CSE412: Software Engineering

[Fall 2023]

Project Proposal

Submitted To

Nishat Tasnim Niloy

Lecturer

Department of Computer Science & Engineering

Submitted by:

Student ID	Student Name
2019-3-60-116	Samia Sultana Nishat
2019-1-60-003	Manisha Majumder
2019-3-60-136	Tahsina Hossain Tanha
2019-3-60-055	Sumaiya Rubaiyat
2020-3-60-049	Tahrima Billal Biva
2020-2-60-087	Nabiha Tahsin Tanha

Date of Submission: 4.11.2023

Dead Code Detection and Elimination

1.Introduction

In the ever-evolving landscape of software development, the quest for efficient, reliable, and maintainable code remains a paramount concern. As software systems grow in complexity and scale, they tend to accumulate redundant, obsolete, and unreachable portions of code, commonly referred to as "dead code." Dead code is code that serves no functional purpose in the software but continues to reside within the codebase, posing several challenges. These challenges encompass wasted computational resources, increased software maintenance efforts, and potential sources of bugs and security vulnerabilities.

The identification and subsequent elimination of dead code, a process vital for software optimization, have garnered substantial attention within the software engineering community. This research paper delves into the multifaceted domain of dead code detection and elimination, aiming to shed light on the importance, methodologies, tools, and real-world implications of this crucial software engineering practice. Dead code may manifest in various forms, including functions or methods that are never called, variables that are defined but never used, and entire modules or classes that are never instantiated. Understanding and addressing these issues is instrumental not only for improving the efficiency of a software system but also for enhancing its maintainability. The presence of dead code can obscure the true functionality of a program, making it more challenging to debug, extend, or modify. In large-scale software projects, the consequences of neglecting dead code can be severe, leading to increased development costs and a higher likelihood of introducing defects during maintenance or updates.

The motivation for dead code detection and elimination extends beyond mere code efficiency. In the era of continuous integration and deployment, where software updates are frequent and rapid, the need to maintain a lean and focused codebase is paramount. Furthermore, as software security concerns continue to grow, dead code

can serve as potential entry points for malicious exploitation, making its timely removal a security imperative.

This research paper aims to provide a comprehensive exploration of dead code detection and elimination, covering both theoretical underpinnings and practical applications. It delves into the various techniques, tools, and best practices available for identifying and removing dead code, with a focus on how these practices can enhance software quality, performance, and security.

The subsequent sections of this paper will delve into the methodology and tools used for dead code detection, provide case studies demonstrating the impact of dead code on software projects, and discuss the implications for both developers and the broader software industry. Ultimately, the goal is to underscore the significance of dead code detection and elimination as an indispensable facet of software engineering, contributing to the creation of more efficient, maintainable, and secure software systems.

2.Background

The background of "dead code detection and elimination" refers to the context and motivation behind conducting a study or research on how to optimize a program to save time and memory usage and remove the number of instructions. In that case the background mainly highlights shrinking the program size, which is required in some cases, ignoring the unnecessary codes while executing which reduces the running time. The dead code optimiser simplifies the program structure and removes the unnecessary complexity to the codebase. It also states that by using static and/or inline functions additionally and enabling the use of link-time optimization, the compiler problem of whether dead code is affected the output can be resolved. The background section might elaborate on the objective of the study, such as evaluating the efficiency, effectiveness, and scalability of dead code optimisers.

3.Related Works

1.The paper [1] focuses on origin of dead code, Comprehension and Modification of unfamiliar source code, refactoring opportunities, effect on correctness and completeness and statistical significance. This paper presents a multi-study investigation into the presence and impact of dead code in software development. The study involves professional developers as well as graduate and undergraduate students and aims to answer several key questions regarding dead code.

2.The paper [2] focuses on the methodology for detecting unused classes in software systems. The approach is language-independent and combines static analysis of the source code and binaries with dynamic information obtained during execution. This approach also addresses challenges related to the use of reflection in modern software systems.

3.This paper [3] focuses on developing a method for accurate dead code detection in embedded C code using arithmetic constraint solving. This paper presents the AVACS transfer project T1, which seeks to develop an accurate dead code detection method for embedded C code.

4.This paper [4] focuses on the concept of dead code, its impact on software development, and the challenges associated with its removal.

5.This paper [5] focuses on a new feature in an Integrated Development Environment (IDE) that offers automated identification and removal of dead code from XQuery programs. The tool aims to enhance code quality and maintainability by automatically detecting and eliminating code that is no longer in use, ultimately improving the development experience for XQuery programmers working in evolving schema environments.

4.Problem Statements

In software programs, "dead" or unused code can have detrimental effects such as higher maintenance costs, less readable code, and possible security flaws. Over time, as software develops, dead code—which includes variables, functions, and entire blocks of code—accumulates. It is essential to find and remove this dead code to

keep codebases functional and healthy. The following main issues and concerns connected to dead code discovery and eradication are addressed in this paper:

1. The volume of code in big software projects can be daunting, making it difficult to manually detect dead code. To manage the detection process's scalability, automated technologies are required.
2. The elimination of dead code should not cause new defects or have a detrimental influence on the software's performance. It is vital to ensure that the elimination procedure is safe and does not impair the program's operation.
3. Dead code removal should ideally be integrated into the development workflow, giving developers with insights and tools to make educated code removal decisions.

4.It is difficult to strike a balance between the necessity for code purity and the preservation of historical context.

5.Objectives

The “Dead Code Detection and Elimination” tool aims to achieve the following objectives:

1.The primary objective is to identify unused codes and to locate the functions, classes and variables that are no longer being utilized within the software application.

2.The removal of dead code improves the code quality and enhances the overall quality of codebase by reducing unnecessary clutter, making it more efficient, readable and maintainable.

3.Dead code can sometimes add unnecessary overhead to the execution of a program. By detecting and eliminating dead code, the execution and performance of a program can be improved.

4.By detecting and eliminating dead code, the dead code detector tool aims to decrease the time and effort required for maintaining and debugging any code.

5.Dead code can complicate the codebase, making it harder to understand and work with the code. By detecting and eliminating the dead code this tool aims to simplify the codebase.

6.Detecting and removing dead code simplifies the code review process, making it easier for team members to assess changes and understand the codebase.

7.Dead code can lead to discrepancies between code behaviour and documentation. The dead code detection and elimination tool aims to align these elements to ensure consistency.

8. Unused or hidden code can pose security risks if it contains faults. Detecting and removing dead code reduces the potential attack surface, enhancing software security.

9.By detecting and eliminating dead codes cleaner codebase will be more accessible for developers to work with, making it easier to introduce new features, fix issues, and maintain and extend the software.

6.Planned Methodology

Dead code detection:

You can conduct a manual code review of your system. For each file, class, module and interface, list where it is being used. Each part for which a use location is not found is a candidate for removal. As a by-product, you get a useful list of dependencies.

You can also do the same for each function, variable, constant and so on. This can get tiresome in a large program. Fortunately, there are automated tools to help you out.

Dead code Elimination:

Dead code is code that is considered unreachable by normal program execution. Dead code can be created by adding code under a condition that never evaluates to true. Dead code should be removed since this type of code can produce unexpected results, if accidentally or maliciously forced to execute.Dead code identification is typically performed by algorithms that implement program flows analysis looking for unreachable code. The dead code is eliminated by instructing compilers to remove the code through compiler flags, i.e., '-fdce' is used for Dead Code Elimination

7.Expected results:

The expected results of “Dead Code Detector” tool is to detect the unused and unnecessary functions, classes of a program which are dead codes and then eliminate the dead code from the program to make the code highly accurate.

8.Refereences:

[1] https://www.cs.wm.edu/~denys/pubs/TSE'18-DeadCode.pdf

[2] https://www.cqse.eu/publications/2014-dead-code-detection-on-class-level.pdf

[3] http://www.sc-square.org/CSA/workshop1-papers/paper2.pdf

[4] https://pure.rug.nl/ws/portalfiles/portal/246710839/ICSME_extended_abstract.pdf

[5] https://hal.science/file/index/docid/494233/filename/ICSE10.pdf

[6] https://www.researchgate.net/publication/370276301_JavaScript_Dead_Code_Identification_Elimination_and_Empirical_Assessment

[7]https://arxiv.org/ftp/arxiv/papers/2106/2106.08948.pdf

[8] https://pdxscholar.library.pdx.edu/cgi/viewcontent.cgi?article=2393&context=honorstheses