# Hey! Are You Injecting Side Effect?: A Tool for Detecting Purity Changes in Java Methods

Naoto Ogura*, Jiachen Yang*, Keisuke Hotta*, Yoshiki Higo* and Shinji Kusumoto*
*Graduate School of Information Science and Technology, Osaka University, Japan
Email: {n-ogura, jc-yang, k-hotta, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—Methods not having side effects (pure methods) are beneficial in some situations. For example, data race does not occur among pure methods in multi-thread programs. Another example is that there are some cases where developers expect methods are pure, such as *equals*, *hashCode*, and getter methods in Java. This paper presents a tool finding code changes where methods become pure/impure. This tool can prevent developers from inducing purity-related bugs to methods. The authors have applied the tool to two open source systems and found (1) a dozen of methods moved to pure/impure repeatedly and (2) there were many cases where purity of methods had changed without code changes.**

## I. Introduction

In functional programming languages, purity of function is an important feature. In this paper, methods not having side effects in object-oriented programming languages are called pure methods[1]. Figure 1 shows examples of pure/impure methods. Pure methods are beneficial in some situations. Data race, which is a major problem in multi-thread programs, does not occur among pure methods. There are also some cases where developers expect methods are pure, such as *equals*, *hashCode*, and getter methods in Java.

The authors assumed, if purity of methods is changed unintentionally, it implies bug inducing. In other words, finding purity changes are useful to find induced bugs and prevent new bugs from being induced. This paper presents a tool finding purity changes in revisions of source code repositories. The authors have applied the tool to two open source systems and there were many methods whose purity was changed.

The tool is open to the public in the authors' website[1].

```java
int add(int a, int b) {
  return a + b;
}
```

(a) Pure Method

```java
private int counter;
int count() {
  return counter++;
}
```

(b) Impure Method

Fig. 1: Examples of Pure/Impure Methods

## II. Preliminaries

### A. Side Effect

In the functional programming languages, a function has side effects when there are states inside or outside of the function definition that has been changed before and after the execution of the function. On the contrary, a function without side effects is called a pure function. For example, if a function modifies the value of a global variable or performing I/O operations, the function has side effects because it changes the state outside the function. If the function modifies a value of a static variable defined inside that function, the function also has side effects because it changes the state inside the function. On the other hand, if the function only modifies the value passed to the function as arguments, or only reads the value from global variables, the function does not have side effects because it does not change the states.

### B. Pure Method

We applied the idea of purity from functional programming languages to the object oriented programming languages and call the methods without side effects as pure methods [1]. The return value of a pure method depends on only the passed arguments, the member fields, and the static fields. A pure method will not change the state inside or outside the method. Because pure methods will not affect the execution of other methods, they are useful in implementing multi-thread programs.

The return values of pure methods depend on only argument variables and field variables. In other words, as long as used field variables are not changed, the return value of pure method will be the same results when the same parameter is passed. Yang et al. reported that caching the return values of pure methods was useful for improvement of the performance at three Java software projects[2].

## III. Tool Overview

### A. Architectural Design

This tool is a command-line based static analysis tool implemented in Java. This tool employs purano [1] and ECTEC [3] developed by our research group. This tool takes a software repository as input and detects purity changes of methods between every pair of consecutive revisions. If the purity of a method is changed between a pair of revisions, the information of the method and the commit are stored into database as output. The targets of this tool are source code of compilable Java software.

[1] http://sdl.ist.osaka-u.ac.jp/~n-ogura/2016_detecting-purity-changes

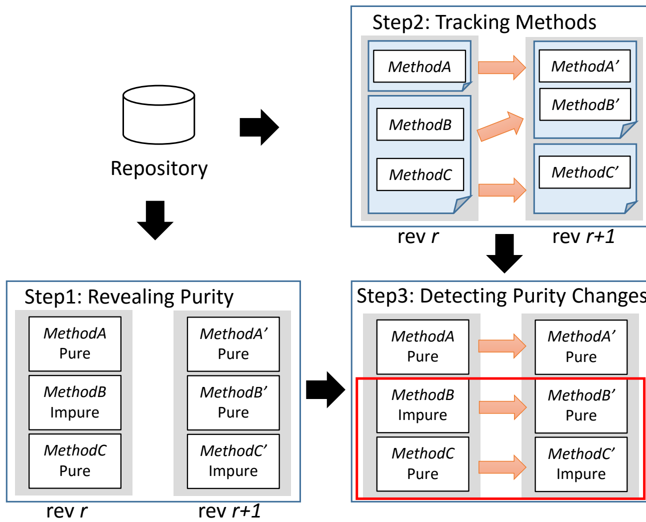


Fig. 2: Overview of Detection Algorithm

### B. Usage in 2 Scenarios

*1) Checking before Committing:* in this scenario, developers use this tool in code implementation. Developers can know purity changes are included in their code changes by using this tool before committing. When purity is changed by the latest developer's change, this tool notifies the developer. This notification helps developers to know unintentional side effect injections before committing. When developers get the notification, they consider whether the side effect injection will be appropriate. In other words, the notificatoin will prevent bug inducing.

*2) Checking history:* in this scenario, developers apply this tool to repositories to check development history. For example, they can know how many times purity of each method has been changed. Past purity changes might be related to bug inducing. Investigating past purity changes can be a way to detect latent bugs, which should improve the quality of the software.

For another example, there is a case where bugs with data race occur in multi-thread programs. The cause of the bugs might be inserting operations assigning the same variable into multiple methods. Purity changes sometimes occur by such insertions. This tool is useful for developers to enumerate past changes to investigate.

TABLE I: Overview of Target Software

| Software | Start Date | End Date | #Revision | LOC |
|---|---|---|---|---|
| jEdit | 2/Sep/2001 | 17/Aug/2012 | 5,302 | 183,093 |
| JFreeChart | 19/Jun/2007 | 28/Jan/2013 | 1,606 | 323,497 |

TABLE II: # of Methods Whose Purity was Changed

| Software | (a) #Targets | (b) #Changed | Ratio (b) against (a) |
|---|---|---|---|
| jEdit | 6,271 | 331 | 0.0527 |
| JFreeChart | 7,790 | 55 | 0.0070 |

### C. Detection Algorithm

This tool detects purity changes by applying the following three steps to a target repository. An overview of those steps is shown in Figure 2.

*1) Step1 (Revealing purity of each method in Java bytecode):* purano [1] analyzes Java bytecode and reveals purity of each method. This tool employs purano and applies it to adjacent revisions.

Purano extracts all methods and variables used in each method. Methods calling impure methods or assigning to field variables are classified into impure method. If an overriding method in a subclass is impure, its overridden method in a superclass is impure.

*2) Step2 (Tracking methods):* by tracking methods from their births to their removals, this tool identifies same methods even if their signatures were changed. This tool employs ECTEC [3] to track methods.

*3) Step3 (Extracting information methods and commits where purity is changed):* purity information is attached to all revisions of each method by using the results of step1 and step2. Then, this tool detects commits and methods where purity was changed.

## IV. Application to OSS Repositories

### A. Target Software

We conducted experiments on two well-known software projects, jEdit and JFreeChart. Table I shows an overview of the target projects. *#Revision* means the number of revisions where at least a source file was added, removed, or changed. *LOC* means the number of lines in source code at *End Date*.

The target projects are written in Java and managed with Subversion. They are compilable with Ant and Maven.

### B. Findings

Table II shows the number of methods whose purity was changed. We found that there were methods whose purity had been changed in software development process. Table III shows the number of methods whose purity was changed to pure or impure only once, or changed multiple times. There are many methods whose purity changes repeatedly, and the frequent purity changes might confuse developers. We investigated whether purity had been changed with or without source code changes. Table IV shows the number of purity changes where source code was changed at the same time. We confirmed many purity changes did not have source code

TABLE III: The Frequency of Purity Changes

| Software | Only Once | | Multiple Times |
|---|---|---|---|
| | To Pure | To Inpure | |
| jEdit | 59 | 115 | 157 |
| JFreeChart | 37 | 18 | 0 |

TABLE IV: # of Purity Changes Where Code Was Changed

| Software | Changed | Not Changed |
|---|---|---|
| jEdit | 258 | 443 |
| JFreeChart | 18 | 37 |

org/gjt/sp/jedit/search/HyperSearchResult.java

```
74      int start = o.startPos.getOffset();
75      int end = o.endPos.getOffset();
76    + System.err.println("#" + i + ": startPos=" +...
77      Selection.Range s = new Selection.Range(
78      start,
79      end
```

Fig. 3: A Part of Changes on r19698 in jEdit

changes. A purity change in a method propagates to other methods. In other words, method purity can change even if its source code is not changed. The results imply developers might change purity of other methods unintentionally.

### C. Bug Detection

We manually investigated changes where purity was changed to impure. We investigated 122 methods whose purity changes to impure with source code changes. Those changes were occurred in different methods. An error output statement was inserted to the 76th line in HyperSearchResult.java on r19698 in jEdit. Figure 3 shows a part of changes on r19698. The change injects a side effect and the statement was removed by r19701, which is two days after. Consequently we concluded the change induced a bug by commit messages or future changes. There are many cases that methods having only a comment "TODO" changed to impure method by implementing its function.

## V. Conclusion

In this paper, we presented a tool to find purity changes in source code repositories. By using this tool, developers can find bugs related to unintentional purity changes. We have applied the tool to two open source systems and there are many methods whose purity was changed repeatedly. In the applications, most of purity changed were not bug inducing, but we found a few problematic purity changes. In the future, we plan to develop an Eclipse plugin and conduct more experiments with developers of open source systems. In addition, we are going to conduct research to make better understanding of whether detected purity changes relate to bug introducing.

## Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers 25220003.

## References

[1] J. Yang, K. Hotta, Y. Higo, and S. Kusumoto, "Revealing purity and side effects on functions for reusing java libraries," in *Proc. of the 14th International Conference on Software Reuse*, Jan. 2015, pp. 314–329.

[2] ——, "Towards purity-guided refactoring in java," in *Proc. of the 31st International Conference on Software Maintenance and Evolution*, Oct. 2015, pp. 521–525.

[3] Y. Higo, K. Hotta, and S. Kusumoto, "Enhancement of crd-based clone tracking," in *Proc. of the 13th International Workshop on Principles of Software Evolution*, Aug. 2013, pp. 28–37.