# Code Clone Detector

MSc Software Engineering
Software Evolution

Authors: Ilia Balandin (15648001) and Vakaris Paulavičius (13430092)
Lecturer Thomas van Binsbergen

December 2024

# Contents

# Chapter 1

# Introduction

In many aspects of life, duplicating someone else's work is unavoidable. Some replicate to learn, others to reduce their workload. The field of software engineering is no exception. Developers frequently reuse code fragments—sometimes deliberately, sometimes out of necessity—resulting in what is commonly referred to as code clones.

While code reuse can speed-up development, it also brings challenges and drawbacks. The motivations behind code cloning and the reasons developers engage in this practice despite its notorious reputation as a "bad code smell" are explored in chapter 2. Chapter 3 continues the discussion by glancing through existing techniques for detecting and managing code clones and explains in detail the approach of the Code Clone Detector (further referred to as CCD) introduced in this report. This chapter provides a detailed explanation of CCD's underlying methodology. In chapter 4 we shift our focus from developing to testing the CCD. In this chapter a Java-based testing benchmark designed to evaluate the tool's accuracy and correctness is introduced. In addition, sections of the chapter discuss how various popular open-source Java projects were tested using the CCD and what results were obtained. In chapter 5 the state of the current implementation of CCD and its scientific relevance are discussed. Finally, chapter 6 concludes the paper, stating that while code cloning is neither inherently good nor bad, tools for its detection and management are essential. The conclusion highlights the need for continued research and innovation in this area.

# Chapter 2

# Reasons behind code clones

There is a general perception that code cloning is undesirable, a view heavily endorsed by various researchers and authors. One prominent voice is M. Fowler, who famously stated: "Number one in the stink parade is duplicated code. If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them" [4]. Other researchers share this perspective, arguing that duplication negatively affects codebase size, increases maintenance complexity, and reduces overall readability [8].

But why do developers clone code in the first place? Some researchers suggest that cloning often results from poor design decisions, short-term thinking, or developers' lack of awareness of existing solutions [7]. In one study, researchers explored the relationship between software quality and duplicate code at the file level [9]. They used the number of file revisions as an indicator of quality, with more revisions suggesting lower quality. Their findings from large-scale legacy systems revealed that modules containing duplicate code exhibited 40% lower quality compared to those without duplication. Moreover, they observed that the more duplicate code a source file contained, the lower its overall quality tended to be.

However, code cloning is not always a harmful practice. [8] argues that some duplicates aid compilers and can be an essential part of specific programming practices. Similarly, Kapser [7] notes that cloning can serve as a strategy for introducing gradual changes to complex, critical systems. It can also be viewed as a method of reusing trusted solutions, simplifying the architecture and enhancing readability by reducing system complexity.

Code cloning presents both opportunities and challenges. It can support developers by enabling code reuse and simplifying specific tasks. On the other hand, it can complicate project management by increasing the complexity of the codebase, leading to higher maintenance costs. Rather than trying to resolve the debate on whether duplication should be completely avoided or selectively embraced, this report introduces CCD, a Java project analysis tool designed to effectively identify and manage code duplication.

# Chapter 3

# Clone detection approaches

Researchers for decades have extensively explored various methods for detecting code duplication, ranging from simple yet universally applicable text-based comparisons [10] to more complex techniques such as Program Dependency Graphs (PDGs) [11]. The Code Clone Detector (CCD) primarily focuses on identifying Type I and Type II code clones, as defined in [5] and [8]. It offers two distinct methods for detecting Type I clones: a text-based approach [10] described in section 3.1 and an Abstract Syntax Tree (AST)-based approach [1] shown in section 3.2. For detecting Type II clones, CCD uses a metrics-based detection model ([3]) described in section 3.3 and AST-based approach. The purpose of this chapter is to explain the algorithms implemented in CCD.

## 3.1 Text-based approach

For the purpose of identifying Type I clones, CCD employs textual clone detection described in section 2.8 of [8]. The rationale for this choice is as follows:

- **Suitability for Type I clone detection:** Textual techniques are particularly appropriate for detecting Type I clones, as these clones are exact textual copies except comments, whitespaces, and empty lines which are removed by the algorithm before comparing.
- **Universality:** Text-based comparison is language-agnostic, allowing for clone detection across programs written in different languages. This is because text-based detection does not rely on knowledge of the syn-

tax or semantics of the programming language and relies purely on its textual representation.

The algorithm for text-based clone detection is based on [10]. Its logic is as follows:

- First, each class is transformed into a standard representation by removing all comments, whitespaces, and empty lines from its textual representation.
- After that, the standard representations of classes are compared. If two classes have identical standard representations, they are classified as Type I clones.

In addition to this basic procedure, CCD text-based algorithm incorporates two key optimizations to enhance the efficiency:

- **Hash function for standard representations:** A hash function computes two metrics for each standard representation: the number of lines and the total length of all lines. When comparing two standard representations, their hashes are checked first. This approach minimizes the need for full comparisons and speeds up the detection process.
- **Ordered standard representation set:** The set of all standard representations was ordered. By sorting these representations in either increasing or decreasing order, the identification of identical standard representations became more efficient.

Code Clone Detector also includes a test suite, ensuring the correctness of the text-based detection algorithm. It can be found in the projects source code.

## 3.2 AST-based approach

The AST-based approach relies on Abstract Syntax Trees (ASTs) to identify code clones. It is one of the most commonly used techniques for clone detection [8]. CCD implements Type I and II clone detection.

### 3.2.1 The general idea

- **Type I clones:** These clones differ from the original code only in comments, whitespace, or empty lines. Since such differences do not affect the structure of the AST, a Type I clone has an abstract syntax tree that is completely identical to its origin.
- **Type II clones:** These clones include changes in variable or function names in addition to the differences allowed for Type-1 clones. Consequently, the ASTs of Type-2 clones are identical to their source except for differences in specific identifiers.

This approach is supported by Baxter et al. [1], who argue that comparing subtrees for structural equality is sufficient for detecting clones. However, this method does not capture "near-miss" clones, which in this context means that Type-3 clones are not identified.

### 3.2.2 The algorithm

Code Clone Detector uses an algorithm that compares classes by examining their ASTs. The algorithm operates in two modes, controlled by an input boolean flag:

- **Exact match mode:** This mode identifies Type-1 clones by requiring completely identical ASTs.
- **Different identifiers mode:** This mode detects both Type-1 and Type-2 clones by allowing variations in variable and function names.

The algorithm consists of three main steps:

- First, each class in the source code is processed, and its AST is traversed using RASCAL's **visit** construct. During traversal, AST nodes are divided into three lists: Declarations, Expressions, and Statements. This classification is based on RASCAL's type system and represents the class in a structured form. The order of elements within each array reflects the structure of the AST, preserving critical information about its organization.
- After that, the representations of all class pairs are compared to identify potential clones. The comparison is conducted in two stages:

  - Sizes of arrays of one class is compared with sizes of arrays of the other class. If the sizes are not identical, the classes are not clones. This preliminary check eliminates most non-clone pairs efficiently, improving the tool's performance.
  - Each pair of arrays (e.g., declarations with declarations, expressions with expressions) is compared element by element. Differences in elements, except for variations in identifiers in the different identifiers mode, indicate that the classes are not clones.

- Lastly, once clone relationships are determined for all class pairs, the results are aggregated into equivalence classes. These equivalence classes form the final output of the algorithm.

The pseudocode of the algorithm can be seen in algorithm 1 figure. Note that functions for comparing declarations, statements and expressions are omitted here.

### 3.2.3 The algorithm problems

The AST comparison process, particularly the content comparison step, relies on conditions to identify non-identical ASTs. Although CCD includes several efficient mechanisms to eliminate most non-identical pairs, some challenges remain. For example, the tool does not distinguish between certain subtle differences, such as "int i;" versus "long i;". This limitation can occasionally lead to false positives.

**Algorithm 1** AST-based code clone detection

```
1: Function: compareAST($t_1$, $t_2$, $bType2$)
2: Extract $(aDecl_1, aExpr_1, aStat_1) \leftarrow t_1$, $(aDecl_2, aExpr_2, aStat_2) \leftarrow t_2$
3: if size($aDecl_1$) $\neq$ size($aDecl_2$) then
4:   return false
5: end if
6: if size($aExpr_1$) $\neq$ size($aExpr_2$) then
7:   return false
8: end if
9: if size($aStat_1$) $\neq$ size($aStat_2$) then
10:   return false
11: end if
12: $i \leftarrow 0$
13: while $i <$ size($aDecl_1$) do
14:   if not compareDecl($aDecl_1[i]$, $aDecl_2[i]$) then
15:     return false
16:   end if
17:   $i \leftarrow i + 1$
18: end while
19: $i \leftarrow 0$
20: while $i <$ size($aExpr_1$) do
21:   if not compareExpr($aExpr_1[i]$, $aExpr_2[i]$, $bType2$) then
22:     return false
23:   end if
24:   $i \leftarrow i + 1$
25: end while
26: $i \leftarrow 0$
27: while $i <$ size($aStat_1$) do
28:   if not compareStat($aStat_1[i]$, $aStat_2[i]$) then
29:     return false
30:   end if
31:   $i \leftarrow i + 1$
32: end while
33: return true
```

```
Function: classifyAST(project, $bType2$)
Initialize $aClass \leftarrow \emptyset$
for each class $c$ in project do
   Add $(\texttt{transformClassToAST}(c), c.src)$ to $aClass$
end for
Initialize $ans \leftarrow \emptyset$
$i \leftarrow 0$
while $i < \texttt{size}(aClass)$ do
   $j \leftarrow i + 1$
   while $j < \texttt{size}(aClass)$ do
      Extract $(t_1, l_1) \leftarrow aClass[i]$, $(t_2, l_2) \leftarrow aClass[j]$
      if compareAST($t_1$, $t_2$, $bType2$) then
         Add $[l_1, l_2]$ to $ans$
      end if
      $j \leftarrow j + 1$
   end while
   $i \leftarrow i + 1$
end while
while true do
   $sz \leftarrow \texttt{size}(ans)$
   $i \leftarrow 0$
   while $i < \texttt{size}(ans)$ do
      $j \leftarrow i + 1$
      while $j < \texttt{size}(ans)$ do
         Extract $l_1 \leftarrow ans[i]$, $l_2 \leftarrow ans[j]$
         $ls \leftarrow \texttt{dup}(l_1 + l_2)$
         if $\texttt{size}(ls) < \texttt{size}(l_1) + \texttt{size}(l_2)$ then
            Remove $ans[j]$ and $ans[i]$ from $ans$
            Add $ls$ to $ans$
         end if
         $j \leftarrow j + 1$
      end while
      $i \leftarrow i + 1$
   end while
end while
return $ans$
```

## 3.3 Metrics-based approach

Another widely used approach for code clone detection is metrics-based detection [3]. A variant of this approach is implemented by CCD for Type II clone detection.

The metrics used in the algorithm are derived from Table 3 in [6]. However, the application of these metrics to Java classes raises several considerations:

- **Language applicability:** [6] does not explicitly state that Java is the target language for its clone-detection approach; instead, it appears to focus on C (see Figure 3). Since the language structure of C is similar to Java, it is highly probable that the metrics are applicable to Java as well.
- **Class vs. Function clones:** [6] focuses on detecting function clones rather than class clones. However, Java classes consist of functions. Therefore, if two classes are clones, their functions are likely clones as well. This enables the metrics from [6] to be adapted for Java class clone detection by summing the metrics of all functions within a class.
- **Metric selection:** [6] presents four comparison points (Tables 1-4) and eight clone classification categories. Only the metrics from Table 3 are used, as [6] states that these points are orthogonal and can be applied independently. Moreover, the metrics from Table 3 are easier to implement in Rascal. Furthermore, the metrics in Table 3 must be identical for Type I and Type-2 clones. For Type-3 clones, minor deviations are allowed. CCD compares these metrics precisely for detecting Type I and Type II clones and allows for a specified delta to detect Type III clones. Although the deltas provided in Table 3 are defined for functions rather than classes, they are applied to classes on the assumption that deviations in function metrics within cloned classes are likely to balance out, resulting in total deltas similar to those in Table 3.

The pseudocode of the metric-based code clone detection can be seen in algorithm 2 figure.

**Algorithm 2** Metrics-based code clone detection

```
Function: compareMetrics($m_1$, $m_2$, $\delta$)
if all metric values match patterns then
    return abs($m_{1i} - m_{2i}$) $\leq \delta_i$ for all $i$
else
    return false
end if

Function: sumMetrics($m_1$, $m_2$)
if all metric values match patterns then
    return metric($m_{1i} + m_{2i}$) for all $i$
else
    return metric($-1, -1, -1.0, -1, -1$)
end if

Function: decisionComplexity(expr)
comp $\leftarrow 0$
for each operator in expr do
    Increment comp by 1
end for
return comp
```

```
Function: calcClass(inp)
if inp is a class declaration then
    Extract relevant components: methods, declarations, expressions
    Calculate unique function calls, declaration, and expression counts
    Compute average complexity from decisions
    return Metric with calculated values
else
    return metric($-1, -1, -1.0, -1, -1$)
end if

Function: classifyType2(project)
Initialize aClass $\leftarrow \emptyset$
for each class $c$ in project do
    met $\leftarrow$ calcClass($c$)
    if met matches existing class then
        Update class list
    else
        Add new class metric entry
    end if
end for
return List of classified classes
```

## Chapter 4

# Analysing Java projects

Having introduced the CCD's approach to code clone detection, this chapter details the methodology for testing CCD and the results of running the analysis on three different Java projects. Section 4.1 discusses the benchmark Java project specifically developed to evaluate the correctness and accuracy of CCD. In section 4.2, the analysis results of two external Java projects are presented. In section 4.3 the effors of analysing external open-source projects are discussed.

## 4.1 Benchmark - SimpleJavaProject

To evaluate the accuracy of CCD, a dedicated Java project named *SimpleJavaProject* was created. This project, located in the **/benchmark** directory, contains examples of Type I, II, and III clones. The *SimpleJavaProject* serves as a comprehensive benchmark for verifying CCD's ability to detect various types of clones effectively. Specifically, the project includes an ensamble of Java classes and methods.

The benchmark project was analysed with CCD, successfully detecting all instances of Type I, II, and III clones without any false negatives. This demonstrates that the tool achieves 100% accuracy within the benchmark context. Appendix A contains the examples of the Java clones from *SimpleJavaProject* as well as the screenshots of the analysis results. The results of *SimpleJavaProject* are also provided in table 4.1.

| | **AST** | **AST (Type 2)** | **Textual** | **Metrics** |
|---|---|---|---|---|
| **Clone Type** | 1 | 2 | 1 | 2 |
| **Clone Classes** | 1 | 1 | 1 | 1 |
| **Biggest Clone Class** | 4 | 6 | 4 | 7 |
| **Clones** | 4 | 6 | 4 | 7 |
| **Biggest Clone (Lines)** | 35 | 35 | 35 | 35 |
| **Duplicated Lines (%)** | 32.26 | 48.39 | 32.26 | 55.07 |

Table 4.1: Clone Analysis Summary for Simple Java Project

## 4.2 Analysing external projects

In addition to the benchmark, CCD analysis was applied to other Java projects. These included two SQL-based database management applications, *SmallSQL* and *HSQLDB*, whose source code was provided by academic staff.

The analysis aimed to investigate code duplication within these industry-relevant Java projects. This selection of different practical applications provides a robust evaluation of CCD's effectiveness across different types of software systems. The subsections below show the results obtained by running the analysis using CCD. Additionally, the figures in Appendix A include images of terminal outputs from the project analysis, along with examples of detected clones presented in textual format.

### 4.2.1 SmallSQL project analysis results

| | **AST** | **AST (Type 2)** | **Textual** | **Metrics** |
|---|---|---|---|---|
| **Clone Type** | 1 | 2 | 1 | 2 |
| **Clone Classes** | 1 | 8 | 1 | 8 |
| **Biggest Clone Class** | 2 | 12 | 2 | 14 |
| **Clones** | 2 | 28 | 2 | 32 |
| **Biggest Clone (Lines)** | 5 | 121 | 5 | 121 |
| **Duplicated Lines (%)** | 0.04 | 1.98 | 0.04 | 2.10 |

Table 4.2: Clone Analysis Summary for SmallSQL Project

### 4.2.2 HSQLDB project analysis results

| | AST | AST (Type 2) | Textual | Metrics |
|---|---|---|---|---|
| **Clone Type** | 1 | 2 | 1 | 2 |
| **Clone Classes** | 5 | 10 | 3 | 10 |
| **Biggest Clone Class** | 6 | 9 | 6 | 6 |
| **Clones** | 14 | 34 | 10 | 30 |
| **Biggest Clone (Lines)** | 34 | 34 | 34 | 34 |
| **Duplicated Lines (%)** | 0.14 | 0.23 | 0.08 | 0.19 |

Table 4.3: Clone Analysis Summary for HSQLDB Project

## 4.3 Considering other open-source projects

In addition to the two projects provided by the academic staff, other open-source Java projects were also considered for analysis. These projects were identified through a GitHub search using the query "language:Java", which returned a pageable list of Java repositories. The results were sorted by popularity in descending order, and one notable industry-relevant project, DBeaver [2], was selected. However, the analysis tool failed to process this project successfully, indicating that it could not interpret the project's AST structure. This highlights that not all projects can be analyzed without prior preparation, emphasizing the need for further work to enable the automatic analysis of such projects using CCD.

# Chapter 5

# Discussion

This project analysed SmallSQL and HSQLDB for code clones and clone statistics, producing surprisingly positive results. In HSQLDB, the percentage of cloned lines is less than 1%, as is the proportion of type-1 clones in SmallSQL. For SmallSQL, approximately 2% of duplicated lines fall under type-2 clones. These figures are significantly lower than the typical clone rates reported in software projects. However, these results are considered valid for the following reasons:

- **Use of multiple clone-detection techniques:** Independent methods consistently produced similar results:
  - For type-1 clones in SmallSQL, both AST and textual approaches reported 0.04% duplicated lines.
  - For type-2 clones in SmallSQL, metrics-based and AST approaches both detected approximately 2% duplicated lines.
  - Similar patterns were observed in the results for HSQLDB.
- **Manual validation:** Clones identified by the tools were manually reviewed and deemed accurate and reasonable.
- **Project characteristics:** SmallSQL and HSQLDB are popular, well-maintained projects, which likely explains their minimal code duplication.

The analysis of these Java projects demonstrates the effectiveness of CCD in identifying code clones within industry-relevant systems, even though some projects may require additional preparation before analysis. The consistent detection results across projects like SmallSQL and HSQLDB underscore the existence of code duplication in software projects, regardless of whether it arises from short-term developer decisions, oversight, or the intentional reuse of established constructs.

# Chapter 6

# Conclusion

In conclusion, the results of this study highlight the significant role of code clone detection in maintaining and improving software quality. Through the implementation and evaluation of the Code Clone Detector, it has been demonstrated that code cloning is a prevalent yet manageable phenomenon within software development. The analysis of both benchmark and external Java projects showcased CCD's ability to detect and categorize clones effectively, achieving results within expected ranges and confirming its reliability and applicability to real-world systems. However, it was also shown that CCD can sometimes fail to read existing Java projects thus fails to analyse them, calling for further algorithmic improvements.

While code cloning can provide short-term development benefits, it also introduces risks to code maintainability and scalability, underscoring the need for robust tools like CCD. These findings emphasize the need for continued research and the development of more advanced code clone detection tools, especially as software systems become increasingly complex. Future work could focus on extending CCD's capabilities to detect more complex clone types (Type III and IV) and exploring integrations with automated refactoring tools to not only identify but also address code duplication issues proactively. Ultimately, this research underscores that managing code duplication is a universal challenge in software engineering, necessitating further advancements in tools and methodologies.

# Bibliography

[1] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. "Clone detection using abstract syntax trees". In: *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272).* IEEE. 1998, pp. 368–377.

[2] D. Contributors. *DBeaver*. `https://github.com/dbeaver/dbeaver`. Accessed: 2024-12-14. 2024.

[3] M. S. Dhavleesh Rattan Rajesh Bhatia. "Software clone detection: A systematic review". In: *Information and Software Technology* 55 (2013), pp. 1165–1199.

[4] M. Fowler. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 2018.

[5] P. Gautam and H. Saini. "Various code clone detection techniques and tools: a comprehensive survey". In: *Smart Trends in Information Technology and Computer Communications: First International Conference, SmartCom 2016, Jaipur, India, August 6–7, 2016, Revised Selected Papers 1.* Springer. 2016, pp. 655–667.

[6] E. M. M. Jean Mayrand Claude Leblanc. "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics". In: *IEEE* (1996), pp. 244–253.

[7] C. Kapser and M. W. Godfrey. "'Cloning Considered Harmful' Considered Harmful". In: *2006 13th Working Conference on Reverse Engineering.* Oct. 2006, pp. 19–28.

[8] R. Koschke. "Software Evolution". In: ed. by T. Mens and S. Demeyer. Springer, 2008. Chap. 2. Identifying and Removing Software Clones, pp. 15–36.

[9] A. Monden, D. Nakae, T. Kamiya, S.-i. Sato, and K.-i. Matsumoto. "Software quality analysis by code clones in industrial legacy software". In: *Proceedings Eighth IEEE Symposium on Software Metrics*. IEEE. 2002, pp. 87–94.

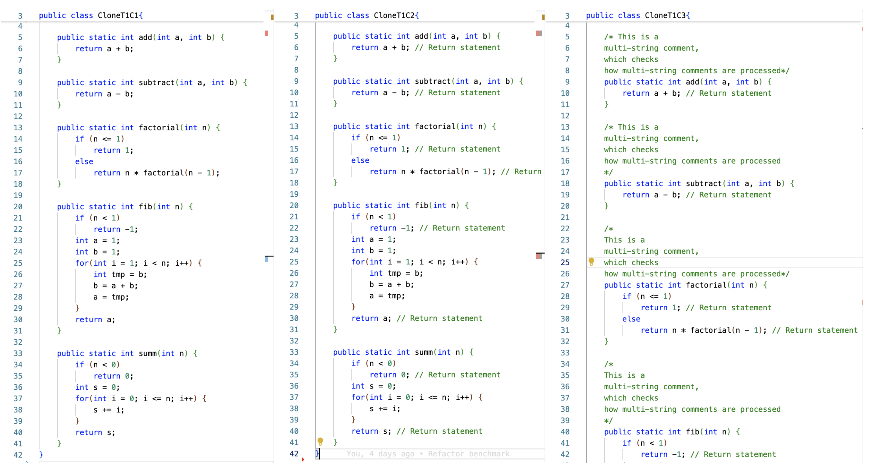
[10] S. D. Stephane Ducasse Matthias Rieger. "A Language Independent Approach for Detecting Duplicated Code". In: *ICSM99* (2000).

[11] Y. Zou, B. Ban, Y. Xue, and Y. Xu. "CCGraph: a PDG-based code clone detector with approximate graph matching". In: *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*. 2020, pp. 931–942.

# Appendices

# Appendix A

# Additional supporting images

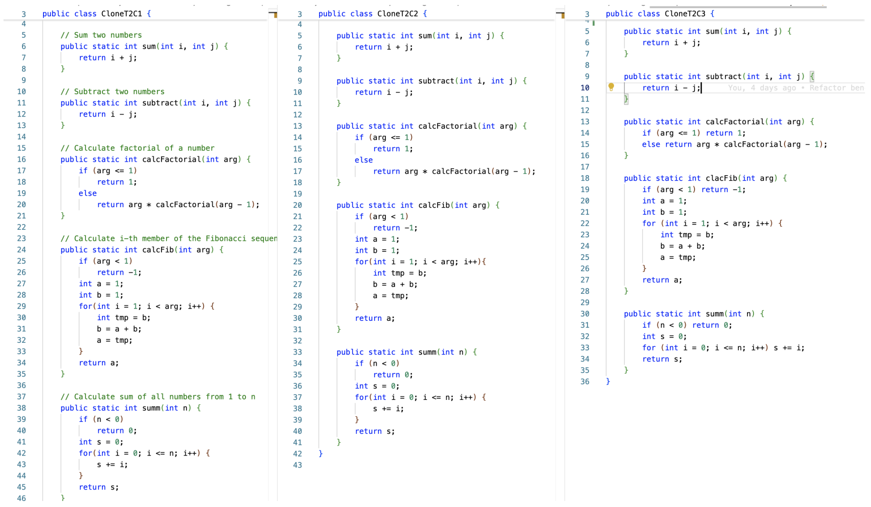


```java
public class CloneT1C1{

    public static int add(int a, int b) {
        return a + b;
    }

    public static int subtract(int a, int b) {
        return a - b;
    }

    public static int factorial(int n) {
        if (n <= 1)
            return 1;
        else
            return n * factorial(n - 1);
    }

    public static int fib(int n) {
        if (n < 1)
            return -1;
        int a = 1;
        int b = 1;
        for(int i = 1; i < n; i++) {
            int tmp = b;
            b = a + b;
            a = tmp;
        }
        return a;
    }

    public static int summ(int n) {
        if (n < 0)
            return 0;
        int s = 0;
        for(int i = 0; i <= n; i++) {
            s += i;
        }
        return s;
    }
}
```

```java
public class CloneT1C2{

    public static int add(int a, int b) {
        return a + b; // Return statement
    }

    public static int subtract(int a, int b) {
        return a - b; // Return statement
    }

    public static int factorial(int n) {
        if (n <= 1)
            return 1; // Return statement
        else
            return n * factorial(n - 1); // Return
    }

    public static int fib(int n) {
        if (n < 1)
            return -1; // Return statement
        int a = 1;
        int b = 1;
        for(int i = 1; i < n; i++) {
            int tmp = b;
            b = a + b;
            a = tmp;
        }
        return a; // Return statement
    }

    public static int summ(int n) {
        if (n < 0)
            return 0; // Return statement
        int s = 0;
        for(int i = 0; i <= n; i++) {
            s += i;
        }
        return s; // Return statement
    }
}       You, 4 days ago • Refactor benchmark
```

```java
public class CloneT1C3{

    /* This is a
    multi-string comment,
    which checks
    how multi-string comments are processed*/
    public static int add(int a, int b) {
        return a + b; // Return statement
    }

    /* This is a
    multi-string comment,
    which checks
    how multi-string comments are processed
    */
    public static int subtract(int a, int b) {
        return a - b; // Return statement
    }

    /*
    This is a
    multi-string comment,
    which checks
    how multi-string comments are processed*/
    public static int factorial(int n) {
        if (n <= 1)
            return 1; // Return statement
        else
            return n * factorial(n - 1); // Return statement
    }

    /*
    This is a
    multi-string comment,
    which checks
    how multi-string comments are processed
    */
    public static int fib(int n) {
        if (n < 1)
            return -1; // Return statement
```

Figure A.1: SimpleJavaProject - Type I clone classes

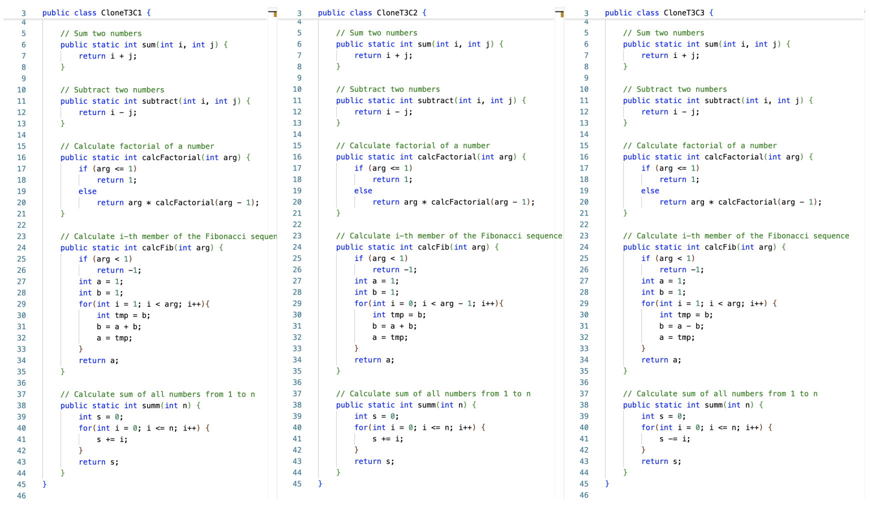


```java
public class CloneT2C1 {

    // Sum two numbers
    public static int sum(int i, int j) {
        return i + j;
    }

    // Subtract two numbers
    public static int subtract(int i, int j) {
        return i - j;
    }

    // Calculate factorial of a number
    public static int calcFactorial(int arg) {
        if (arg <= 1)
            return 1;
        else
            return arg * calcFactorial(arg - 1);
    }

    // Calculate i-th member of the Fibonacci sequen
    public static int calcFib(int arg) {
        if (arg < 1)
            return -1;
        int a = 1;
        int b = 1;
        for(int i = 1; i < arg; i++) {
            int tmp = b;
            b = a + b;
            a = tmp;
        }
        return a;
    }

    // Calculate sum of all numbers from 1 to n
    public static int summ(int n) {
        if (n < 0)
            return 0;
        int s = 0;
        for(int i = 0; i <= n; i++) {
            s += i;
        }
        return s;
    }
```

```java
public class CloneT2C2 {

    public static int sum(int i, int j) {
        return i + j;
    }

    public static int subtract(int i, int j) {
        return i - j;
    }

    public static int calcFactorial(int arg) {
        if (arg <= 1)
            return 1;
        else
            return arg * calcFactorial(arg - 1);
    }

    public static int calcFib(int arg) {
        if (arg < 1)
            return -1;
        int a = 1;
        int b = 1;
        for(int i = 1; i < arg; i++){
            int tmp = b;
            b = a + b;
            a = tmp;
        }
        return a;
    }

    public static int summ(int n) {
        if (n < 0)
            return 0;
        int s = 0;
        for(int i = 0; i <= n; i++) {
            s += i;
        }
        return s;
    }
}
```

```java
public class CloneT2C3 {
    public static int sum(int i, int j) {
        return i + j;
    }

    public static int subtract(int i, int j) {
        return i - j;        You, 4 days ago • Refactor ben
    }

    public static int calcFactorial(int arg) {
        if (arg <= 1) return 1;
        else return arg * calcFactorial(arg - 1);
    }

    public static int clacFib(int arg) {
        if (arg < 1) return -1;
        int a = 1;
        int b = 1;
        for (int i = 1; i < arg; i++) {
            int tmp = b;
            b = a + b;
            a = tmp;
        }
        return a;
    }

    public static int summ(int n) {
        if (n < 0) return 0;
        int s = 0;
        for (int i = 0; i <= n; i++) s += i;
        return s;
    }
}
```

Figure A.2: SimpleJavaProject - Type II clone classes

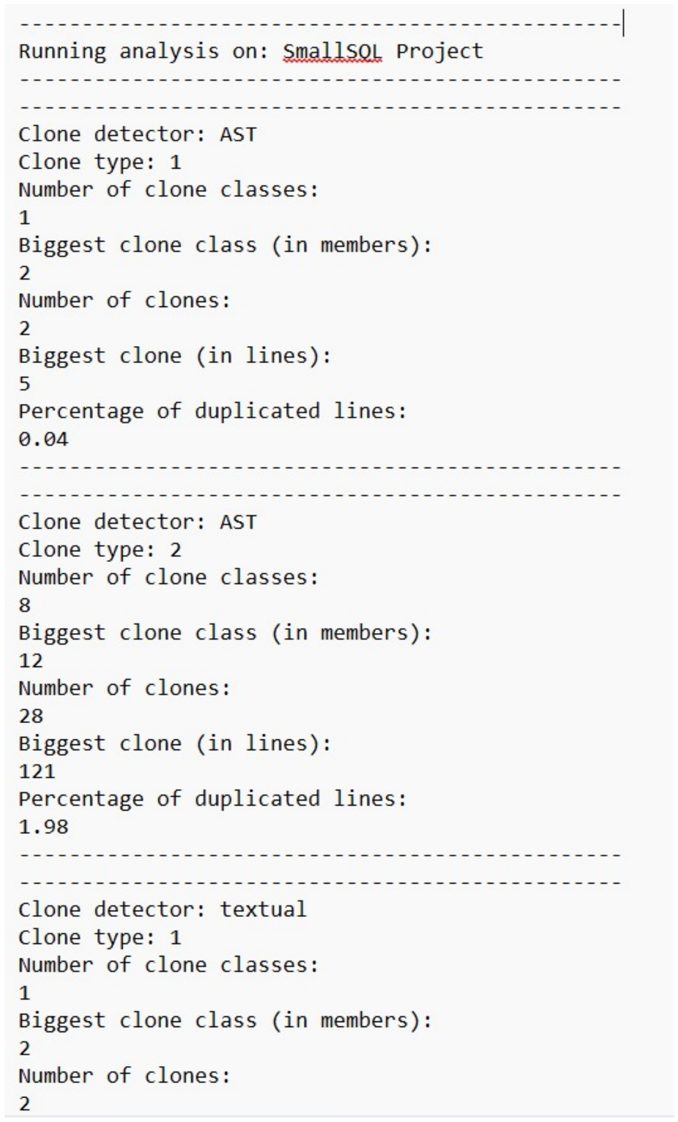


```java
public class CloneT3C1 {

    // Sum two numbers
    public static int sum(int i, int j) {
        return i + j;
    }

    // Subtract two numbers
    public static int subtract(int i, int j) {
        return i - j;
    }

    // Calculate factorial of a number
    public static int calcFactorial(int arg) {
        if (arg <= 1)
            return 1;
        else
            return arg * calcFactorial(arg - 1);
    }

    // Calculate i-th member of the Fibonacci sequen
    public static int calcFib(int arg) {
        if (arg < 1)
            return -1;
        int a = 1;
        int b = 1;
        for(int i = 1; i < arg; i++){
            int tmp = b;
            b = a + b;
            a = tmp;
        }
        return a;
    }

    // Calculate sum of all numbers from 1 to n
    public static int summ(int n) {
        int s = 0;
        for(int i = 0; i <= n; i++) {
            s += i;
        }
        return s;
    }
}
```

```java
public class CloneT3C2 {

    // Sum two numbers
    public static int sum(int i, int j) {
        return i + j;
    }

    // Subtract two numbers
    public static int subtract(int i, int j) {
        return i - j;
    }

    // Calculate factorial of a number
    public static int calcFactorial(int arg) {
        if (arg <= 1)
            return 1;
        else
            return arg * calcFactorial(arg - 1);
    }

    // Calculate i-th member of the Fibonacci sequence
    public static int calcFib(int arg) {
        if (arg < 1)
            return -1;
        int a = 1;
        int b = 1;
        for(int i = 0; i < arg - 1; i++){
            int tmp = b;
            b = a + b;
            a = tmp;
        }
        return a;
    }

    // Calculate sum of all numbers from 1 to n
    public static int summ(int n) {
        int s = 0;
        for(int i = 0; i <= n; i++) {
            s += i;
        }
        return s;
    }
}
```

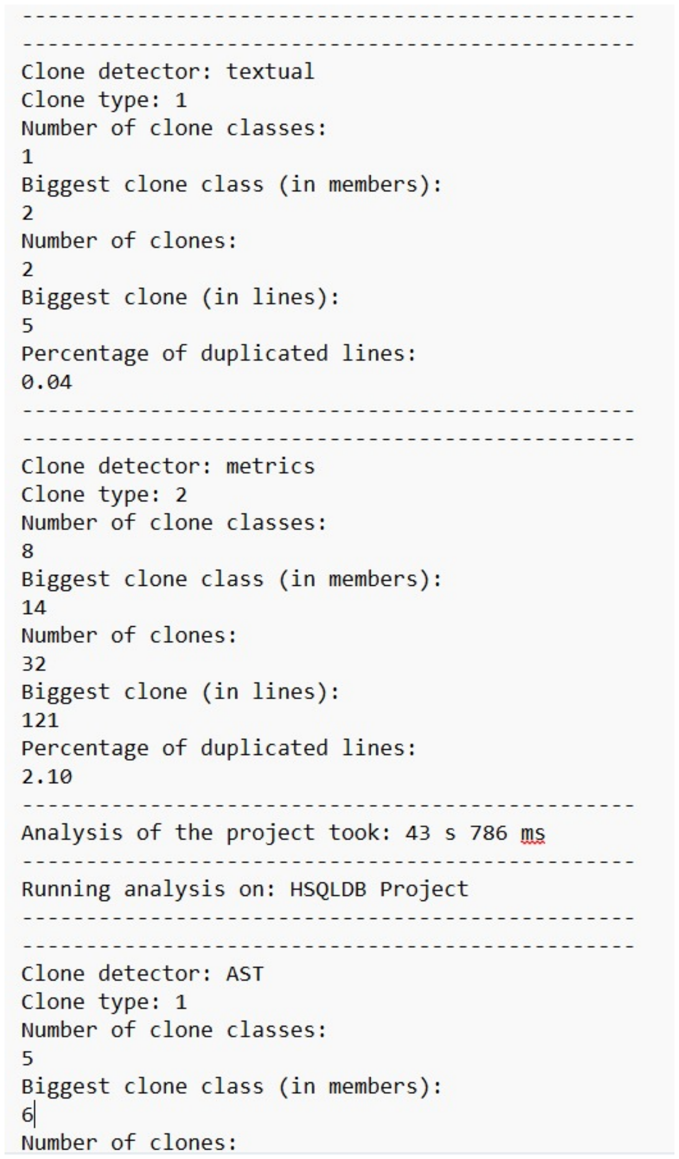
```java
public class CloneT3C3 {

    // Sum two numbers
    public static int sum(int i, int j) {
        return i + j;
    }

    // Subtract two numbers
    public static int subtract(int i, int j) {
        return i - j;
    }

    // Calculate factorial of a number
    public static int calcFactorial(int arg) {
        if (arg <= 1)
            return 1;
        else
            return arg * calcFactorial(arg - 1);
    }

    // Calculate i-th member of the Fibonacci sequence
    public static int calcFib(int arg) {
        if (arg < 1)
            return -1;
        int a = 1;
        int b = 1;
        for(int i = 1; i < arg; i++) {
            int tmp = b;
            b = a - b;
            a = tmp;
        }
        return a;
    }

    // Calculate sum of all numbers from 1 to n
    public static int summ(int n) {
        int s = 0;
        for(int i = 0; i <= n; i++) {
            s -= i;
        }
        return s;
    }
}
```

Figure A.3: SimpleJavaProject - Type III clone classes



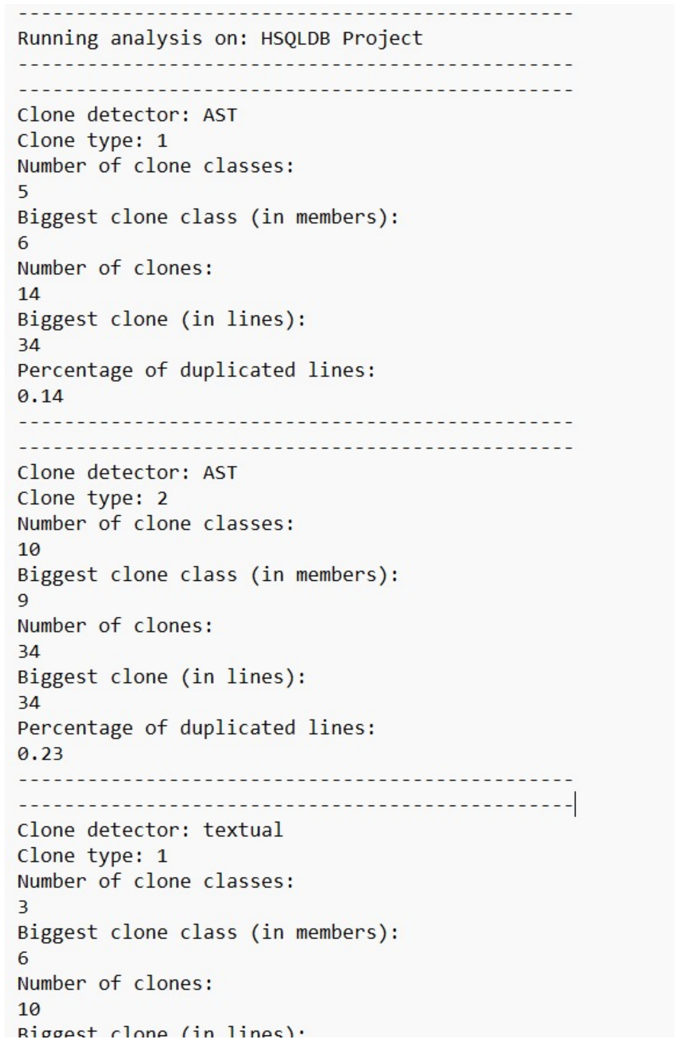
```
------------------------------------------------
Running analysis on: SmallSQL Project
------------------------------------------------
------------------------------------------------
Clone detector: AST
Clone type: 1
Number of clone classes:
1
Biggest clone class (in members):
2
Number of clones:
2
Biggest clone (in lines):
5
Percentage of duplicated lines:
0.04
------------------------------------------------
------------------------------------------------
Clone detector: AST
Clone type: 2
Number of clone classes:
8
Biggest clone class (in members):
12
Number of clones:
28
Biggest clone (in lines):
121
Percentage of duplicated lines:
1.98
------------------------------------------------
------------------------------------------------
Clone detector: textual
Clone type: 1
Number of clone classes:
1
Biggest clone class (in members):
2
Number of clones:
2
```

Figure A.4: Analysis results 1



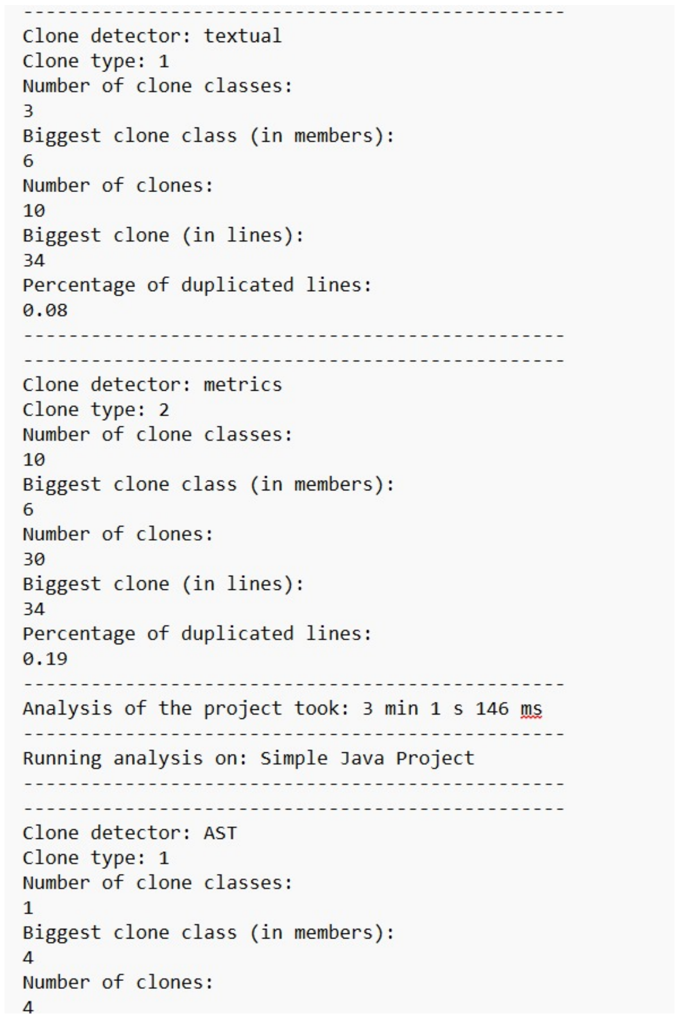
```
------------------------------------------------
------------------------------------------------
Clone detector: textual
Clone type: 1
Number of clone classes:
1
Biggest clone class (in members):
2
Number of clones:
2
Biggest clone (in lines):
5
Percentage of duplicated lines:
0.04
------------------------------------------------
------------------------------------------------
Clone detector: metrics
Clone type: 2
Number of clone classes:
8
Biggest clone class (in members):
14
Number of clones:
32
Biggest clone (in lines):
121
Percentage of duplicated lines:
2.10
------------------------------------------------
Analysis of the project took: 43 s 786 ms
------------------------------------------------
Running analysis on: HSQLDB Project
------------------------------------------------
------------------------------------------------
Clone detector: AST
Clone type: 1
Number of clone classes:
5
Biggest clone class (in members):
6
Number of clones:
```

Figure A.5: Analysis results 2



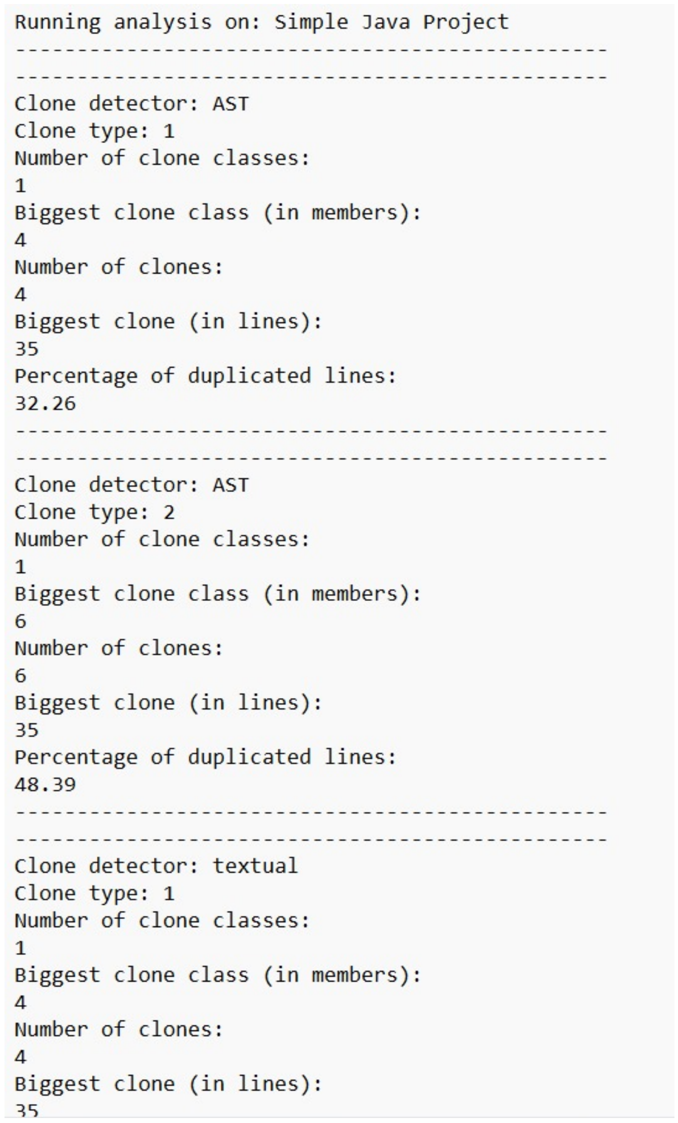
```
-----------------------------------------------
Running analysis on: HSQLDB Project
-----------------------------------------------
-----------------------------------------------
Clone detector: AST
Clone type: 1
Number of clone classes:
5
Biggest clone class (in members):
6
Number of clones:
14
Biggest clone (in lines):
34
Percentage of duplicated lines:
0.14
-----------------------------------------------
-----------------------------------------------
Clone detector: AST
Clone type: 2
Number of clone classes:
10
Biggest clone class (in members):
9
Number of clones:
34
Biggest clone (in lines):
34
Percentage of duplicated lines:
0.23
-----------------------------------------------
-----------------------------------------------
Clone detector: textual
Clone type: 1
Number of clone classes:
3
Biggest clone class (in members):
6
Number of clones:
10
Biggest clone (in lines):
```

Figure A.6: Analysis results 3



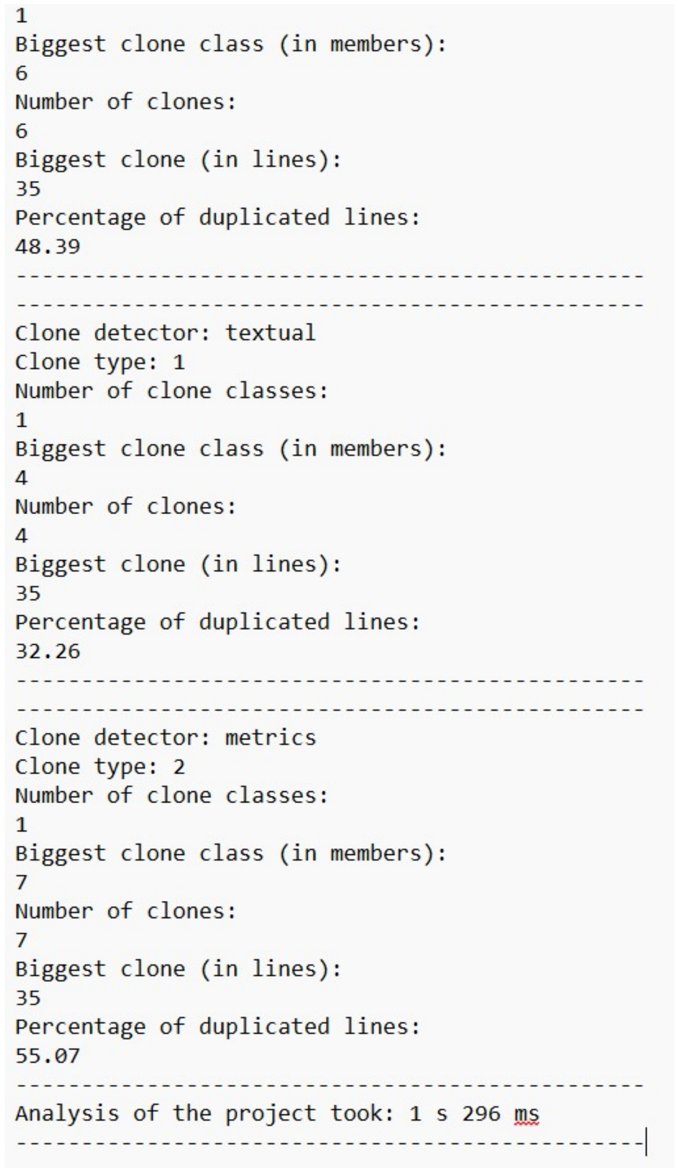
```
-----------------------------------------------
Clone detector: textual
Clone type: 1
Number of clone classes:
3
Biggest clone class (in members):
6
Number of clones:
10
Biggest clone (in lines):
34
Percentage of duplicated lines:
0.08
-----------------------------------------------
-----------------------------------------------
Clone detector: metrics
Clone type: 2
Number of clone classes:
10
Biggest clone class (in members):
6
Number of clones:
30
Biggest clone (in lines):
34
Percentage of duplicated lines:
0.19
-----------------------------------------------
Analysis of the project took: 3 min 1 s 146 ms
-----------------------------------------------
Running analysis on: Simple Java Project
-----------------------------------------------
-----------------------------------------------
Clone detector: AST
Clone type: 1
Number of clone classes:
1
Biggest clone class (in members):
4
Number of clones:
4
```

Figure A.7: Analysis results 4



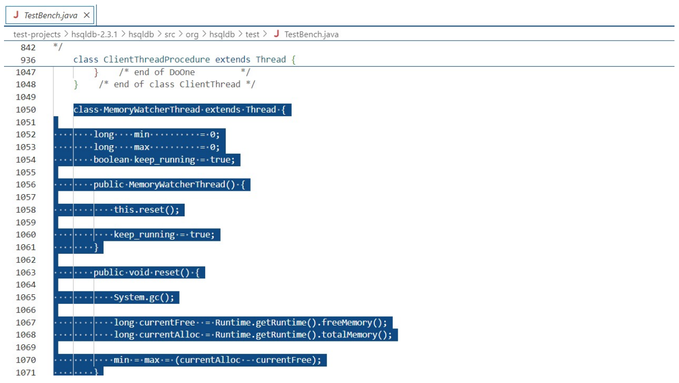
```
Running analysis on: Simple Java Project
------------------------------------------------
------------------------------------------------
Clone detector: AST
Clone type: 1
Number of clone classes:
1
Biggest clone class (in members):
4
Number of clones:
4
Biggest clone (in lines):
35
Percentage of duplicated lines:
32.26
------------------------------------------------
------------------------------------------------
Clone detector: AST
Clone type: 2
Number of clone classes:
1
Biggest clone class (in members):
6
Number of clones:
6
Biggest clone (in lines):
35
Percentage of duplicated lines:
48.39
------------------------------------------------
------------------------------------------------
Clone detector: textual
Clone type: 1
Number of clone classes:
1
Biggest clone class (in members):
4
Number of clones:
4
Biggest clone (in lines):
35
```

Figure A.8: Analysis results 5



```
1
Biggest clone class (in members):
6
Number of clones:
6
Biggest clone (in lines):
35
Percentage of duplicated lines:
48.39
-----------------------------------------------
-----------------------------------------------
Clone detector: textual
Clone type: 1
Number of clone classes:
1
Biggest clone class (in members):
4
Number of clones:
4
Biggest clone (in lines):
35
Percentage of duplicated lines:
32.26
-----------------------------------------------
-----------------------------------------------
Clone detector: metrics
Clone type: 2
Number of clone classes:
1
Biggest clone class (in members):
7
Number of clones:
7
Biggest clone (in lines):
35
Percentage of duplicated lines:
55.07
-----------------------------------------------
Analysis of the project took: 1 s 296 ms
-----------------------------------------------
```

Figure A.9: Analysis results 6

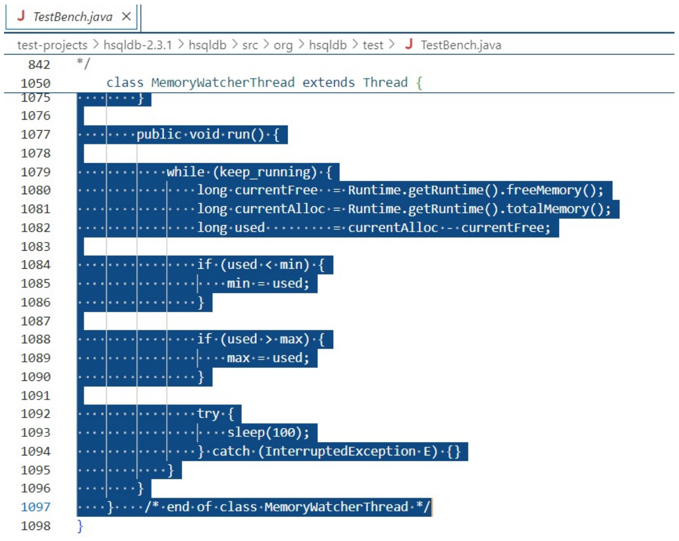


TestBench.java

test-projects > hsqldb-2.3.1 > hsqldb > src > org > hsqldb > test > TestBench.java

```java
*/
    class ClientThreadProcedure extends Thread {
            }    /* end of DoOne         */
    }    /* end of class ClientThread */

    class MemoryWatcherThread extends Thread {

        long    min           = 0;
        long    max           = 0;
        boolean keep_running  = true;

        public MemoryWatcherThread() {

            this.reset();

            keep_running = true;
        }

        public void reset() {

            System.gc();

            long currentFree  = Runtime.getRuntime().freeMemory();
            long currentAlloc = Runtime.getRuntime().totalMemory();

            min = max = (currentAlloc - currentFree);
        }
```

Figure A.10: Type I clone in textual form

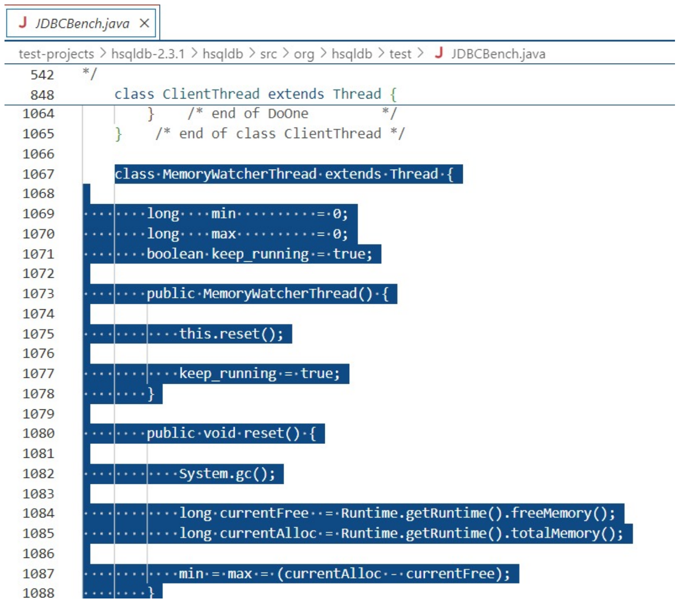


TestBench.java

test-projects > hsqldb-2.3.1 > hsqldb > src > org > hsqldb > test > TestBench.java

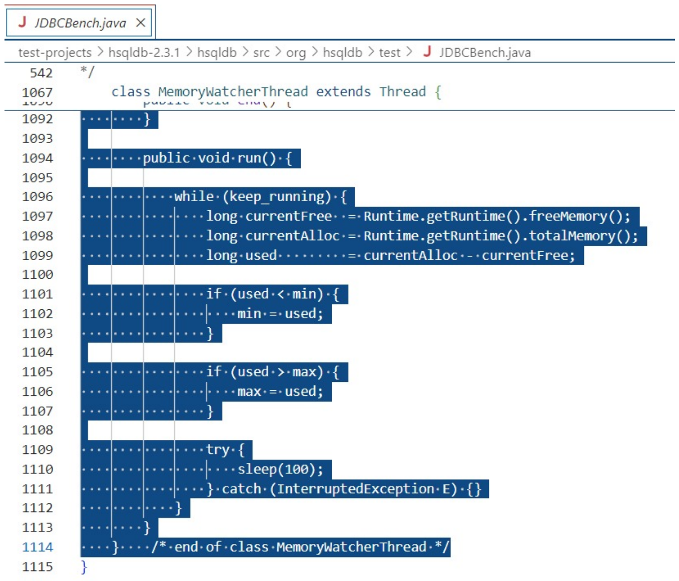
```java
*/
    class MemoryWatcherThread extends Thread {
        }

        public void run() {

            while (keep_running) {
                long currentFree  = Runtime.getRuntime().freeMemory();
                long currentAlloc = Runtime.getRuntime().totalMemory();
                long used         = currentAlloc - currentFree;

                if (used < min) {
                    min = used;
                }

                if (used > max) {
                    max = used;
                }

                try {
                    sleep(100);
                } catch (InterruptedException E) {}
            }
        }
    }    /* end of class MemoryWatcherThread */
}
```

Figure A.11: Type I clone in textual form



```
JDBCBench.java
test-projects > hsqldb-2.3.1 > hsqldb > src > org > hsqldb > test > JDBCBench.java
*/
    class ClientThread extends Thread {
        }    /* end of DoOne         */
    }    /* end of class ClientThread */

    class MemoryWatcherThread extends Thread {

        long    min          = 0;
        long    max          = 0;
        boolean keep_running = true;

        public MemoryWatcherThread() {

            this.reset();

            keep_running = true;
        }

        public void reset() {

            System.gc();

            long currentFree  = Runtime.getRuntime().freeMemory();
            long currentAlloc = Runtime.getRuntime().totalMemory();

            min = max = (currentAlloc - currentFree);
        }
```

Figure A.12: Type I clone in textual form



```
JDBCBench.java
test-projects > hsqldb-2.3.1 > hsqldb > src > org > hsqldb > test > JDBCBench.java
*/
    class MemoryWatcherThread extends Thread {
        }

        public void run() {

            while (keep_running) {
                long currentFree  = Runtime.getRuntime().freeMemory();
                long currentAlloc = Runtime.getRuntime().totalMemory();
                long used         = currentAlloc - currentFree;

                if (used < min) {
                    min = used;
                }

                if (used > max) {
                    max = used;
                }

                try {
                    sleep(100);
                } catch (InterruptedException E) {}
            }
        }
    }    /* end of class MemoryWatcherThread */
}
```

Figure A.13: Type I clone in textual form