) Application of Machine Learning or Data Mining algorithms to detect Code Smells, in order to validate the Smells previously detected by the other techniques used and also to detect even more Smells

(d) Application of the Liveness concept on the analysis of Software Metrics, Code Smells and Refactorings

B. Poster

In this appendix, we present the poster that summarizes our Ph.D. research project. There, we give an overview of our motivation, main objectives, and proposed approach.

Live Software Inspection and Refactoring

Motivation

- Besides developing software systems, developers need to **maintain** them;
- The **effort** and **time** needed to **understand** or **maintain** a complex software system are very **high**;
- Development environments already give some **assistance** to developers;
- But they could provide **better real-time feedback** and **support**.

Objectives

Increase, improve, and speed up the feedback about the **quality of the code design**, given to developers regarding their software systems:

- Helping developers to make **better** decisions about the code design;
- **Increasing** the source code quality;
- **Speeding-up** the convergence to a good solution.

Background

Liveness

Real-time feedback that shortens the edit-compile-run cycle [1]

Software Metrics

Measurement of specific software characteristics [2]

Code Smell

Surface indicator that usually corresponds to a deeper problem in the system [3]

Refactoring

Changes made in a software system, without modifying the behavior implemented [3]

Live Inspection and Refactoring

*A higher level of liveness combined with the **constant** exposure to **software quality metrics** and **refactoring suggestions** can increase the overall **quality** of software systems, and reduce their **development time**.*

- **RQ1:** Are the identified refactorings capable of mitigating the code smells detected?
- **RQ2:** When developers constantly receive refactoring suggestions, can they reach better programming solutions, with more quality?
- **RQ3:** When we apply a higher level of liveness in a refactoring suggestion approach, are developers able to shorten their total development time?



- **VSCode Extension** that analyzes **software metrics**, in real and programming time, to detect specific code smells;
- Then, it suggests **refactoring** techniques to solve the **code smells**, providing information about the evolution of particular **metrics**.

Past Work

- Development of a tool for **VSCode** that analyzes **quality metrics** in real and programming time;
- It also detects "*Extract Variable*" and "*Split Variable*" refactoring opportunities.

Conclusions

- We propose a solution capable of providing **live feedback** to developers about their software systems;
- With this software developers will be able to improve the **overall quality** of their software systems.

Future Work

- Development of the proposed tool for **VSCode**;
- Empirical validation of the **hypothesis**, using **academic** and **industrial** case studies.

References

- [1] Steven L. Tanimoto. Viva: A visual language for image processing. Journal of Visual Languages and Computing, 1(2):127 – 139, 1990.
- [2] Maurice H. Halstead. Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc., New York, NY, USA, 1977.
- [3] Kent Beck, John Brant, William Opdyke, Martin Fowler and don Roberts. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.