EXAMINATION OFFICE
IU.ORG

INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

RESEARCH ESSAY

Tasks for Course: ISSE01_E – Seminar: Software Engineering

Task 2: Clean Code and Refactoring

How can an organization ensure that Clean Coding is applied in practice / organization?

Date: 03.10.2024

Author’s name: Denys Novakov

Matriculation number: IU14074702

Tutor’s name: Carsten Skerra

Table of contents

Nowadays, the IT market faces continuously growing requirements and innovations, which cause the growth of organizations, the widening of software teams, and respectively the increase of the software size and complexity. According to the current trend and modern needs, it's crucial for an organization to create scalable, high-quality, and easily maintainable software solutions.

In the real world, where the software size is steadily growing, new technologies are appearing, but the deadlines remain at the same place, the willingness to sacrifice code quality in order to achieve some stated goals in schedule or to implement some new features can exist. But such an approach, which can probably help to finish some part of the software within the schedule can significantly affect further development. It leads to many issues that appear in subsequent development stages, such as code redundancy, poor code reusability, decreased readability, increased bugs, and many others.

Another is cooperation within a team. Modern development teams normally consist of many different experts responsible for a certain domain of software development. That fact makes it vital for an organization to organize the coding process in a way, that each participant can clearly understand the code and feels comfortable working with it. Otherwise, it can lead to slower development and a negative impact on team morale, since it's not really comfortable to work with the messy code.

But how can an organization address these development issues? To create understandable, high-quality code, that can be easily extended and maintained in the future, an organization must establish consistent clean coding and refactoring practices. Mens and Tourwé's survey of software refactoring emphasizes the importance of the systematical usage of these practices to enable the appropriate software development process. The implementation of consistent clear coding practices is sometimes very challenging for an organization due to many reasons. Different expertise levels of developers, software complexity, and deadlines complicate the consistent usage of these practices.

This essay is aimed to research different methods how can an organisation ensure the consistent application of clean coding and refactoring within a project, which automated tools and metrics can be used for this purpose, and how to estimate the impact of applied clean coding and refactoring practices.

To answer the research question, let’s dive deeper into the concept of clear coding and refactoring and understand its main principles, purpose, and properties.

Martin Flowler’s book “Refactoring: Improving the Design of Existing Code” is a fine source to gain a better understanding of how clean coding and refactoring work. Flower defines clean coding as the practice of systematical code modifications, that are aimed to make it more understandable for other people, to improve its extendability and maintainability in the future, without changing its behavior and functionality. He emphasizes, that the systematical refactoring and improvements in the code are equally important with its functional behavior.

In general, clean coding is the approach, that represents a set of rules, practices, positive code patterns, and code antipatterns, naming rules, which serve to enhance the code quality and enable effective software development. The one important aspect, that must be pointed out, is that clean coding is not just a design pattern, which is established at the beginning of the project, but rather the iterative process, where the appropriate code quality is achieved through small incremental changes, that happen during the entire project.

Since the organization can have peculiarities related to the development process, these rules can be considered as a basis for acceptable code quality. However, this set of rules and code patterns is not strictly defined and can be specifically adapted for current needs and requirements defined for a certain project.

Martin Flowler provides in his book a comprehensive set of best practices for clean coding, which serves as a basis for successful code improvements, but he also underscores that the application of these code patterns may vary depending on the project concept, boundaries, objectives of the development team, available technologies and the entire software development landscape. Moreover, in the modern industry, the development team and project requirements can easily be changed during the project, which requires clean coding practices to be flexible.

The flexibility of the clean coding enables effective decision-making within the project team and helps the organisation to prioritize, and modify the rules, to make them the most suitable for the current project situation.

After the terms, clean coding and refactoring are clearly defined, and all their properties and abilities are recognized, let’s understand in more detail, which benefits can an organization obtain by supporting the consistent application of clean coding and refactoring.

The primary advantage associated with the efficient application of clean coding is the increased maintainability and scalability of the software. By adhering to the stated rules, practices of clean coding, and implementing constant code refactoring, the organization can reach the appropriate code quality, which enables the efficient maintenance and extension of created software in the future.

Furthermore, well-established code refactoring can help an organization by means of business. The high quality of code can sufficiently decrease the cost for implementing some changes, which appear during the project and for adding some new functionalities or features since the work with the bad-structured and messy code can bring many unexpected costs, and delay the development.

Another impact is that the application of clean coding ensures less quantity of bugs and issues in the source code. Khomh and Gueheneuc's work (2009) illustrates, that the software created under clean coding rules and refactoring, will inevitably contain fewer issues and will have enhanced stability, which means, that fixing some bugs will require less expense.

In addition, the modern software development trends show, that the perisistent staff flow is the essential practice for an enterprise. Sometimes, during the project, it becomes clear that the development team lacks the expertise in some domain to implement some part of the software. In this case, some new specialists must be involved in the project. These specialists must be provided with high-quality code, to avoid unnecessary time expenses for familiarization and understanding how the project is structured.

Moreover, if the core developers of the project retire, their followers must have a well-structured code base to continue the development process without any obstacles.

But how can an organization ensure that clean coding practice and refactoring activities are established at enough level within the project and bring all listed benefits? And how can an organization overcome all challenges that are related to their efficient usage?

In the next part of the essay different methods and practices for efficient application of clean coding and refactoring, measuring the results of their application, and assessing possible changes in the future will be researched.

The first challenge associated with the methodical implementation of clean coding and refactoring is the different expertise levels within the development team. In the modern enterprise, it’s common practice, that the development team consists of many programmers with different expertise levels. Some of them can be responsible for the execution of less important or routine tasks, while some more experienced developers can implement more valuable and game-changing features.

Despite this fact, to keep an efficient clean coding application, an organization must maintain a uniform level of awareness and understanding of clean coding rules established for a certain project. Each participant must gain a clear understanding, of how the code is structured, regardless of the level of the programmer, who wrote it. Otherwise, the organization will face many communication issues and misunderstandings within a team, which can postpone the project release or increase the estimated budget.

But how can an organization achieve this goal?

An organization can implement training sessions, workshops, or seminars, where more skilled developers can share their experience of clean coding and refactoring to supplement the knowledge of other developers with practical skills. Today’s market offers a lot of frameworks and good practices for conducting such events. The one important aspect, that must be considered, is that these training sessions or seminars must be conducted consistently and must be an inherent part of the development process.

Vasileva & Schmedding (2016) in their work illustrate, that this educational process is not a one-time activity, but rather a set of consistent iterations, that enhance the level of knowledge of developers, and must consider all emerging technologies, tools, and methods, which support the application of clean coding and refactoring.

Another detail, that can sufficiently increase the code quality and efficiency of refactoring is the right time scheduling. Sometimes, due to the short deadlines, the project governance can prioritize the implementation of some core features, that are crucial for the entire project, sacrificing the conducting of refactoring, but such an approach can lead to poor code quality that will affect the future development. Therefore, it’s very important to dedicate enough time for code refactoring activities, when the project schedule is created and agreed upon by the stakeholders.

Another practice, that can supplement the conduction of seminars and workshops and enable the right usage of clean coding rules and the efficient conduction of the refactoring is code reviews.

Code reviews are a powerful mechanism, which enables continuous learning and consistent collaboration within the development team. Systematical conduction of code reviews after each phase of the development gives an opportunity for less experienced developers to gain some knowledge from more seasoned specialists. It allows to improve the overall expertise level of the development team and to create a clean coding culture among developers. According to Mens & Tourwé (2004), collaborative code reviews can sufficiently help to approve the changes in the source code and ensure their alignment with the established clean coding rules, which underscores the importance of their conduction.

Moreover, code reviews are not only considerable for knowledge sharing. Consistent code reviews can support bug fixing in the source code. In Flowler's work (1999) he declares that code reviews can help to identify bugs at early project stages. This approach can reduce the costs and efforts, required to fix some issues and help to stay within the approved budget for the project implementation.

But all considered approaches require a lot of resources for implementation. In large projects, which consist of thousands of lines of code and have extensive infrastructure it's not so easy to conduct code reviews for each written piece of software, to support the consistent conduction of workshops, and seminars, involve all developers to attend them, and organize them in a such way as to receive the maximum outcome from their conduction.

To address this problem, the IT market offers a lot of automated tools and software solutions as static code analyzers or refactoring tools, which can assist in controlling the quality of the source code. These tools provide real-time feedback to the developer, whether the code is aligned to clean code conventions, identify potential issues in the source code as early as possible, and suggest some possible improvements, which can further improve the overall code quality.

In the next section of the essay, we will dive deeper into the different types of these automated solutions, how they can impact on the refactoring activities and maintenance of the clean coding standards, and their application scenarios.

A lot of automated tools, that can be useful to simplify the routine tasks related to refactoring and clean coding are available in today’s IT market. Typically, they act at different stages of the project abilities and provide different benefits for a certain development aspect. Although the market obviously provides many custom solutions, this software can be classified.

The first type of automated software, which can be used for checking the alignment of the source code with clean coding rules is static code analyzers. Their main purpose is to automatically review the written code in real-time and identify unacceptable code patterns and potential issues, which can affect the development process in the future. Their application is especially profitable for extensive projects since they don’t require the execution of code to perform a code review. Furthermore, by applying static code analyzers within the development environment, an organization can create different metrics, to evaluate the quality of code and the result of conducted activities, aimed to enhance the alignment to clean coding practices and enable further decision-making.

Vasileva & Schmedding (2016) in their research have proven the positive effects of the usage of tools for static code analysis. They used the static code analyzer during the project development and produced metrics, which describe certain code aspects to evaluate how the code quality has changed after each project iteration. They emphasize, that long-term integration and application of measurement tools enhance the overall code quality and enable the right decision-making.

The second type of software is refactoring tools. They serve to simplify the tasks associated with the improvement of the internal code structure and help the developers to save the time, needed for manual refactoring, enabling them to dedicate more time to more valuable and creative project activities. Many modern integrated development environments such as Visual Studio Code or Eclipse offer different built-in tools and extensions to support the refactoring processes, which makes them easy to use and integrate into the project development landscape.

Mens & Tourwé (2004) in their refactoring survey note, that the automated software for refactoring purposes increases the refactoring efficiency, and can significantly speed up the refactoring processes, enabling the developers to support the maintainability of the code, and easily integrate all required changes. It makes essential for a modern IT organization, to have automated refactoring tools in their development inventory, which enable the most efficient process for enhancing the code quality.

One more tool, that can assist the organization in ensuring the alignment to clean coding standards is continuous integration tools such as GitHub Actions. When the project code base has multiple contributors and the refactoring is consistently conducted, it's important to ensure, that all changes, which were parallel applied to the source code don't affect the existing functionality. Continuous integration tools serve to automate this process and regularly conduct the build and testing activities, to ensure that all changes made, are aligned with the established clean coding rules.

In, today's fast-growing IT industry, it's also important for an organization to stay up to date with all appearing technical solutions, aimed to support the enhancement the code quality and ensure the alignment to clean code practices. All available solutions, for example, AI-supported refactoring tools, must be taken into account to optimize the refactoring activities as much as possible, and to receive the maximum outcome from consistently applied clean code standards. In conclusion, the alignment to established clean coding rules and consistent conduction of refactoring activities can sufficiently contribute to the project's success, enabling the creation of stable and efficient code, that is easily scalable and maintainable in the future. To ensure the right application of clean coding and refactoring within the project team an organization must consider various aspects, which can affect the application results.

An organization must support continuous educational events such as workshops, seminars, and code reviews to increase the motivation of the development team, keep the level of awareness about established clean coding rules, and create a clean code culture within the development team. Furthermore, it's very important to integrate the refactoring activities into the project plan and dedicate an adequate amount of time to their appropriate conduction.

To simplify and make the clean coding and refactoring activities as efficient as possible, an organization must consider the application of automated tools for code analysis and refactoring and code metrics to decrease the time for performing routine refactoring tasks and enable the right decision-making. Furthermore, the development inventory of an organization must be flexible and stay up to date with all appearing solutions in the IT market.

By adhering to these practices, an organization can reach the maximum level of alignment to clean code standards and practices, automate the refactoring and enhance its efficiency, and obtain the maximum benefits from the application of these practices.

Bibliography

- Fowler, M. (1999). Refactoring: Improving the Design of Existing Code.
- Khomh, F., & Gueheneuc, Y. G. (2009). Exploratory study of the impact of antipatterns on class change- and fault-proneness.
- Vasileva, A., & Schmedding, D. (2016). How to Improve Code Quality by Measurement and Refactoring.
- Mens, T., & Tourwé, T. (2004). A survey of software refactoring.