Enhancing Secure Coding Assistant With Error Correction and Contract Programming

Chen Li, Benjamin White, Jun Dai, Cui Zhang
Department of Computer Science
California State University, Sacramento
6000 J Street, Sacramento, CA 95819, USA
{li3, benwhite, jun.dai, zhangc}@csus.edu

ABSTRACT

As cyber-attacks have become more prevalent in the recent decade, companies and governments have learnt the significant importance of enforcing robust programming practices to ensure software security and reliability during code generation. Various tools have been developed for the purpose of assisting programmers in secure coding, and the initial version of our tool called “Secure Coding Assistant” is one of such development efforts. Designed to support CERT rule violation detection, the tool is featured by “providing a mechanism to detect rule violations early” and by “filling the void of open source tools”. The tool is promising in secure programming education compared to other commercial products, however, the initial version does not provide assistance in error correction, nor does it takes into account the potentials of employing contract programming enforcement to assist users in improving program reliability. To achieve error correction and defect localization for both software security and reliability in Java programs, this paper presents our efforts for the implementations of assisting error correction and enforcing contract programming. Our tool is maintained on GitHub at http://benw408701.github.io/SecureCodingAssistant/.

Keywords

Robust Programming, Software Reliability, Software Security, Eclipse, Java

1. INTRODUCTION

In the recent decade, cyber-attack has become more prevalent. Based on the statistic published on Statista.com, between 2005 and 2014 millions of data records have been breached in the United Stated [1]. For example, a data breach to Heartland Payment Systems in 2008 and 2009 resulted in 130 million records comprised [2]. JP Morgan Chase, the largest bank in the United States, was the victim of a security breach in 2014 impacting over 76 million household accounts and seven million small businesses [3]. In 2015, Anthem Blue Cross, one of the largest health insurance companies in the US, was attacked resulting in about 78.8 million people’s personal information being stolen [4]. Had the security vulnerabilities been detected at the software development stage, the likelihood of those incidents would have been greatly reduced.

Despite of the security vulnerabilities which are exploited by hackers to compromise the system, low software reliability is another contributor to poor quality of software and often leads to security loopholes. The well-known incident of Ariane 5 in 1996 was the result of poor software reliability, which led to the fatal crash of the rocket shortly after its launch. Investigation conducted by the French Space Agency and European Space Agency pointed to an overflow error introduced in the guidance software converting a 64-bit floating-point horizontal velocity to a 16-bit signed integer. The error led to the shutdown of the guidance system and eventually caused the rocket to veer off course. The catastrophic event of Ariane 5 cost European Space Agency 10 years of R&D effort and 7 billion dollars, making the bug the most costly error in history [5]. The disaster could have been avoided if contract programming methodology had been applied to the guidance software development.

The above incidences call for “robust programming practices” to detect and correct any defects in the code, for the purposes of ensuring both software security (in case of any threats coming from hackers) and reliability (in case of any risks from non-malicious activities)[1]. The tool named “Secure Coding Assistant” is our response to the community’s demands and efforts. The initial version [6] mainly targets at solving the insecure coding practices, by providing a mechanism based on CERT rules [7] to detect any rule violations at the stage as early as writing code. To fill the void of open source products, the tool is developed as a non-commercial contribution and released through GitHub to promote secure programming education.

However, with the focus on early defect detection, the initial version of Secure Coding Assistant didn’t contain facilities to provide assistance in error correction. It also didn’t provide support to improve code correctness for the sake of risk mitigation in terms of software reliability, where the scenarios may not have any attackers (i.e. the vulnerability is trigger by other factors). Of course, the design rationales at that time didn’t include the potentials of contract programming, which can be integrated with rule violation in one coherent ecosystem to enforce programming practices to ensure software reliability as well as security. This paper presents our efforts in the current version to fill the afore-mentioned gaps, achieving the enforcement of code robustness, based on assisting error correction and contract programming.

This paper is structured as follows: Section 2 reviews related work, where Section 2.2 introduces more details about contract programming. Section 3 outlines the design, Section 4 gives the implementation details, and Section 5 makes the conclusion.

[1] Following paper [23], this paper uses the terms “secure programming” and “robust programming” synonymously, meaning actually “secure and robust programming”.

2. RELATED WORK

2.1 Existing Tools for Enforcing Secure Programming Practices

Fifteen existing static analysis tools, which were developed to support the enforcement of secure coding practices, have been reviewed by our previous work [6]. As shown in Table 1, although nine of them support early detection, others are late in defect localization, and most of them don't provide much detail regarding vulnerabilities and the mechanisms to find them. The only three open source tools before our initial release are FindBugs [8], ASIDE [9] and PMD [24], with FindBugs lacking support in early detection of rule violation, ASIDE limited in only providing assistance in web application development, and PMD focusing on detection of inefficient code.

Amongst all the tools, only ASIDE supports assistance in automatic fix [9]. However, it follows the OWASP rules and primarily focuses on detecting and fixing vulnerabilities in web application development. In contrast, the Secure Coding Assistant is more generic, following CERT rules [7] and focuses on providing solutions for any software development using Java. Table 1 is the expansion of our previews review of static analysis tools [6]. The highlighted columns compare the newly added features between those tools and our new version.

Table 1. Review of static analysis tools for security vulnerabilities

Tool	Early/Late Detection	Open/ Closed	Error Correct ion	Contract Programming
White Box Testing/Binary Static Analysis	Late	Closed	**No**	**No**
Fortify Static Code Analyzer	Late	Closed	**No**	**No**
Sentinel Source	Late	Closed	**No**	**No**
Klockwork Insight	Early	Closed	**No**	**No**
SecureAssist	Early	Closed	**No**	**No**
Early Security Vulnerability Detector (ESVD)	Early	Closed	**No**	**No**
Static Security Vulnerability Analyzer	Early	Closed	**No**	**No**
Contrast for Eclipse	Late	Closed	**No**	**No**
SonarLint	Early	Closed	**No**	**No**
CxSuite	Late	Closed	**No**	**No**
Goanna Studio	Early	Closed	**No**	**No**
FindBugs	Late	Open	**No**	**No**
Coverity Prevent	Early	Closed	**No**	**No**
ASIDE	Early	Open	**Yes**	**No**
PMD	Early	Open	**No**	**No**
Secure Coding Assistant (Current Version)	Early	Open	**Yes**	**Yes**

2.2 Existing Tools for Contract Programming in Java

Contract programming is also called Design by Contract [10]. It is an approach which helps producing correct and reliable software by specifying the contracts in terms of pre-conditions, post-conditions for methods, and invariants for class, and by checking the contracts automatically at run time. Eiffle [11] is the first programming language which implemented Design by Contract. Most other languages do not provide this built-in mechanism, however, much effort has been made to provide tools for supporting Design by Contract in other programming languages such as Java [12-17], C# [18], PHP [19].

Specifically, Table 2 shows several Design by Contract tools supported for Java [12-17]. Among them, Jass and iContract are comment-based tools, jContractor uses additional methods to define contract and performs contract checking using name conversion, while Modern Jass, Contract4J and Cofoja specify the contract via annotation.

After evaluating those tools, Cofoja [16] was selected for integration with the new version of Secure Coding Assistant. This tool uses annotations like @Invariant, @Requires and @Ensures to specify the invariant, pre-condition, and post-condition contracts, respectively. It provides run-time checking by utilizing annotation processing and bytecode instrumentation [16]. The rationales for this selection are listed below:

- Cofoja is featured with syntactic checking, which provides instant feedback to developers upon the detection of syntactic error in the contracts;
- The enforcement of Design by Contract methodology requires Secure Coding Assistant to be able to analyze the contract. Annotation-based tools have such advantages over comment-based tools in locating and extracting contracts;
- Defining contract via annotations is more elegant, and brings the benefit of code readability and ease of testing;
- Unlike Modern Jass, which doesn't provide Window version of the Eclipse plug-in jar file [20], Cofoja is compatible with any system installed with JDK 6 or higher.

Table 2. Existing tools for Design by Contract in Java

Tool	Implementation	Support Contract Syntactic Checking
Jass	comment	No
Modern Jass	annotation	Yes
iContract	comment	No
jContractor	contract method	No
Contract4J	annotation	No
Cofoja	annotation	Yes

3. Design

3.1 Goal

The goal of the project is to help developers learn and adopt robust programming practices by providing them a mechanism embedded into the Eclipse code development environment. The mechanism is supposed to support assistance to users for both defect localization and error correction, either through rule violation checking or contract programming enforcement.

3.2 Architecture

To build an early detection tool which provides instant feedback to the developers, the new version of Secure Coding Assistant inherits its previous design to run background checking to monitor code changes and to detect rule violations [6]. For code analysis, the abstract syntax tree (AST) which represents the structure of source code is traversed. When a rule violation is detected, a marker is created at the place where a violation occurs. Any subsequent code changes will clear all the markers created and will trigger a new round of AST node traversal.

In the new version of the Secure Coding Assistant, we find that the detection of the absence of "Design by Contract" can be achieved based on the same design, i.e. AST node traversal afore-mentioned to provide corrective solutions to developers. Based on this observation, the enforcement of contract programming is also done in terms of rule violation detection. Specifically, rules used for checking the existence of pre-condition, post-condition, and invariant are added to the RuleFactory along with CERT rules. Upon detection of a rule violation, the newly added method getSolutions() will be called and the solutions will be rendered to developers along with the problem description. The workflow of the new version of Secure Coding Assistant is shown in Figure 1. This Figure is the expansion of our previous design [6]. The highlighted blocks show the expansion of our new version.

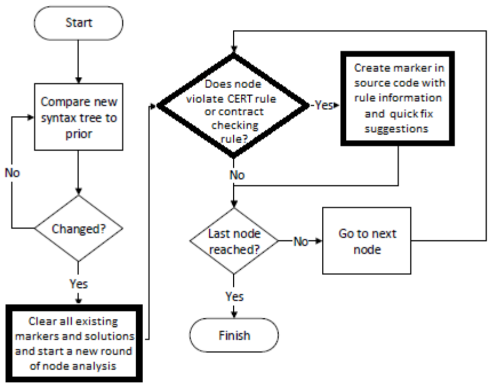

Start
Compare new syntax tree to prior
No
Changed?
Yes
Clear all existing markers and solutions and start a new round of node analysis
Does node violate CERT rule or contract checking rule?
Yes
Create marker in source code with rule information and quick fix suggestions
No
Last node reached?
No
Go to next node
Yes
Finish

Figure 1 Workflow of Secure Coding Assistant.

4. Implementation

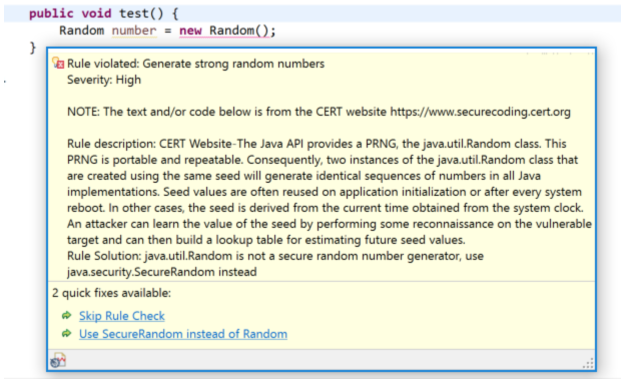
4.1 Implementation of "Quick Fix" Feature

Plugin Development Environment (PDE) provided by Eclipse allows developers to extend and customize the development environment [21]. In the initial version, the extension point *org.eclipse.jdt.core.compilationParticipant* was extended, which enabled the tool to detect the occurrence of an event, such as *build* action, *clean* action or *reconcile* operation, etc., and such events further initiated code analysis for rule violation detection. *org.eclipse.core.resources.problemmarker* was the other point extended, which allowed the tool to customize a problem marker upon the detection of a CERT rule violation [6].

In the new version of Secure Coding Assistant, an extra extension point *org.eclipse.ui.ide.markerResolution* is extended to generate solutions to the problem indicated by rule violations. To bind a resolution generator with the problem marker, the ID of *org.eclipse.core.resources.problemmarker* is assigned to the *markerType* of *org.eclipse.ui.ide.markerResolution*. In addition, two new attributes, *ruleID* and *hashCode* are added to *org.eclipse.core.resources.problemmarker* to serve as a solution map key. Once a problem marker is created, the *getResolutions()* method is invoked with a problem marker object. Then, by appending the hash code value to the rule ID, a key can be easily formed and the *ASTRewrite* object used to generate the quick fixes can be retrieved.

As an infrastructure that describes changes made to AST nodes, the AST rewriter can translate those modifications into text edits. By clicking on the label that is associated with the text edits, the modification will be applied to the source code [22].

To fit into user needs in different scenarios, solutions are offered as two groups of options as shown by Figure 2: to fix the issue or to skip rule check. By choosing the former one, the original source code will be modified upon the solution selected (Figure 3-top), whereas by choosing the latter one, a *@SkipRuleCheck* annotation containing the rule name is added before the method (Figure 3-bottom). As a result, all the violations of the specific rule in the method will be ignored and the corresponding problem marker will be removed. If two or more rules are ignored within the same method, the rule names will be listed together in the annotation.



```
public void test() {
    Random number = new Random();
}
```

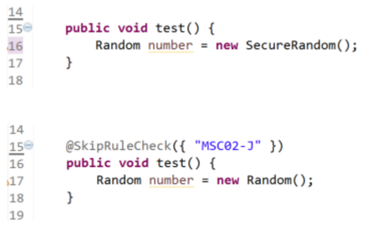
Rule violated: Generate strong random numbers
Severity: High

NOTE: The text and/or code below is from the CERT website https://www.securecoding.cert.org

Rule description: CERT Website-The Java API provides a PRNG, the java.util.Random class. This PRNG is portable and repeatable. Consequently, two instances of the java.util.Random class that are created using the same seed will generate identical sequences of numbers in all Java implementations. Seed values are often reused on application initialization or after every system reboot. In other cases, the seed is derived from the current time obtained from the system clock. An attacker can learn the value of the seed by performing some reconnaissance on the vulnerable target and can then build a lookup table for estimating future seed values.
Rule Solution: java.util.Random is not a secure random number generator, use java.security.SecureRandom instead

2 quick fixes available:
Skip Rule Check
Use SecureRandom instead of Random

Figure 2 Secure Coding Assistant quick fix feature



```

public void test() {
    Random number = new SecureRandom();
}

```

```

@SkipRuleCheck({ "MSC02-J" })
public void test() {
    Random number = new Random();
}

```

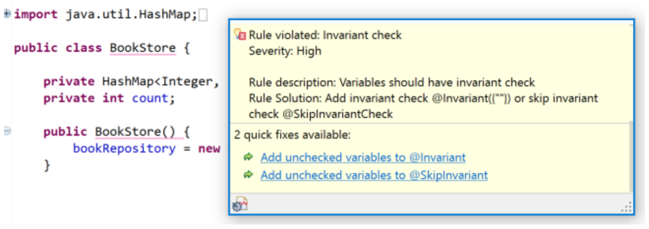
Figure 3 Secure random generator quick fix results: use SecureRandom (top) or skip rule check (bottom)

4.2 Integration of Design by Contract Methodology

In the new version of Secure Coding Assistant, the enforcement of "Design by Contract" is accomplished by the mandatory usage of Cofoja, which means the tool itself only checks for the presence of the contract annotations against Cofoja's library, while leaving the syntactic and semantic checking to Cofoja. The same logic used for CERT rule violation detection is applied for checking the presence of the annotations. Three classes implementing the *IRule* interface are added to the *RuleFactory*, and each method or type declaration node will be evaluated by calling the *violated()* method in all the classes to check for inclusion of the annotations of *@Requires*, *@Ensures* or

@Invariant based on the node's type. As shown by Figure 4, a problem marker will be created if a contract annotation is found missing in the node.

Similar to the CERT rule violation, the checking of the contracts can be skipped by adding the variable name or condition to the value array of *@SkipInvariantCheck* or *@SkipConditionCheck*, placing the annotation before the class or method declaration. As a result, the specified invariant or contract checking will be waived and the problem marker will be removed accordingly, as shown by Figure 5.



```
import java.util.HashMap;

public class BookStore {

    private HashMap<Integer,
    private int count;

    public BookStore() {
        bookRepository = new
    }
```

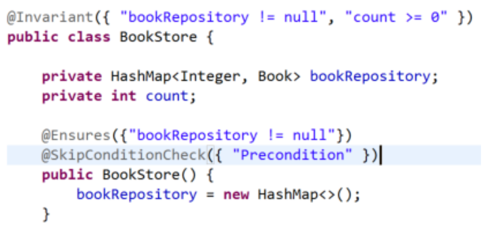
Rule violated: Invariant check
Severity: High

Rule description: Variables should have invariant check
Rule Solution: Add invariant check @Invariant({""}) or skip invariant check @SkipInvariantCheck

2 quick fixes available:

Add unchecked variables to @Invariant
Add unchecked variables to @SkipInvariant

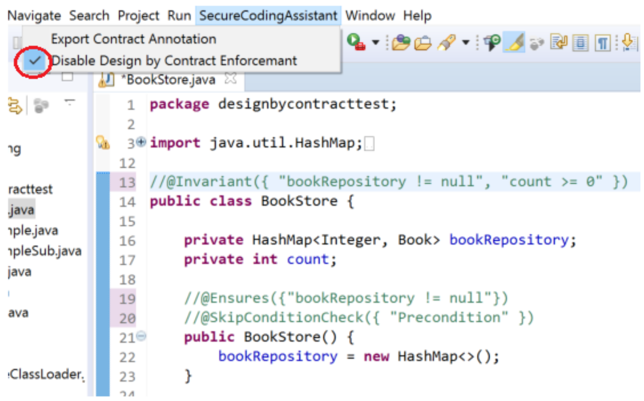
Figure 4 Class and method without contract definition



```
@Invariant({ "bookRepository != null", "count >= 0" })
public class BookStore {

    private HashMap<Integer, Book> bookRepository;
    private int count;

    @Ensures({"bookRepository != null"})
    @SkipConditionCheck({ "Precondition" })
    public BookStore() {
        bookRepository = new HashMap<>();
    }
```

Figure 5 Method skip precondition check

Figure 6 illustrates that the enforcement of "Design by Contract" can also be waived all at once by checking the "Disable Design by Contract Enforcement" button under "Secure Coding Assistant" menu. This function is achieved by extending the *org.eclipse.ui.menus* and *org.eclipse.ui.commands* extension points, in which the three classes used to check the presence of contracts are removed from the *RuleFactory*.

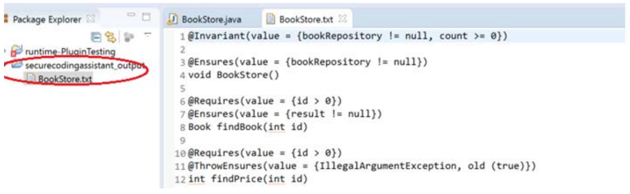


Navigate Search Project Run SecureCodingAssistant Window Help
Export Contract Annotation
Disable Design by Contract Enforcemant
*BookStore.java
ng
racttest
.java
nple.java
npleSub.java
java
ava
ClassLoader

```
package designbycontracttest;

import java.util.HashMap;

//@Invariant({ "bookRepository != null", "count >= 0" })
public class BookStore {

    private HashMap<Integer, Book> bookRepository;
    private int count;

    //@Ensures({"bookRepository != null"})
    //@SkipConditionCheck({ "Precondition" })
    public BookStore() {
        bookRepository = new HashMap<>();
    }
```

Figure 6 Class and method without contract definition and with "Disable Design by Contract Enforcement" checked

In addition, to facilitate the documentation of contracts, the contract annotation and the method signature can be exported by clicking on the "Export Contract Annotation" button under "Secure Coding Assistant" menu, which, as shown by Figure 7, yields a text file with the same name of the class.



```
@Invariant(value = {bookRepository != null, count >= 0})

@Ensures(value = {bookRepository != null})
void BookStore()

@Requires(value = {id > 0})
@Ensures(value = {result != null})
Book findBook(int id)

@Requires(value = {id > 0})
@ThrowEnsures(value = {IllegalArgumentException, old (true)})
int findPrice(int id)
```

Figure 7 Exporting the contracts and method signatures

5. Limitations, Conclusion and Future Work

This paper presents a coding assistance tool, which supports the detection of CERT rule violations, the enforcement of contract programming, and a one-click feature to help users quickly correct the detected code defects. As an open source development, the tool can serve as a practical and efficient application in educating developers on robust programming practices. However, for proof of concept, current implementation priorities are given to only sample rules in a small subset (about one quarter) of CERT library. In addition, the capability of the quick fix feature has not been evaluated for all cases. The future work will focus on expansion of the rule set, thorough tests of the tool features, as well as education evaluations with the tool applied in possible computer science classes with Java programming.

6. Acknowledgements

Acknowledgements and attributions are given to Carnegie Mellon University and its Software Engineering Institute, as this publication incorporates portions of the "SEI CERT Oracle Coding Standard for Java" (c) 2017 Carnegie Mellon University, with special permission from its Software Engineering Institute". Any material of Carnegie Mellon University and/or its software engineering institute contained herein is furnished on an "as-is" basis. Carnegie Mellon University makes no warranties of any kind, either expressed or implied, as to any matter including, but not limited to, warranty of fitness for purpose or merchantability, exclusivity, or results obtained from use of the material, Carnegie Mellon University does not make any warranty of any kind with respect to freedom from patent, trademark, or copyright infringement. This publication has not been reviewed nor is it endorsed by Carnegie Mellon University or its Software Engineering Institute. CERT and CERT Coordination Center are registered trademarks of Carnegie Mellon University. Java is a registered trademark of Oracle. Inc.

7. References

[1] Annual number of data breaches and exposed records in the United States from 2005 to 2015 (in millions). URL= http://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/.

[2] Lewis, D. Heartland Payment Systems Suffers Data Breach. URL= http://www.forbes.com/sites/davelewis/2015/05/31/heartland-payment-systems-suffers-data-breach/#7821b60d2985.

[3] Gushe, D., October 5, 2014. JP Morgan Chase reveals massive data breach affecting 76m households. *The Guardian*. URL= http://www.theguardian.com/business/2014/oct/02/jp-morgan-76m-households-affected-data-breach.

[4] Richman, J. Anthem Blue Cross hack: What you need to know about the health insurer's personal information breach. URL= http://www.mercurynews.com/2015/02/05/anthem-blue-cross-hack-what-you-need-to-know-about-the-health-insurers-personal-information-breach/.

[5] Lions, J.L. Ariane 5 Flight 501 Failure. URL= http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html.

[6] White, B., Dai. J., Zhang, C., 2016. Secure Coding Assistant: Enforcing Secure Coding Practices Using the Eclipse Development Environment. *Proceedings of the National Cyber Summit* (Huntsville, AL, Jun 8-9 2016).

[7] CERT. Carnegie Mellon University. URL= http://https://www.securecoding.cert.org.

[8] Cole, B., Hakim, D., Hovemeyer, D., Lazarus, R., Pugh, W., and Stephens, K., 2006. Improving your software using static analysis to find bugs. *Proceedings of the Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (Portland, Oregon, USA2006), ACM, 1176667, 673-674. DOI= http://dx.doi.org/10.1145/1176617.1176667.

[9] Xie, J., Chu, B., Lipford, H.R., and Melton, J.T., 2011. ASIDE: IDE support for web application security. *Proceedings of the the 27th Annual Computer Security Applications Conference* (Orlando, Florida, USA2011), ACM, 2076770, 267-276. DOI= http://dx.doi.org/10.1145/2076732.2076770.

[10] Meyer, B., 1992. Applying "Design by Contract". *Computer 25*, 10, 40-51. DOI= http://dx.doi.org/10.1109/2.161279.

[11] Meyer, B., 1985. *Eiffel: A Language for Software Engineering*. Technical Report TR-CS-85-19 University of California, Santa Barbara.

[12] Bartetzko, D., Fischer, C., Möller, M., and Wehrheim, H., 2001. Jass — Java with Assertions1 1This work was partially funded by the German Research Council (DFG) under grant OL 98/3-1. *Electronic Notes in Theoretical Computer Science 55*, 2 (2001/10/01), 103-117. DOI= http://dx.doi.org/http://dx.doi.org/10.1016/S1571-0661(04)00247-6.

[13] Rieken, J., 2007. Design by contract for java-revised Department of Information, University Oldenburg.

[14] Kramer, R., 1998. iContract - The Java(tm) Design by Contract(tm) Tool. *Proceedings of the Technology of Object-Oriented Languages and Systems* (1998), IEEE Computer Society, 832856, 295.

[15] Karaorman, M., Holzle, U., and Bruno, J., 1999. jContractor: A Reflective Java Library to Support Design by Contract.

[16] Lê, N.M., 2011. *Contracts for java: A practical framework for contract programming.* Google Switzerland GmbH.

[17] Contract4J. URL= http://https://deanwampler.github.io/contract4j/.

[18] Wu, R.H., 2004. Support for Design by Contract in the C# Programming Language California State University, Sacramento.

[19] Enderlin, I., Dadeau, F., Giorgetti, A., and Ben Othman, A., 2011. Praspel: A Specification Language for Contract-Based Testing in PHP. In *Testing Software and Systems: 23rd IFIP WG 6.1 International Conference, ICTSS 2011, Paris, France, November 7-10, 2011. Proceedings*, B. WOLFF and F. ZAÏDI Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 64-79. DOI= http://dx.doi.org/10.1007/978-3-642-24580-0_6.

[20] Modern Jass. URL= http://modernjass.sourceforge.net/.

[21] PDE. URL= http://www.eclipse.org/pde/.

[22] ASTRewrite. URL= http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2Frewrite%2FASTRewrite.html.

[23] Matt Bishop, Jun Dai, Melissa Dark, Ida Ngambeki, Phillip Nico, and Minghua Zhu. Evaluating Secure Programming Knowledge. In *Proceedings of the10th World Conference on Information Security Education (WISE 10)*, Rome, Italy, May 29-31, 2017 (accepted, to appear).

[24] PMD Java source code analyzer. URL=https://github.com/pmd/pmd.