Helping Student Programmers Identify and Fix Bugs Using Static Analysis Tools

Allyson L. Senger

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Stephen H. Edwards, Co-chair
Margaret Ellis, Co-chair
Clifford A. Shaffer

December 2, 2021
Blacksburg, Virginia

Keywords: static analysis tools, FindBugs, error messages, novice programmers, feedback

Helping Student Programmers Identify and Fix Bugs Using Static Analysis Tools

Allyson L. Senger

(ABSTRACT)

Static analysis tools can be used to help programmers identify problems in their code. However, these tools often assume that developers have some programming background knowledge, so they can be hard to use in an educational context. We investigated the most common FindBugs errors from student code submissions and determined those errors that were related to incorrect solutions to problems and potential struggling for students. FindBugs is a static analysis tool that looks for incorrect patterns in Java bytecode analysis to identify potential coding flaws. For the common errors, we rewrote some of the original FindBugs messages to help students more easily understand the problems with their code. We found that students with at least one FindBugs warning in their final submission to an assignment had more submissions, longer work times, and lower correctness scores than students who did not have a FindBugs warning in their final submission. Adding modified FindBugs feedback to the automated grader resulted in students making fewer submissions and decreasing the length of time required to complete assignments.

Helping Student Programmers Identify and Fix Bugs Using Static Analysis Tools

Allyson L. Senger

(GENERAL AUDIENCE ABSTRACT)

Professional software developers use automated tools when they code to help them catch potential coding problems. These tools are difficult for novice student programmers because they do not have the same level of background as professionals. In this work, we attempted to change the feedback given by these tools so that students could understand it and use it to fix their code. We found that, across all of the undergraduate courses in this study, FindBugs warnings were associated with students having more trouble with assignments. When students could see FindBugs warnings, their time to complete assignments and the number of attempts they made both went down.

Dedication

To the teaching and professional women who inspired my interest in STEM, thank you for the support. I now join you to encourage the next generation of young women to pursue STEM.

Acknowledgments

There are many people I would like to thank for their help and support as I completed my college education. First, I would like to thank my advisors, Dr. Steve Edwards and Professor Margaret Ellis for working with me and advocating for me this past year and a half. I have learned so much working with the two of you. I would like to thank Dr. Clifford Shaffer for his invaluable feedback on improving this research.

Next, I would like to thank my friends who have supported and loved me through my college career. All of you are amazing people, and I cannot wait to see where you go in the future. I want to especially thank the horn section of the Marching Virginians for being my family away from home. You reminded me to laugh a little every day and remember that school is about more than just getting a degree.

Thank you to my grandparents who have always encouraged me to do what I love in all aspects of my life. Lastly, I want to thank my family. Dad, Mom, Ashlyn, and Jenna, you have all helped me keep moving forward when school became too overwhelming. You are the best support system I could have asked for, and I could not have done this without you.

This material is based upon work supported by the National Science Foundation under Grant No. DRL-1740765. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Figures

List of Tables

List of Abbreviations

CS# Computer Science course level

IDE Integrated Development Environment

An IDE is a software tool used to help developers write code. It often includes functionality for creating new files and projects, debugging existing code, and running code without the need for invocation from a command line window.

Computer science courses across universities are often referred to by their level. Introductory classes are CS1 classes, classes at the next level are CS2, and so on. This gives us a way to talk about students with different amounts of programming knowledge without specifying which language they are using in their coursework.

Chapter 1

Introduction

In addition to the compilation and runtime errors that appear in code, potential problems exist that will not be detected by these methods. These include logical errors and likely problematic code. The static analysis tool FindBugs uses Java bytecode pattern recognition to link common incorrect patterns to student-submitted code that follows those same patterns.

FindBugs can help detect problems, but it was still created for use by professional developers. Although we cannot claim that the existing feedback is not useful to novices, the warning messages may require background knowledge of computer science principles that students do not yet have.

When learning to program, students may not know how to interpret such messages or know what underlying issue in the code caused them to appear. The learning curve of interpreting the messages of compilers, IDEs, and runtime tools is a significant hurdle for students. Our goal is to give new programmers the best chance at finding and fixing these bugs on their own without needing to ask an instructor what each warning message means.

We added the FindBugs tool to Web-CAT, the automated grading tool at Virginia Tech, to see if it was helpful for our novice programming students to improve their code more quickly and receive higher final scores on programming assignments. We began with Edwards et al.'s [9] work using FindBugs in the context of short-form programming assignments. Their

Figure 1.1: Research Timeline

work looked at which FindBugs warnings would be appropriate for students, if the presence of FindBugs warnings caused lower correctness score, and if FindBugs warnings indicated that students were struggling with their assignments. Edwards et al. used a subset of the FindBugs warnings that were most applicable to student programmers. They examined the time spent on an exercise, the total number of submissions to an exercise, and the student's final score on an exercise to determine if students were struggling. They also investigated the prevalence of FindBugs warnings in the exercises to see if they were associated with those indicators of struggling. When we added FindBugs to our automated grader for students, we also edited some of the messages to be more easily readable by novice programmers. The same logic was followed in Web-CAT for the static analysis tools CheckStyle and PMD.

A high-level overview of the timeline of this research can be seen in Figure 1.1. This work started by replicating the Edwards et al. study [9], but in the context of larger student programming assignments. We ran FindBugs on four semesters of student data on class programming assignments from Fall 2017 to Spring 2019. This data gave an initial indication of the most frequent errors students would have encountered had FindBugs been added to Web-CAT at that time.

From the Fall 2017 to 2019 data, we were able to exclude warnings that were not relevant

to our students. We also decided to rewrite the warning messages for warnings that could be difficult for novices to understand. The new warning messages were used instead of the FindBugs defaults whenever a student encountered them.

The FindBugs tool was then added to Web-CAT for the Summer II 2021 semester as an initial trial with fewer students than in a fall or spring semester. More messages were rewritten based on the results of the Summer data. From there, FindBugs was added to Web-CAT for the undergraduate Java classes at Virginia Tech for the Fall 2021 semester. We were able to gather more data to compare to Fall 2017 to Spring 2019 data; this helped us examine the potential impacts FindBugs has on student programmers on a larger scale than the summer study.

1.1 Web-CAT

Web-CAT (or the Web-based Center for Automated Testing) [21] is an automated grading tool used by Virginia Tech in its Java programming classes. When students submit an assignment, they are given a score based on the correctness of their code compared to instructor reference tests, the code coverage of their unit tests, and the styling and documentation of their code. The feedback is sorted into categories like test case problems, compiler errors, and formatting issues. An example of the scoring bars can be seen in Figure 1.2, and an example of the more detailed feedback can be seen in Figure 1.3.

Included in the style and documentation category are the tools PMD and CheckStyle. PMD catches problems such as unused variables, while CheckStyle focuses on the formatting of code in a readable way (e.g. placing spaces around the arguments on either side of ==).

Web-CAT shows student progress on assignments across multiple submissions, but only as

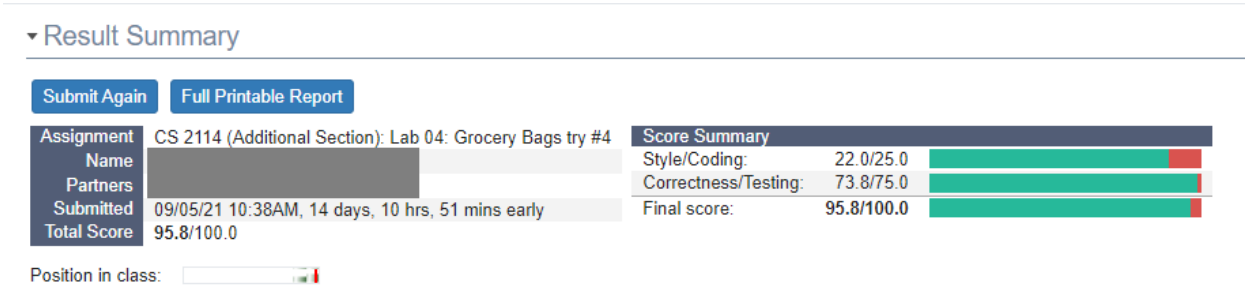

Figure 1.2: Web-CAT Scoring Bars

check-ins. It does not continuously monitor the errors students are making on their local IDE; instead, only the code state at the time of submission is collected, similar to how a repository like GitHub would work.

1.2 FindBugs

FindBugs was developed as a static analysis tool to detect potential bugs in software created by professional developers, which was heavily based on some previous Lint tools. In their paper *Finding Bugs is Easy* [14], Hovemeyer and Pugh explain how their tool works. FindBugs identifies problems by using bug patterns or "places where code does not follow usual correct practice in the use of a language feature or library API." The authors state that quality assurance and unit testing often misses the types of bugs their tool is able to detect.

FindBugs differs from other static analysis tools because it checks for bugs, not just the style in which the code is written like PMD or CheckStyle. Other forms of bug checking, like code inspection, are often helpful, but they require human effort and are swayed by human perceptions of what the code is supposed to do. Dynamic analysis (with unit testing, for example) is helpful because it only shows ways the program can actually fail; however, this

▾ File Details

File	Staff Cmts	Staff Pts	AutoGrade Cmts	AutoGrade Pts	Methods and Conditions Executed
FileIO.java	0	0.0	0	0.0	
groceries/ArrayBasedBag.java	0	0.0	0	0.0	100.0%
groceries/ArrayBasedBagTest.java	0	0.0	0	0.0	
groceries/BagADT.java	0	0.0	0	0.0	
groceries/GroceryBag.java	0	0.0	3	-4.2	96.6%
groceries/GroceryBagTest.java	0	0.0	0	0.0	

▾ Results from Running Your Tests (100%)

The results of running your own test cases are shown below. Click on a failed test to see the reason for the failure and an execution trace that shows where the error occurred.

Passes: 11/11 Errors: 0/11 Failures: 0/11

▶ groceries.ArrayBasedBagTest (0.016 s)
▶ groceries.GroceryBagTest (0.003 s)

100%

▾ Code Coverage from Your Tests (98%)

Code Coverage: 98% (percentage of methods and conditions exercised by your tests)

You can improve your testing by looking for any lines highlighted in this color in your code listings above. Such lines have not been sufficiently tested--hover your mouse over them to find out why.

▾ Estimate of Problem Coverage (100%)

Problem coverage: 100%

Your solution appears to cover all required behavior for this assignment. Make sure that your tests cover all of the behavior required.

For this assignment, the proportion of the problem that is covered by your test cases is being assessed by running a suite of reference tests against your solution, and comparing the results of the reference tests against the results produced by your tests.

Figure 1.3: Detailed Web-CAT Feedback

has the weakness of not being able to test paths for which the developer did not write a test case.

FindBugs uses detectors for bug patterns in several categories: class structure and hierarchy, linear code scanning, control sensitive code, and dataflow. Each of the Findbugs warnings is found based on discrepancies in typical code patterns in one of these categories. For example, NP_NULL_ON_SOME_PATH would fall into the category of a dataflow bug because in some paths of execution, the code will attempt to dereference a null value.

Some bugs are more important to consider in beginner code than others. For the undergraduate courses at Virginia Tech, it is important that we emphasize errors that appear frequently in student code. Many students have issues with unused variables, NullPointerExceptions, ignored return values, and infinite looping. These can all be detected by FindBugs. Some of these problems are detected by the compiler but not all. A compiler will warn a student if their method does not return the type of variable it is expecting. However, calling a function like substring(val) without storing its result into another String is a warning a compiler will not give. Running code will also often detect infinite loops. However, if the code only has an infinite loop in some situations and the developer does not test that situation, FindBugs can detect that an infinite loop would occur if the code was run in that way. It is not as important in this context to have FindBugs detect errors with concurrency or serializability because those are not concepts used in the undergraduate curriculum.

Hovemeyer and Pugh [14] speculated on the cause of errors in code. Some of their ideas were that we as programmers (even experienced programmers) make silly mistakes and that many errors do not show up until runtime. The authors state that "tools such as FindBugs can make a valuable addition to style checkers, since they find a significant number of real bugs while producing only a moderate number of overall warnings." Including FindBugs in Web-CAT is another tool students can use to help improve their coding skills. Student

programmers are inevitably going to make mistakes in their code or test it incorrectly. In this case, their runtime bugs may not appear in their IDE. FindBugs can catch potential runtime errors as well if students had not already seen them in their IDE.

1.3 FindBugs and Web-CAT

Integrating FindBugs into Web-CAT provides another tool for students to find flaws in their code. There is some crossover between FindBugs and other tools like PMD and CheckStyle. For example, PMD and FindBugs will both mark an unused variable as bad practice. However, FindBugs will also find certain logical errors; one example is that it can find branching logic where a variable remains null in one branch and may be dereferenced later in the program, causing a NullPointerException.

All static analysis tools on Web-CAT are run using Apache Ant [2], which is a Java command line tool to build files and run applications. Student code is run against FindBugs, PMD, and CheckStyle using an Ant build file.

1.4 Research Questions

FindBugs is a static analysis tool designed for professional developers. Students learning to program may make different errors from professionals. It is important to determine how FindBugs can be used in a student context and if it is useful for students like it is for professionals. The following questions were examined regarding FindBugs and its use in a student programming context.

RQ1 Which FindBugs warnings are relevant to students completing full-size programming

assignments?

Static analysis tools are often produced for professional developers to streamline their software development process. Since these tools are not designed for students, it is important to investigate if they can be useful when teaching students how to program. We want to see if static analysis tools are able to catch errors that students commonly make, not just more complex errors like problems with multithreading or efficiency.

RQ2 Do the FindBugs warnings correlate with incorrectness in student solutions?

Just because a static analysis tool detects a problem in code does not mean that problem is detrimental to the correctness of the code. We want to examine if FindBugs warnings appear that could have a significant impact on the correctness of student code.

RQ3 Are FindBugs warnings associated with greater struggling on an exercise, in terms of time spent, submissions made, or final score?

Using static analysis tools is beneficial to developers because they can identify problematic code without first needing a test case to fail to indicate a problem. We want to investigate if the feedback provided by FindBugs can help students solve problems in their coding assignments.

RQ4 Does providing and modifying FindBugs feedback cause students to make fewer coding errors and fix them more quickly?

When bugs occur that impact code correctness, it is important that they are fixed quickly. We want to investigate if providing students with modified FindBugs feedback means they write code with fewer errors and if the lifetime of those errors is decreased.

Chapter 2

Review of Literature

This work covers the use of a static analysis tool in an educational context. We review the literature on static analysis tools, programming error messages, and feedback. We also cover some research on the common misconceptions of student programmers.

2.1 Static Analysis Tools

Many kinds of static analysis tools have been adopted in the educational context to help students produce better code more quickly without the need for a professor or teaching assistant to review it first. Edwards et al. [8] used PMD and CheckStyle tools on student code to examine which static analysis errors occurred most often and how those errors varied according to how much experience students had with programming. CheckStyle [5] primarily focuses on coding style, it can detect some potential logical errors like the definition of a covariant equals method. It also catches coding flaws that may make code hard to read, such as overly long lines or lack of spaces between arithmetic operators. PMD [17] has some overlapping function with CheckStyle but looks more at how the code itself is structured. PMD can find problems such as unused imports (which may indicate a programmer has left out an intended feature) or incrementing a loop variable inside the loop itself instead of just in the loop header. Edwards et al. [8] used coursework from five semesters of student submissions to Web-CAT at the CS1, CS2, and CS3 levels to determine the most frequent

static analysis errors experienced by students.

The most common static analysis warning students received from PMD and CheckStyle was a missing Javadoc comment. Javadoc comments are a standard way for Java programmers to document their code so it can be easily understood by another developer. The inclusion of Javadocs in student coursework encourages students to practice good programming habits for a career in the software industry. This warning appears in many student submissions, likely because students often wait to write comments until their code passes all the instructor reference tests, leading to multiple submissions reporting the same problems because Javadocs do not impact the correctness of student code. Across all course levels, formatting and documentation issues represented most of the static analysis feedback for students. Other useful feedback from static analysis tools like PMD and CheckStyle is likely ignored because of an overwhelming amount of formatting feedback that students would have to sift through.

It may seem like the warnings for missing Javadoc comments could be removed so that students only see warnings relevant to the correctness of their code. However, students should be encouraged to document their code. If these warnings were not shown in addition to the rest from static analysis tools, it is unlikely students would document their code. The warnings produced by the automated grader are all relevant, but their importance varies depending on where a student is in the development process.

Student code had more errors in initial submissions than in final submissions across all of the courses. However, the presence of errors from the coding flaws category in this study was more indicative of a final grade than the large amount of formatting and documentation problems. The authors determine that the amount of feedback PMD and CheckStyle produce does not relate to the correctness of the student code. The large volume of feedback could be detrimental to students because it prevents them from finding the important messages

about bugs that could be causing correctness issues in their code.

Delev and Gjorgjevikj [6] also used static analysis tools to examine code written by novice programmers. The introductory programming classes at their institution are taught in C, so they used the static analysis tools Clang Static Analyzer and CppCheck. Clang Static Analyzer primarily uses the control flow of the program to detect problems, while CppCheck looks for bugs not typically found at compile time such as memory leaks or dereferences of null values. Clang had success detecting dead stores and dead code, as well as logical errors like bad assignments and uninitialized values. CppCheck found issues with formatting strings and unused variables.

The authors address a common issue with new programmers: it can be difficult for them to even run the tools we want to use. Therefore, automatic grading tools like Web-CAT help students use these tools without in-depth knowledge about how the command line or static analysis tools work. Automating "significantly lower[s] the entry barrier for novice programmers caused by complicated tools or integrated development environments" [6]. This is important because it allows new programmers to benefit from the tools that were designed for professional developers. The students in Delev and Gjorgjevikj's work also claimed that "the easiest bugs to fix are those found by the compiler or some other tool. This suggests that for students, locating bugs is more difficult than fixing them. . ." This shows a way static analysis tools can benefit student programmers.

In Hovemeyer and Pugh's work [14], they explain using the FindBugs static analysis tool. This tool detects coding flaws by looking at "places where code does not follow usual correct practice in the use of a language feature or library API." The authors state that static analysis tools can be used to help catch errors that would not be found during quality assurance.

Unlike tools such as CheckStyle and PMD, FindBugs works to identify logical flaws in code,

not just stylistic errors or those that are immediately obvious. For example, FindBugs will mark if a particular code execution path could possibly result in a NullPointerException, even if that path never occurs at runtime. FindBugs is a useful addition over other static analysis tools because it can catch true coding errors. Below are some examples of FindBugs warnings that indicate actual coding flaws that would not be caught by CheckStyle and PMD.

1. ES_COMPARING_STRINGS_WITH_EQ This warning indicates that code is comparing String objects with == instead of .equals(). This means that memory addresses are being compared, not the contents of the Strings. String comparisons that should be true will fail when using ==.

2. SA_LOCAL_SELF_ASSIGNMENT_INSTEAD_OF_FIELD This warning occurs when a field and a local variable have the same name. Instead of writing to the field, the local variable is assigned to itself. In a constructor this would appear as name = name instead of this.name = name.

3. UCF_USELESS_CONTROL_FLOW_NEXT_LINE If a conditional statement is accidentally ended with a semicolon, the execution flow of the program is not changed even if indentation makes it look like it should. Likely, a conditional statement should be used to modify control flow, so the semicolon should be removed.

4. NP_NULL_ON_SOME_PATH This warning indicates that a conditional statement will dereference a null value if it is executed.

5. NS_DANGEROUS_NON_SHORT_CIRCUIT Short circuiting occurs when enough conditions of a statement have been checked to determine the final truth value of the full statement. This warning occurs when the non-short-circuit operators are used.

> The remaining conditions could cause exceptions or make changes to variables that were unintended.

Other research has looked at the usefulness of static analysis tools in the context of grading and assessing student code. Striewe and Goedicke [20] compared different static analysis tools to see which were most useful for learning. Several different approaches for analysis exist within the broad category of static analysis tools. One difference between static analysis tools is whether they use source code or byte code analysis. Byte code is only available if code compiles, but student code that is submitted to an online grader is expected to compile (even if it produces warnings). Compilers have some amount of optimization that may remove statements in source code; if we want to warn students about potential dead code, we would have to perform static analysis on the source code. Some static analysis tools work on single files at a time while others run multiple files to complete their analysis. For example, CheckStyle looks only at individual source files, which makes sense because of the types of errors it is intended to detect are contained within a single file.

CheckStyle and PMD can be modified to running only certain checks on a given programming assignment. However, FindBugs either runs all checks or none, meaning Web-CAT receives all of the FindBugs data on the backend. We only show students warnings that are relevant to their assignments. All three of these tools allow the weighting of specific warnings to change, which is important for grading student submissions where some problems are more detrimental than others.

Rutar et al. [13] compared the bug finding tools that exist specifically for Java programming. As previously mentioned, static analysis tools often have overlap in the types of errors they detect, but the location at which those issues are marked varies depending on the tool. This can make it difficult to prevent multiple tools from marking the same error and leading

to confusion for students. Some tools also repeat the same warning multiple times on a single line of code. This repetition is often not important for novice programmers because a warning on the first occurrence of the problem is often enough for them to remedy the problem. It is also difficult to balance the number of false positives and false negatives a static analysis tool generates. For any tool involved in grading student assignments, we want to ensure the deductions occur only for problematic code, not false positives. The authors state that "there does not seem to be any general correlation between the total number of warnings one tool generates and the total number of warnings another tool generates for any given class."

Static analysis tools also have the challenge of categorizing the severity of an error. Unfortunately, Rutar et al. explain that "for any particular program bug, the severity of the error cannot be separated from the context in which the program is used" [13]. While there is no immediate remedy for this, it could still be helpful to attempt to categorize warnings by severity to ensure that the most important problems are the ones resolved most quickly.

Rutar et al. examined multiple Java bug finding tools as well. FindBugs works on single methods of code at a time and tries to match submitted code to the syntax of other code which is known to be suspicious. It investigates syntax and dataflow on the Java bytecode. PMD has the largest overlap with FindBugs of the tools the authors reviewed; it typically looks at style issues that may cause problems (for example, an empty catch statement). PMD, however, runs on Java source code. PMD and FindBugs have run times on the same order of magnitude. ESC/Java (the Extended Static Checking system for Java) works on source code like PMD, but it uses proofs to validate the state of a Java program.

2.2 Error Messages

Another important area of work is how error messages are interpreted by students and how their information is conveyed.

Nienaltowski et al. [16] looked at how students used messages of three different styles to fix coding errors. These categories were created by examining the compiler messages from five languages (C++, Java, Ada, Scheme, and Eiffel). Short form error messages are those generated by most compilers. They typically flag a line number and have some reference to the variables used on that line. Visual inline form messages are shown in the context in which the error occurred. These messages will often highlight the line of code where an error occurred within the source code and give the error message in a terminal or console. Finally, long form errors are those that include error codes and descriptions, potential fixes, line numbers, and other information about a bug.

This study presented students with a section of code containing an error and an error message of one of the three types described above. Students were then asked to answer a multiple choice question to identify the error. The authors used three types of errors: unknown identifiers, incorrect numbers of arguments, and violations of private access. They paired these errors with each form of error message to generate their set of multiple choice questions to investigate the effectiveness of each type of error message.

Students with higher self-reported levels of programming experience identified errors more quickly than peers with less experience. However, only one of the two student groups from the universities in the study had higher correctness of identified errors if they were more experienced. Giving students more information in long form error messages did not necessarily lead to more correct answers, especially among novice programmers. The visual inline form of error messages seemed to be the least effective in this study. Nienaltowski et al. [16] state

that "more detailed messages do not necessarily simplify the understanding of errors but that it matters more where information is placed and how it is structured." This study was performed on a small dataset where students from one university had more experience than students at the other, so there was little consistency or strong conclusions for the study, but it is still important to recognize the impact of different types of error messages in student learning.

Becker et al. [3] examined how the current text-based error message research may not be sufficient for teaching new programmers with industry-level tools. The compiler's diagnosis of the problem likely does not correspond to how a novice would see the problem. The authors give the example from Java of:

```
system.out.println()
^
error: package system does not exist
```

As experienced programmers, we know that the Java package System is capitalized, but this message does not indicate that to a beginner. Also, because of the exact nature of programming, the challenges differ from normal reading and writing struggles. This problem is compounded for students whose primary language is not the same as that of the messages the compiler produces. The authors state that "the level of exactness demanded by a compiler is arguably unique to the world of computing education."

Improved error messages do not only benefit beginners. Experienced programmers are often expected to switch to an unfamiliar language quickly; error messages that explain the actual source of a problem are more beneficial to everyone, yet the actual problem can be difficult to teach a compiler to detect. The failure of compiler error messages to adequately explain

problems to students also impacts instructors. When a student cannot interpret a message, teaching assistants and professors must spend time helping students both understand the message and fix the problem.

Becker et al. [3] give guidelines for modifying compiler messages to be more easily understood by novice programmers. One of the most important is readability. Error messages need to be written in terms that students will understand without a large base of programming knowledge. Novice-friendly messages are not common because most compilers and tools are created for industrial use. Another important factor in understanding an error is recognizing in what context it occurs. This includes information provided by the stack trace, variable states in a debugger, and values in memory while the program is executing. Some IDEs try to highlight the location of an error, but sometimes they only point out where the error first became a problem, not where it initially appeared. For example, in Java the error "class, interface, or enum expected" often appears when missing a bracket to close a class body, method, or constructor. However, the message may indicate to a novice programmer that they need to create an interface or enum, which is rarely the solution to this problem. The authors also suggest that providing examples could help students identify errors in their own code. Having examples may also make it easier to correct an error in the future because students are able to generalize an error instead of just fixing one specific instance.

Other research has been done on how effective messages designed for new programmers are in practice. Marceau et al. [15] looked at how students reacted to error messages in Racket. Specifically, the authors wanted to know if students fixed the error, did something completely unrelated to the error (indicating they did not understand its meaning), or randomly made changes in highlighted code to try to make it work. The IDE used in this study only presented students with one error message at a time, meaning there was no questioning which error students were trying to fix at the time of their edits. This study continuously monitored

student code instead of only seeing snapshots of progress at submission times. Marceau et al. state that the three steps to interacting with an error message include reading it, understanding it, and creating a solution to the problem. They created a rubric to help categorize where students began to have issues with error messages; they especially wanted to investigate if "students [got] stuck earlier in the sequence with particular kinds of errors," because that would show difficulty understanding what the error actually was, not necessarily struggling to find a solution.

An initial attempt to remedy the issues in understanding error messages could try to use easier words and grammar to make the problem more easily understood. However, this does not necessarily work in programming education where the error messages assume some background programming knowledge. It could also be that the error message is not reflective of the underlying error, as Becker et al. [3] discuss.

Marceau et al. also explain that the number of errors a novice must make to benefit from better understanding of error messages is small. If a student produces roughly 5 errors in an hour lab period, they are encountering an error message about every 10 minutes. Even though the total number of errors in a project may be small for novice programmers because of the shorter assignments, the time to resolution for an error is still important for new programmers because they are likely to have more errors than professional developers and resolve those issues more slowly.

2.3 Feedback

Feedback on student work is vital to their improvement. Giving adequate feedback for students to improve is difficult in many fields, but in computer science, it can be challenging because of the amount of underlying concepts needed to fully understand a question or piece

of code. Fernandez Aleman et al. [11] looked at the effects of response-driven feedback for students in computer science. They reasoned that students who had the same groups of incorrect answers to a set of multiple choice questions likely had the same fundamental misunderstanding about a concept. The students were given feedback based on the misconception, but they were not told which questions they had answered incorrectly. According to the authors, "not giving the exact answer makes the test more like a game or puzzle and encourages reflection and cognition" instead of blindly changing answers to get a higher score. A small gap of time between a student's first and second attempt at the questions could indicate that they were trying to guess the answers instead of understanding the feedback. However, this study had to assume the feedback they were giving was relevant to the issues the students were facing because there is not solid evidence that the groupings of answers actually indicate a particular conceptual misunderstanding.

Qian and Lehman [18] also attempted to address student misconceptions with targeted feedback. This study included students in the K-12 school system instead of students in higher education, but the ideas about correcting misconceptions remain the same. The authors provide the concept of variables as one that is frequently misunderstood. New programmers will often assume the names of variables have some meaning to the compiler; that is, if a student writes a variable named 'list', they may assume the computer knows they want to create a list regardless of the actual variable type. This study used short-form programming exercises to find the most common compiler and reference test errors produced. The authors modified the feedback for 10 reference test errors and provided them to one group of students working on a problem but not to the other. The group of students who received the modified feedback were more likely to improve their solutions, with 45% of the modified feedback group improving as opposed to only 34% of the non-feedback group. Students in the modified feedback group also improved more when only considering the most common errors. In

addition, the number of submissions made was lower for the group receiving the modified feedback. One potential problem with the validity of this study is that "when analyzing two feedback messages cases with the worst improvement rates, the results showed that the quantitative analysis labeled some solutions as 'not improved' because some errors were still present even though the error related to the feedback was fixed." This is a limitation of code running against reference tests; solving one error does not mean the program will necessarily be 100% correct.

2.4 Conceptual Understanding of Novice Programmers

Another important topic to understand when working with error messages for new programmers is the misconceptions or logical errors they experience. Eckerdal and Thuné's work looked at how novice programmers understood classes and objects in Java [7]. They wanted to investigate the conceptions students had instead of just misconceptions to figure out how best to bridge the gap between student understanding and correctness of Java design principles. Although the sample size of the study was small, the authors found that a student's understanding of objects and classes tended to occur together. This makes sense because the Java language is built around classes made of objects that are also objects themselves. However, students mostly understood the definition of objects versus classes based on the text of the programs themselves, not the actual concept of an object. Some misconceptions that were common among their students were that objects are just used as placeholders to abstract away variables and that objects are equivalent to variables.

Ettles et al. [10] also investigated the common logic errors made by novice programmers. The authors classified all the errors into either algorithmic problems, lack of understanding of a problem, and general misconceptions. They found that misconceptions most often

caused logical errors. Logical errors are those that compile correctly but fail to produce a correct solution. A major problem with logical errors is that "often there is no readily available feedback on the location or nature of the error." This contrasts to syntactic error which are flagged at compile time. For more experienced programmers, these errors could be tackled with a debugger; however, novice programmers are not always taught to use a debugger early in their education (especially if the tool is fairly complex for the language like C's gdb). This means it is difficult for new programmers to track down logic errors. Some common misconceptions that Ettles et al. [10] noticed in their study was a misunderstanding of how integer division worked, uninitialized values (which sometimes come with a warning from the IDE), and problems with indexing arrays (frequently off-by-one errors).

Butler and Morgan [4] attempted to find difficulties for students in understanding high- and low-level concepts in programming. Something like object-oriented programming has high conceptual difficulty, while a control structure less conceptual difficulty. The authors list variables, decisions and loops, arrays, and testing and debugging as concepts with which students experience moderate difficulty. These mid-level concepts cover a lot of the potential problems that can be caught by static analysis tools. The authors found that a concept with more feedback (like syntax) from the compiler and from autograders was easier for students to understand. High-level concepts are more difficult for students because there is less feedback associated with them.

Other research has been completed on languages that uses different concepts from Java. Adcock et al. [1] looked at pointer errors students made in C++ programs. Pointer errors often create null pointers or memory leaks, which can parallel to the null pointer exceptions and uninitialized objects in a language like Java. These problems are not detected at compile time but can cause errors at program runtime. The authors found that "vast majority of student pointer errors either would not have been noticed at all, or would have been detected

only much later in execution" which supports the use of static analysis tools to help students locate errors in their code without direct interaction with a teaching assistant or instructor. Their study indicated that "about 5/6 of all errors . . . might not have been detected at all without checked pointers; and if they appeared, they would have resulted in later mysterious behavior of the program" [1]. This study shows how potentially helpful static analysis tools could be in helping students resolve their own coding issues.

Chapter 3

Implementation

This chapter explains the initial work with FindBugs in a short-form programming context and how this work evolved from there to implement FindBugs in a longer-form programming context across multiple courses. This includes the results of the replication study from Summer 2021 [19] and how that impacted the data collection for Fall 2021. We also explain why certain errors were excluded from the analysis. In addition, we discuss the implementation problems we encountered when trying to implement the FindBugs feedback into Web-CAT. Figures 3.1, 3.2, and 3.3 show the breakdown of the FindBugs warnings into the different categories defined by the creators and by us for our analyses.

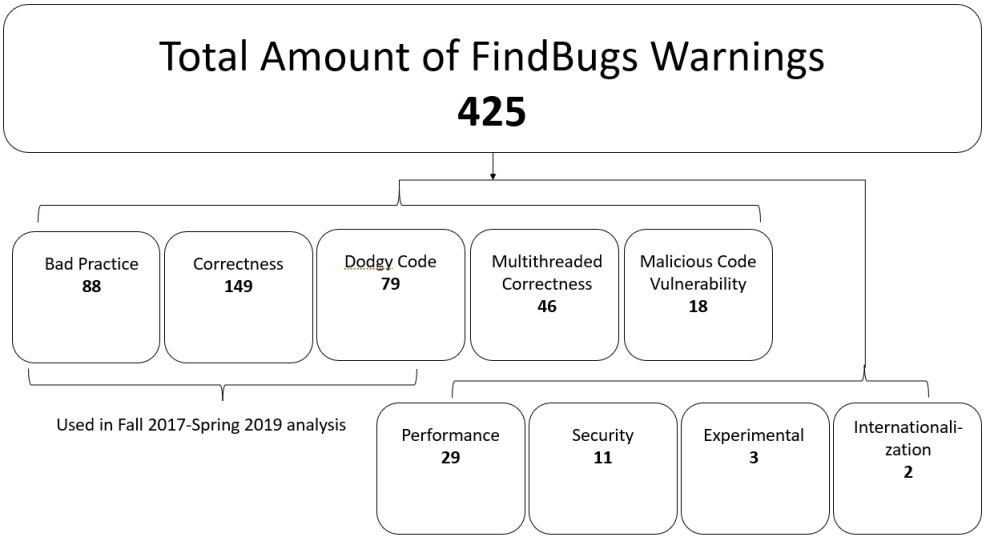

Figure 3.1: Breakdown of FindBugs Messages By Category

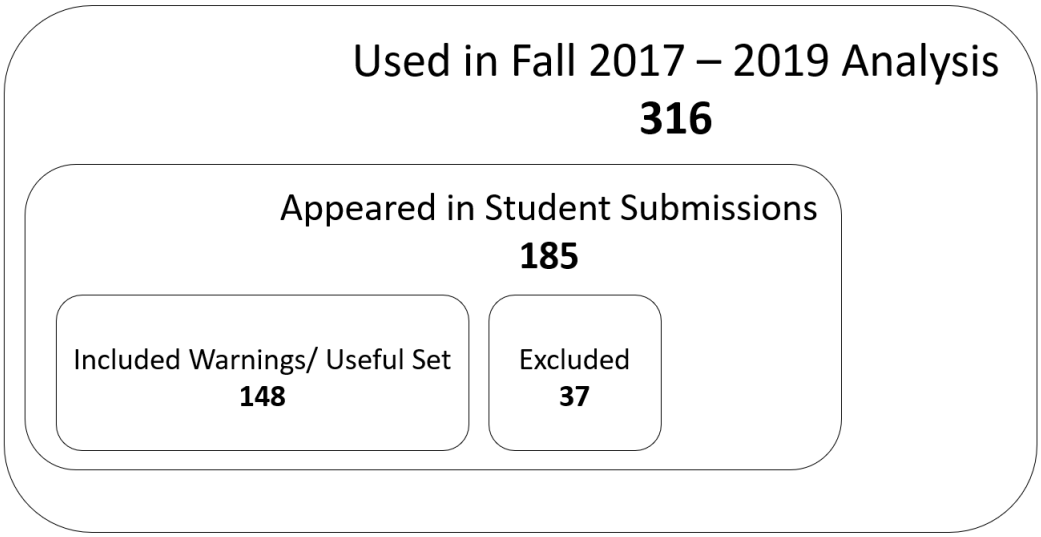

Figure 3.2: Breakdown of FindBugs Warnings Used in Fall 2017-Spring 2019 Analysis

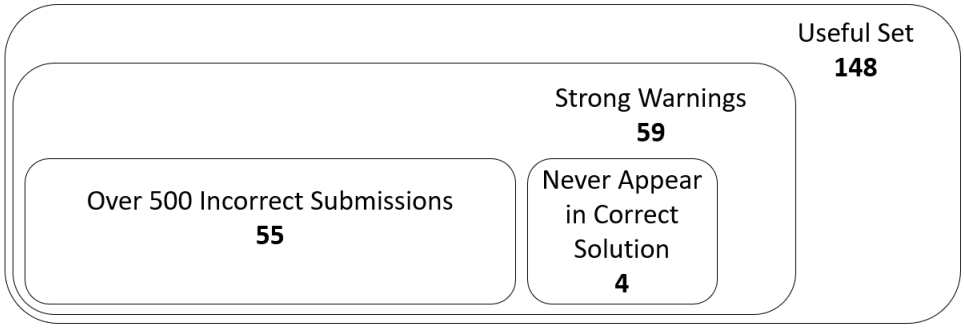

Figure 3.3: Breakdown of FindBugs Warnings From Useful Set

3.1 Preliminary Investigation of FindBugs in Short-Form Programming Exercises

Edwards et al. [9] conducted a study in a CS1 course on Code Workout exercises for undergraduate students at Virginia Tech using the FindBugs tool. CodeWorkout is a tool that assigns short-form programming exercises to students. These can range in size from a single line of code to the definition of a method (usually only 10-20 lines). Using data from the Fall 2017 semester, they discovered that over 92% of students encountered at least one FindBugs

Category	Number of FindBugs Warnings
Correctness	149
Bad Practice	88
Dodgy Code	79
Multithreaded Correctness	46
Performance	29
Malicious Code Vulnerability	18
Security	11
Experimental	3
Internationalization	2

Table 3.1: Categories of FindBugs Warnings

warning during their time in the course. Students who had at least one warning in their submissions also tended to make more submissions, take more time on the assignments, and receive lower scores. This could indicate a relationship between the presence of FindBugs warnings and struggling students. This study was done after the completion of the semester, so students were not able to see the FindBugs warnings their code generated at the time.

This study opted to only work with errors in the FindBugs categories of Bad Practice, Dodgy Code, and Correctness, as these categories are the ones that are most likely to impact novice programmers. In the FindBugs warning set, there are 88 possible Bad Practice warnings, 149 possible Correctness warnings, and 79 Dodgy Code warnings, meaning that this study initially began with 316 possible warnings. The FindBugs categories and the number of warnings in each category can be found in Table 3.1.

Of these 316, only 76 occurred in student submissions at some point during the Fall 2017 semester. From those 76, Edwards et al. [9] removed the warnings that appeared that were unlikely to help students resolve their problems from the dataset. These warnings were either not helpful for improving the correctness of a submission or were not taught in the curriculum. For example, the RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE warning is more complicated than necessary because it checks for null twice; however, checking for null

more than necessary does not impact code correctness. The HE_EQUALS_USE_HASHCODE (which warns against writing an equals method without a hashCode method) was removed because it is not taught in the curriculum at Virginia Tech. The following warnings were excluded from analysis:

- IM_BAD_CHECK_FOR_ODD
- DMI_USELESS_SUBSTRING
- RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE
- NM_METHOD_NAMING_CONVENTION
- BC_EQUALS_METHOD_SHOULD_WORK_FOR_ALL_OBJECTS
- HE_EQUALS_USE_HASHCODE
- NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT

The initial study found 15 warnings that only appeared in incorrect submissions and never in correct submissions. An additional 8 warnings were found to appear 3 times more frequently in incorrect submissions than correct submissions. Table 3.2 shows these 23 warnings and ranks them according to the number of times they appeared in the Summer II 2021 replication study of Web-CAT submissions. Some errors from the original study did not appear in the Web-CAT context.

#	Error Code	Description
37954	ES_COMPARING_STRINGS_WITH_EQ	string comparison with == instead of equals()
22201	EC_BAD_ARRAY_COMPARE	equals() call on arrays instead of Arrays.equals()
12526	UC_USELESS_OBJECT	object created but never used
10149	EC_UNRELATED_TYPES	equals() comparison on different object types
6726	NP_NULL_ON_SOME_PATH	execution path guarantees null value
4719	RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE	check for null after variable has already been dereferenced
4387	FE_FLOATING_POINT_EQUALITY	incorrect comparison of floating point values
3175	ICAST_IDIV_CAST_TO_DOUBLE	incorrect double division
2145	DB_DUPLICATE_BRANCHES	same code in multiple conditional statements
2026	IP_PARAMETER_IS_DEAD_BUT_OVERWRITTEN	parameter value ignored and overwritten
1598	RV_RETURN_VALUE_IGNORED	return value not saved
1520	NP_LOAD_OF_KNOWN_NULL_VALUE	value known to be null at this point in execution
933	NP_ALWAYS_NULL	dereference of null pointer
554	EC_NULL_ARG	equals(null) always returns false
527	UCF_USELESS_CONTROL_FLOW	control statement with no effect
513	INT_BAD_COMPARISON_WITH_NONNEGATIVE_VALUE	guaranteed nonnegative value compared with negative or 0
433	IL_INFINITE_LOOP	infinite loop
322	GC_UNRELATED_TYPES	adding unrelated Object type to collection
184	EC_UNRELATED_CLASS_AND_INTERFACE	comparison of an interface object where other does not implement interface
49	QBA_QUESTIONABLE_BOOLEAN_ASSIGNMENT	assigns boolean value instead of compare with ==
41	UCF_USELESS_CONTROL_FLOW_NEXT_LINE	semicolon after condition, incorrect execution
9	RANGE_STRING_INDEX	string index is out of bounds
0	DMI_COLLECTIONS_SHOULD_NOT_CONTAIN_THEMSELVES	attempt to add collection to itself

Table 3.2: FindBugs warnings from 2019 study [9].

3.2 Initial Rewrite of Warning Messages

Based on the initial Edwards et al. study [9] and the analysis of FindBugs warnings in historical Web-CAT submissions, we configured Web-CAT to give students FindBugs feedback in their assignment submissions starting in Summer II of 2021. The bugs that FindBugs can detect each have a description given on their website [12]. The original messages are also shown in Appendix A. However, these messages are written for experienced programmers; new programmers at the CS1 or CS2 level do not have the knowledge required to interpret the meaning of the messages. Therefore, we have rewritten the messages that may have been difficult for students that occurred in over 10 submissions in the Edwards et al. study [9]. We decided to start with these messages since they were problems students were actually encountering in their coursework.

We rewrote the messages in an attempt to help students understand the problems with their code without needing to get help from an instructor to interpret the meaning of the dense static analysis tool message. Often, students do not need to fully understand a concept to fix the problem and prevent it from occurring in the future. We tried to simplify the language involved in the FindBugs feedback when possible and added examples to concepts that were difficult to understand.

For some messages, we decided that the original FindBugs description of the error was easy enough for novice programmers to understand. If one of these warnings is triggered, Web-CAT gives students the existing FindBugs message.

3.2.1 Examples of Adequate Original Messages

Some of the FindBugs default messages are appropriate for novices, so we have not edited those messages. The following subsections give some examples of FindBugs default messages that we decided were understandable by new programmers.

RANGE_STRING_INDEX

RANGE_STRING_INDEX is the FindBugs warning that triggers when code tries to access a String at an out-of-bound index. The FindBugs message for this warning is:

"String method is called and specified string index is out of bounds. This will result in StringIndexOutOfBoundsException at runtime."[12]

This message informs students exactly what the problem is with their code. Many students will see this problem at runtime locally on their IDE instead of after a submission to the automated grader. However, if this error triggers from FindBugs, students get enough information to resolve the error without needing to know extensive background information; the only additional information they need to know is that Java starts counting at 0 and the index of a String cannot exceed the length of the String. RANGE_ARRAY_INDEX has a similar idea. Students learn about array indexing early in the curriculum, so ArrayIndexOutOfBounds errors are some of the first they learn to address. These FindBugs warnings are beneficial in helping to catch off-by-one errors.

DB_DUPLICATE_BRANCHES

FindBugs will also indicate if a student has the same code in two conditional branches with the error code DB_DUPLICATE_BRANCHES. The message FindBugs provides for this

uses simple language that students could understand so long as they know the meaning of a conditional branch. The feedback students receive is:

"This method uses the same code to implement two branches of a conditional branch. Check to ensure that this isn't a coding mistake."[12]

3.2.2 Examples of Rewritten Messages

Some FindBugs messages require knowledge to interpret that student programmers may not yet have at the CS1 or CS2 level. We rewrote the FindBugs messages that were too advanced for introductory students. The new messages attempt to give help with resolving coding errors without requiring advanced background knowledge.

GC_UNRELATED_TYPES

The FindBugs warning GC_UNRELATED_TYPES attempts to warn students about adding items of multiple types (that are not related by inheritance) to the same collection. This is the description given when running FindBugs without replaced messages:

"This call to a generic collection method contains an argument with an incompatible class from that of the collection's parameter (i.e., the type of the argument is neither a supertype nor a subtype of the corresponding generic type argument). Therefore, it is unlikely that the collection contains any objects that are equal to the method argument used here. Most likely, the wrong value is being passed to the method . . ." [12]

This explanation requires students to understand generic typing well enough to know that only classes with types related by inheritance are able to be added to the same collection. This is more advanced than what students actually need to know to solve the problem in most

cases. Sometimes, students are unaware of the return type of a method. This is especially true when they chain method calls together where intermediate variables are not required. This situation could result in the return type from their chain of method calls not matching the type that could be added to the collection. Here is the rewritten FindBugs message that is now shown on Web-CAT for this warning, which simplifies the wording while explaining to students what is needed to fix the problem:

"A collection has a specific type of objects it contains. Make sure the item you are trying to add to the collection has the same type as the rest of the collection."

This message only requires that students know the definition of a collection. This is a topic that is covered early in the computer science curriculum (usually in a data structures type of course), so most students will recognize the problem immediately: You cannot add an item to a collection if it is not of the type that collection accepts. In addition, for students who have not yet learned about collections, this message is easier for an instructor or teaching assistant to explain with programming terms that a student already understands.

NP_ALWAYS_NULL

Other messages FindBugs provides require some understanding of Java programming, but they are so essential to the language that there is not a better way to describe them. Instead, we try to expand on the FindBugs information to give the student a better chance to understand the source of the problem and fix their code. NP_ALWAYS_NULL is an example of one of these messages. This warning is thrown when a student's code attempts to dereference a null object. It provides the following feedback:

"A null pointer is dereferenced here. This will lead to a NullPointerException when the code is executed." Check that all Objects are initialized correctly.[12]

In this message, we just expanded on the original FindBugs message with a common source of a NullPointerException: an uninitialized object. This message directs students to look back at their objects and make sure none of them are null and causing the exception.

3.3 FindBugs Warnings without Deductions

FindBugs deducts points from Web-CAT submissions in the Style/Coding category because flaws detected by FindBugs can impact correctness or just be general bad practice. FindBugs errors do not always affect code correctness (in terms of instructor-made tests), but they can help students identify potential problem areas in their code. Not all of the FindBugs warnings cause deductions. Only FindBugs warnings that were likely to be coding errors impacting the correctness of a student's code caused point deductions. See Table 3.3 for a full list of these warnings.

Error Code	Description
DLS_DEAD_LOCAL_STORE	variable assigned but not read or used
RV_RETURN_VALUE_IGNORED_NO_SIDE_EFFECT	return value not saved
ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD	instance method writes to static
UC_USELESS_OBJECT	object created but never used
SF_SWITCH_NO_DEFAULT	switch statement has no default case
UC_USELESS_CONDITION	condition always yields the same result
BC_UNCONFIRMED_CAST	unchecked cast, could be invalid
RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE	value known to be nonnull
BC_VACUOUS_INSTANCEOF	instanceof test always returns true
REC_CATCH_EXCEPTION	exception caught, but not thrown in try block
UC_USELESS_VOID_METHOD	void method appears to have no side effects
SA_FIELD_DOUBLE_ASSIGNMENT	e.g., field_x = field_x = 12; (probable typo)
DB_DUPLICATE_BRANCHES	same code in multiple conditional statements
URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD	public or protected field never read
UUF_UNUSED_PUBLIC_OR_PROTECTED_FIELD	public or protected field is never used
NP_LOAD_OF_KNOWN_NULL_VALUE	value known to be null loaded
DMI_HARDCODED_ABSOLUTE_FILENAME	absolute instead of relative path
SF_SWITCH_FALLTHROUGH	switch needs breaks or returns
RV_RETURN_VALUE_IGNORED_INFERRED	method called, but return value ignored
IM_AVERAGE_COMPUTATION_COULD_OVERFLOW	possible overflow with large values
NP_NULL_ON_SOME_PATH_MIGHT_BE_INFEASIBLE	possible NPE in some execution
UCF_USELESS_CONTROL_FLOW	conditional statement does not change execution
SA_LOCAL_SELF_ASSIGNMENT	e.g. x = x (probable typo)
INT_VACUOUS_BIT_OPERATION	bit operation does not change values
RCN_REDUNDANT_NULLCHECK_OF_NULL_VALUE	known null value checked again
NS_NON_SHORT_CIRCUIT	combined conditional statement uses | and &
UWF_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD	public or protected field not written
NP_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD	deref of unwritten field
QF_QUESTIONABLE_FOR_LOOP	for loop incrementing unlikely variable
SA_LOCAL_DOUBLE_ASSIGNMENT	e.g., x = x = 5 (probable typo)
NP_IMMEDIATE_DEREFERENCE_OF_READLINE	no check readLine contains information
ICAST_QUESTIONABLE_UNSIGNED_RIGHT_SHIFT	upper bits of shift discarded
NP_DEREFERENCE_OF_READLINE_VALUE	no check readLine contains information
EQ_UNUSUAL	no normal pattern for typechecking 'this'
NP_NULL_ON_SOME_PATH_FROM_RETURN_VALUE	return value deref. without null check
DLS_DEAD_LOCAL_STORE_OF_NULL	local variable assigned null then not used
UCF_USELESS_CONTROL_FLOW_NEXT_LINE	same control flow if branch is taken or not
IA_AMBIGUOUS_INVOCATION_OF_INHERITED_OR_OUTER_METHOD	check invocation is correct method
DMI_NONSERIALIZABLE_OBJECT_WRITTEN	non-serializable object sent to writeObject()
RCN_REDUNDANT_COMPARISON_OF_NULL_AND_NONNULL_VALUE	known null compared to known nonnull
RV_DONT_JUST_NULL_CHECK_READLINE	null check of readLine() not used again
RCN_REDUNDANT_COMPARISON_TWO_NULL_VALUES	comparison of two known null values
UC_USELESS_OBJECT_STACK	object performs changes without side effect

Table 3.3: FindBugs warnings without Web-CAT deductions

For example, DLS_DEAD_LOCAL_STORE is a frequently-occurring warning from student code. This warning indicates that an object is created that does not have any effect in the program execution. Sometimes, this error comes from a student's code being unfinished; if they have not implemented all functions, then some objects may not be used. Students may have also created more objects than they needed for testing and forgot to remove the extras. However, this warning can also indicate to students that they are not interacting with the object even if they think they are. A dead local store could help a student realize that another section of their code that should be modifying or using the object is incorrect. Because we cannot be sure which situation has occurred when FindBugs encounters a dead local store, we do not include a deduction for this warning.

3.4 Initial Study of FindBugs Warnings in Student Code

During the summer of 2021, we conducted a replication study of Edwards et al.'s [9] paper on using FindBugs in a short-form programming exercise context with students in the Fall 2017 section of CS 1114: Introduction to Software Design. We performed a replication study [19] to show that FindBugs is still a valuable tool in longer programming assignments across all of the Java classes at Virginia Tech. CS 1114 is a standard CS1 course, and CS 2114 is a standard CS2 course. CS 3114 is a more advanced data structures and algorithms course that students take after several CS2 courses.

CS 1114 submissions had an average of 4 FindBugs warnings per submission, while CS 2114 and CS 3114 had higher averages of 17.3 and 13.5 warnings per submission respectively. Because of the increased size of assignments from the CodeWorkout context of the Edwards et al. study, more FindBugs warnings appeared in long-form programming assignments. In contrast to the 7.42% of submissions containing a useful warning in the Edwards et al. study,

56% of submissions contained a useful warning in our analysis of the data from Web-CAT. See Figure 3.2 for how we determined the useful set of warnings for the Summer 2021 study.

We used four semesters of data from Fall 2017 - Spring 2019. We ran the FindBugs tool against each submission to Web-CAT during this timeframe to see how frequently FindBugs warnings would have been triggered had this tool been implemented at the time. Before we incorporate live feedback for students into Web-CAT, we had to determine which FindBugs warnings were appropriate in the Web-CAT context. FindBugs [12] has 425 patterns in byte code that are problematic. However, for a student population, we only care about those warnings in the categories of Correctness, Bad Practice, and Dodgy Code. Categories involving efficiency or threading, for example, are not relevant in our Java curriculum. A total of 185 different FindBugs messages appeared in student submissions across the four semesters mentioned above, which made up our initial useful set (see Figures 3.2 and 3.3. Then, we went through the list and removed those warnings that were harmless, were excluded in the original study, or did not fit well with our curriculum. For example, we removed IJU_SETUP_NO_SUPER (which indicates that the setUp() method for unit testing should include a call to super.setUp()) because we do not require this in our JUnit testing. We also removed NM_CLASS_NAMING_CONVENTION (which explains that class names should be nouns with each word capitalized) because it was unlikely to affect the correctness of student code. In total, we excluded 37 warnings from our initial useful set, leaving a total of 148 warnings to make up our useful set.

The Edwards et al. paper also defined a strong set of warnings as those that, when present, resulted in an incorrect submission. We used a threshold of at least 500 failed submissions as the cutoff for our strong set (a total of 55 warnings). The strong set also included warnings that always resulted in incorrect submissions; we required these to appear at least 25 times to be counted. We ended up with 4 warnings that never resulted in correct submissions in

the Fall 2017 to Spring 2019 data set. Our strong set consisted of 59 total warnings. These are the warnings we focused on for the rest of the study.

We next focused on if those 59 warnings were associated with students struggling to complete programming assignments. The three areas we looked at were the number of submissions a student made for an assignment, the amount of time taken to complete the assignment, and the final correctness score of the assignment. We divided the final submissions for assignments into two categories: final submissions that included at least one warning from the strong set and final submissions that did not include any warnings from the strong set.

The first measure of student struggling on assignments was an increased number of submissions to that assignment. We found that students who had warnings from the strong set in their final submission had 14.89 submissions on average while those without averaged only 10.18. This is shown in Figure 3.4. This could indicate students were struggling to complete the assignments because of problems that FindBugs could have potentially caught. Adding FindBugs in as a diagnostic tool could show students the source of problems earlier, allowing them to fix them sooner in their development process.

Secondly, we used total work time until the final submission as another metric to see if students were struggling. Since Web-CAT only has a record of the total time between submissions, we had to do some estimation to determine work session length (amount of work between submissions, accounting for time not spent programming like sleeping) for a student on an assignment. We decided to use 3 hours as a cutoff for length of a work session, because 86% of the student work sessions fell under this limit. Any gap in submissions longer than 3 hours was set to be the average time between submissions for that student on an assignment to help produce a more accurate estimate even though we only have periodic check-in points using Web-CAT. Students with strong FindBugs warnings in their final submission took an estimated 5.83 hours to complete an assignment while those whose final submissions did not

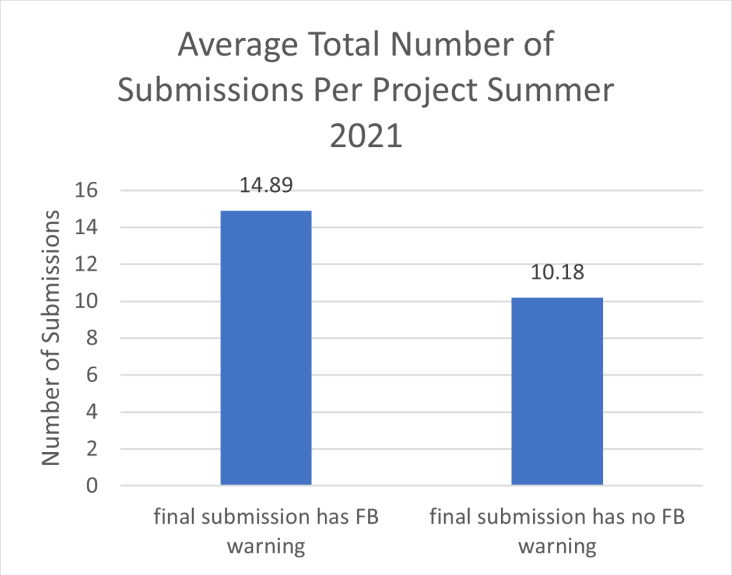

Figure 3.4: Historical Average Number of Submissions Per Project With and Without FindBugs Warnings

have FindBugs warnings averaged 4.59 hours. This is shown in Figure 3.5. Course, student, and the existence of an error in the student's submission history for that assignment were all significant and showed that strong warnings anywhere in a submission history were likely to indicate an increased total work time.

The final area we examined was the correctness score a student received on an assignment based on instructor reference tests. Those students who had a FindBugs warning in their final submission to an assignment passed, on average, 89.77% of instructor reference tests, while those without a FindBugs warning passed 92.75% of tests. This is shown in Figure 3.6.

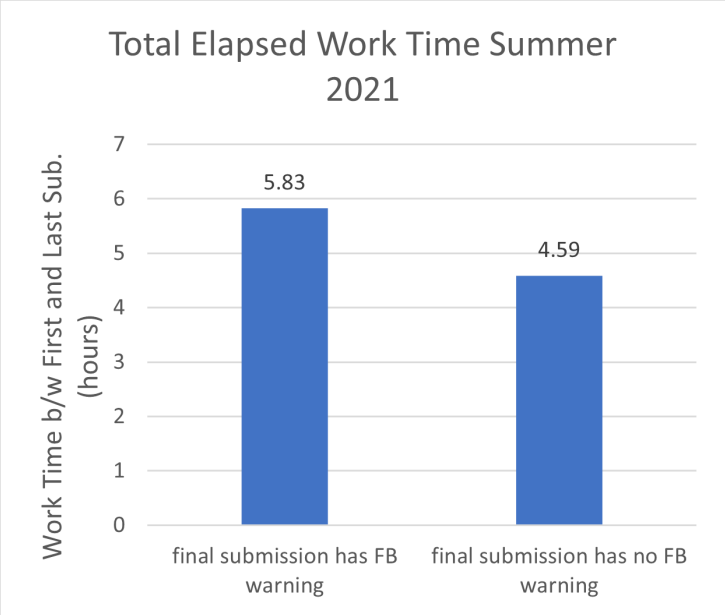

Figure 3.5: Historical Total Elapsed Time Between First and Last Submissions With and Without FindBugs Warnings

3.5 Updating Messages After Replication Study

After running the replication study of Edwards et al. [9] over the summer, we [19] determine the set of errors that were relevant to our students in the Web-CAT context with long-form programming assignments. We have a total of 59 warnings that appeared in more than 500 incorrect submissions, indicating they likely cause problems for students getting their code correct (see Figure 3.3.

We decided to rewrite the strong warnings from Figure 3.3 because they represented challenges to our students. This set of warnings differed from the ones found in the Edwards et al. study[9]; therefore, we decided to rewrite the messages that appeared for this context as well. The list of these warnings and their rewritten messages can be found in A.

Some of these warnings were consistent with the Edwards et al. [9] study in the short-form programming assignment context, but many errors that were generated in Web-CAT did not

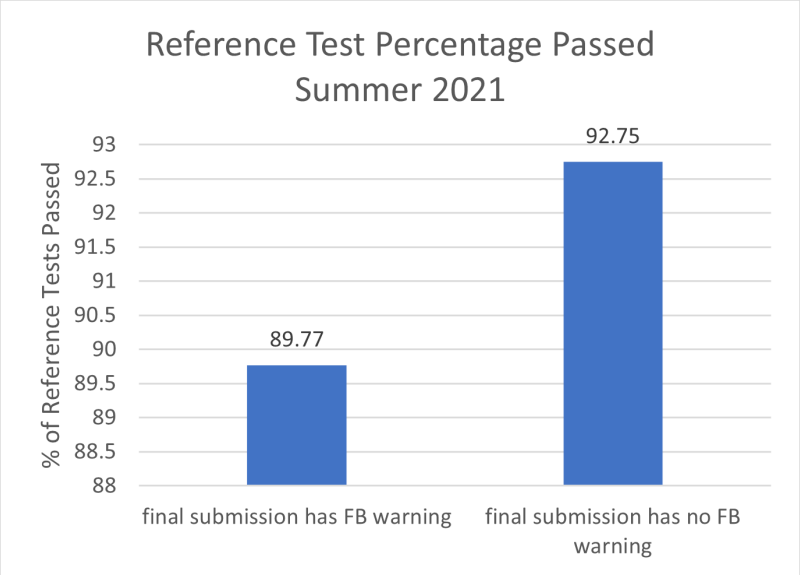

Figure 3.6: Historical Reference Test Percentages of Final Submissions With and Without FindBugs Warnings

appear in that original study. This is likely because the longer programming assignments resulted in more problems, and also different problems. The short-form context often had students writing individual methods or classes, while longer programming assignments require multiple classes with many methods. This could be the cause of a more diverse set of errors in the Web-CAT context.

3.6 Excluded Messages From Replication Study

From the set of warnings that appeared in student code between Fall 2017-Spring 2019, we decided to exclude 35 that students had triggered. Since the analysis was completed after these semesters, students never saw these errors because FindBugs was post completion of the courses. The remainder of this section describes why some warnings were removed from analysis.

Two of the warnings that appeared in the Edwards et al. study [9] were added back to the analysis. We opted to include BC_EQUALS_METHOD_SHOULD_WORK_FOR_ALL_OBJECTS and NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT. These warnings were triggering because of instructor code in the short-form programming exercises, but they are important messages for the students to receive for longer programming assignments. In CS 2114, there is a large focus on writing equals() methods, and both of these FindBugs warnings relate to potential problems with an equals method. BC_EQUALS_METHOD_SHOULD_WORK_FOR_ALL_OBJECTS indicates that an equals method should take an Object as its parameter and not make an assumption about the type of object that will be passed to the method. NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT requires that the equals method always returns false if given a null argument.

We opted to remove some warnings from our analysis because they either were unlikely to represent a true problem in the code or because they do not align with the content we teach in our classes. The warnings that were not included can be found in Appendix A.2.

Many of the warnings we excluded involved the use of hashcodes which is not taught in the Virginia Tech coursework. While it may be common practice in industry to write a hashcode method for every class that overrides equals, this is not something we have historically taught students at this level.

We also excluded warnings that related to JUnit testing, like IJU_SETUP_NO_SUPER and IJU_TEARDOWN_NO_SUPER because we do not require students to make a super.setUp() call, nor do we require the use of a teardown method at all. These warnings do not indicate problems for our students.

Other warnings, such as those involving naming conventions or overflow of numerical values are unlikely to impact the correctness of student code. The warnings against comparisons

with signed bytes and unsigned right shifts occurred a low number of times in our data collection because these are not concepts taught for use in our Java courses. If students are using these complex ideas in their code, they likely know what they are doing or we need to redirect them to a different code structure anyway.

The last large chunk of excluded errors is those relating to serialization and cloning. These are not concepts we explore in the curriculum, so those warnings have been removed from analysis to prevent false positives

The full list of the excluded errors can be found in A.2, along with the category of FindBugs warning to which each belongs.

3.7 Implementation Problems

We started using the FindBugs tool in Web-CAT toward the beginning of the Fall 2021 semester in three undergraduate Java courses and one graduate course. There have been several problems we have had to fix during the time it has been active.

First, we had problems with some of the FindBugs messages being shown to students. In fact, the Web-CAT file in which the FindBugs warning appeared would no longer be displayed because of HTML validation errors on the page contents. This was determined to be a problem with the < and > characters included in some messages causing the HTML to be incorrectly structured. This problem was due to non-HTML-compatible syntax in messages generated by FindBugs itself, and postprocessing was added to fix the issue so the resulting HTML would be displayable.

The ampersand character (&) is also a special character used in HTML markup, and other FindBugs messages also included this symbol in a way that produced invalid HTML. This

means that any FindBugs messages related to the logical AND operation had to be escaped to allow Web-CAT to generate HTML files for a student to view their code.

Whenever the default FindBugs message included information about student-specific variables or method names, we included it and added the modified message afterward.

There is also a slight problem in the visualization of how Web-CAT shows deductions for FindBugs warnings. We have FindBugs deducting from the style category of points, and the deductions show in the main score bar for the assignment. However, in the section of the results page that breaks down deductions by which file they come from, we are not seeing FindBugs deductions, which makes it take longer for students to locate the problematic statements in their code.

3.8 Potential Performance Impacts

Before adding FindBugs to the Web-CAT analysis, we were concerned that the amount of extra time required to run the tool would make it impossible for us to use during a normal semester. We decided to turn on FindBugs feedback in Summer II of 2021 to see what impact the tool had on running student submissions. We discovered that the static analysis checks on Web-CAT are run in parallel with the reference tests, so the additional time required to run FindBugs did not significantly impact the throughput of the automated grader.

Chapter 4

Results

This chapter discusses the results we received from giving students FindBugs feedback during the Summer II 2021 session. It also discusses the results from Fall 2021 and answers the research questions found in both the initial Edwards et al. study [9] and the Senger et al. replication study from Summer 2021 [19].

4.1 Summer Feedback Results

Throughout the Summer 2021 sections of our Java classes, we were able to turn on FindBugs warnings to give students more feedback on their code. We included the rewritten messages for the errors that appeared in the Edwards et al. study [9] since those were FindBugs warnings we knew students had struggled with in the initial study. Some of the FindBugs warnings were understandable by new programmers without rewriting the message. For example, the FindBugs message for SF_SWITCH_NO_DEFAULT reads "This method contains a switch statement where default case is missing. Usually you need to provide a default case." This message should be clear to a student who has decided to use a switch statement. However, a message such as "This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used." for DLS_DEAD_LOCAL_STORE could be misunderstood because students at the CS1 and early CS2 level may not know what counts

as a variable read and therefore may misunderstand the message. We made modifications to this message to read "This local variable has a value, but the value was never used. Make sure the value is used where intended, or remove it if it is not needed." This explanation does not require a student to understand the meaning of a read. It also suggests possible solutions to the problem.

Students in the summer had fewer FindBugs issues appear in final submissions (on average 2.2 per final submission) than those students from previous semesters, likely because they were actually able to see and correct the FindBugs warnings between submissions. This also resulted in fewer bugs total on average across all the submissions a student made for an assignment, decreasing from an average of 91.48 FindBugs issues per history versus 11.29 per history in the summer. This is a drastic difference that has several possible explanations. In the Summer II semester, students were able to see the FindBugs warnings in their feedback which was not the case for the historical data. It is less likely students will fix errors that they cannot see. A summer session course also has shorter assignment periods and may result in students making fewer submissions; our courses now use submission energy to make sure students do not submit too rapidly, which was not true in the Fall 2017 to Spring 2019 data.

The summer session functioned as a trial run for FindBugs before using it in the Fall 2021 semester in all of the Java programming courses using Web-CAT. The results of the Summer II 2021 study can be seen in Figures 4.1, 4.2, and 4.3.

4.2 Fall 2021 Results

During the Fall 2021 semester, we enabled FindBugs to run on all student code submissions for 4 courses: CS 1114 Introduction to Software Design, CS 2114 Software Design and

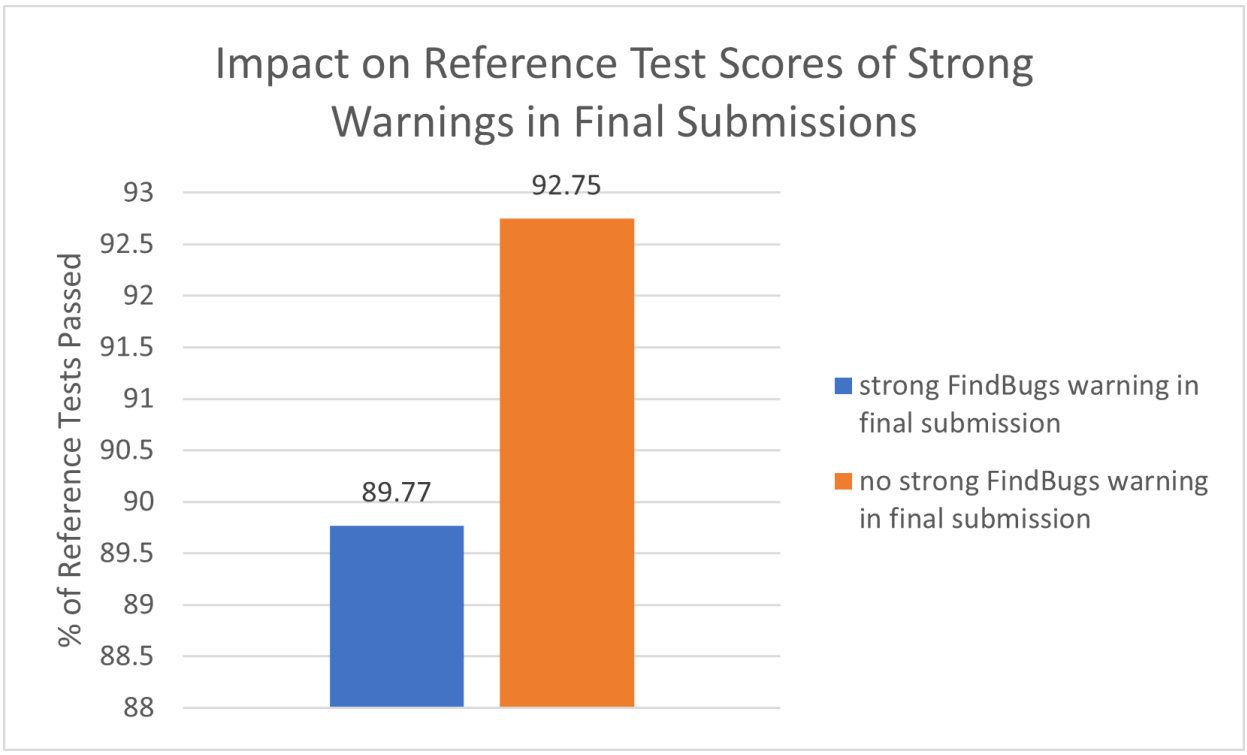

Figure 4.1: Percentage of Reference Tests Passed When a FindBugs Warning Does and Does Not Appear in Final Submission

Data Structures, CS 3114 Data Structures and Algorithms, and CS 5040 Intermediate Data Structures and Algorithm Analysis. These courses all use Java as the primary programming language. The first three courses are undergraduate courses while CS 5040 is a graduate courses used to replicate CS 3114 for graduate students who did not have the equivalent of CS 3114 in their undergraduate studies. The FindBugs tool was turned on a few weeks into the semester, so student feedback for the first few labs and projects in each course did not include FindBugs messages. However, FindBugs has run on all labs and projects since it was turned on. The Fall 2021 data set includes the majority of the projects from the semester.

Using the set of 59 errors gathered in the Senger et al. [19] study from the summer semester, we rewrote the default FindBugs messages that were not present in the historical data. Out of the 59 errors in our error set, we rewrote the messages for 39. These errors represent

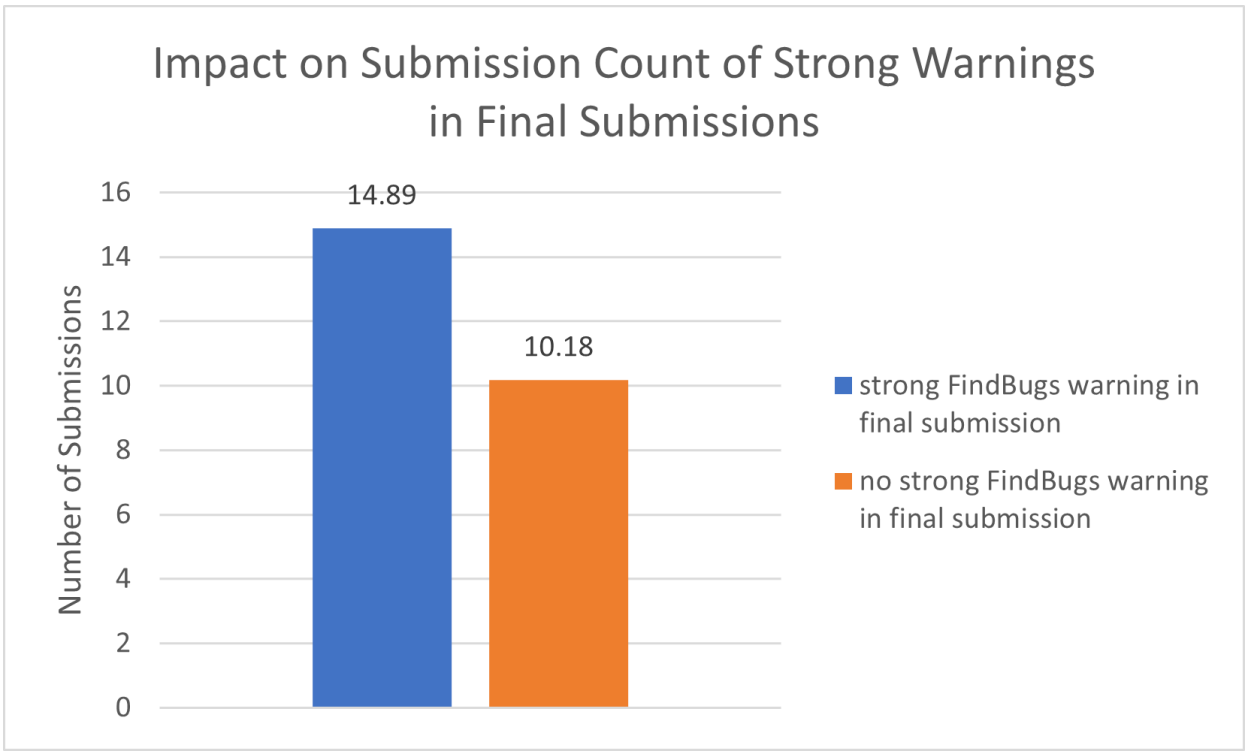

Figure 4.2: Average Number of Student Submissions Per Assignment When a FindBugs Warning Does and Does Not Appear in Final Submission

the most common problems in student code that impacted the correctness of the instructor reference tests run on Web-CAT. Not all messages that appeared in Fall 2021 also appeared in Summer 2021. We selected more of the messages that seemed like they may be difficult to interpret that did not appear in the Summer set to rewrite for Fall 2021.

Although we did collect FindBugs data, there were some complications with CS 3114 and CS 5040. In CS 3114, students were encouraged to work in teams to complete assignments. Because FindBugs runs on compiled submissions and one group member submitted, only the group member's submission had a FindBugs report; however, Web-CAT counts these as two or three different submissions when partners are included for an assignment, meaning that our submissions data did not match up with our FindBugs reports. In CS 5040, a new assignment configuration was created that did not have FindBugs enabled, so we do not

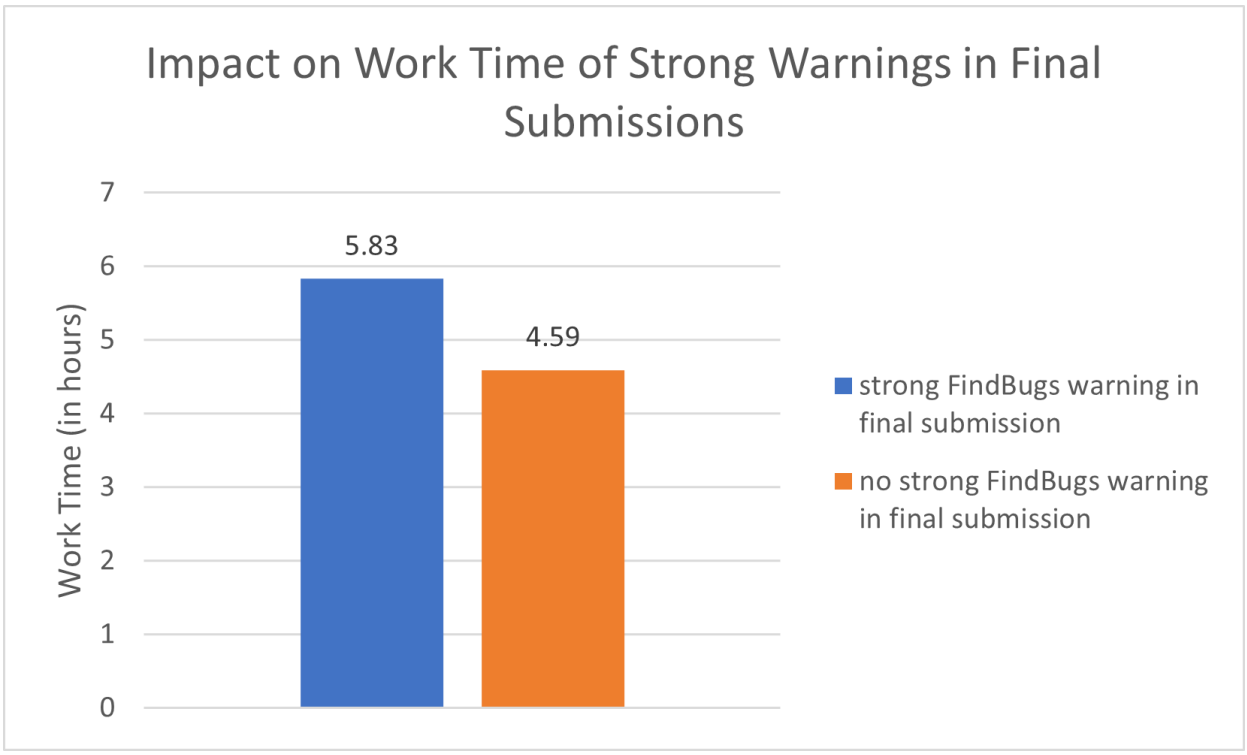

Figure 4.3: Worktime When a FindBugs Warning Does and Does Not Appear in Final Submission

have data from that course. Therefore, these results will compare Fall 2021 data from CS 1114 and CS 2114 against historic data for the CS 1114 and CS 2114 courses.

The first three research questions are from the Senger et al. [19] study. The same investigations were used with our new Fall 2021 data. RQ4 (the impact of providing FindBugs feedback) is a new addition to this research.

4.2.1 RQ1: Which FindBugs warnings are relevant to students completing full-size programming assignments?

During the Fall 2021 semester (once FindBugs was turned on), FindBugs ran on 27,140 student submissions from CS 1114 and CS 2114. CS 1114 had 9 programming projects,

while CS 2114 had 3 major programming projects.

The Fall 2021 data contained 98 different FindBugs warnings from CS 1114 and CS 2114 student submissions. Those warnings that occurred more than 100 times (across all submissions) are shown in Table 4.1. The most common warning to appear in Fall 2021 courses was DLS_DEAD_LOCAL_STORE, which indicates a variable assignment to a variable that is not used in the rest of its scope. It is reasonable that this is the most common warning to appear because it does not always cause a code failure, even though it is not good practice. This warning can also indicate that students have not yet completed their submissions and will be using those variables later in their work process. However, some of the top warnings student received are concerning for code correctness, specifically ES_COMPARING_STRINGS_WITH_EQ (which does memory address comparison instead of content comparison of Strings) and NP_UNWRITTEN_FIELD (which attempts to read a field that has not yet had a value written to it). It is important that FindBugs is recognizing these errors because they indicate coding problems that our students may not have been able to locate otherwise.

It is also important to examine if students are seeing FindBugs warnings enough for them to be relevant to improving their code. When we examined unique submissions, 13,188/27,582 of the submissions contained a FindBugs warning. This means that 47.81% of compiling submissions Web-CAT received generated at least one FindBugs warning. A compiling submission is one that compiles in Java without any errors. This value is slightly down from the Senger et al. [19] percentage of 56% across four semesters of data.

When we examined the entire submission history of a student for a single assignment, we found that 2,700/4,570 histories contained a FindBugs warning, which is 59.1%. This means that for each assignment, just over half of the students were shown a FindBugs warning at some point in their work on that assignment. The student could have seen any of the

#	Error Code	Description
32419	DLS_DEAD_LOCAL_STORE	variable assigned but not read or used
9661	RV_RETURN_VALUE_IGNORED_NO_SIDE_EFFECT	return value not saved
4653	UWF_UNWRITTEN_FIELD	field is never assigned a value
3610	ES_COMPARING_STRINGS_WITH_EQ	string comparison with == instead of equals()
3369	NP_UNWRITTEN_FIELD	dereference of uninitialized field
3143	EC_BAD_ARRAY_COMPARE	equals() call on arrays instead of Arrays.equals()
1906	UC_USELESS_OBJECT	object created but never used
1219	UC_USELESS_CONDITION	condition always yields the same result
1176	BC_EQUALS_METHOD_SHOULD_WORK_FOR_ALL_OBJECTS	equals(Object o) should not assume type for o
1126	EC_UNRELATED_TYPES	equals() comparison on different object types
1034	NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT	equals(Object) should check for null argument
850	NP_NULL_ON_SOME_PATH	execution path guarantees null value
774	UR_UNINIT_READ	read of uninitialized field
740	BC_VACUOUS_INSTANCEOF	instanceof test always returns true
622	DLS_DEAD_LOCAL_STORE_SHADOWS_FIELD	unused variable with same name as field
566	ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD	instance method writes to static
564	IL_INFINITE_RECURSIVE_LOOP	infinite recursive loop
426	EQ_GETCLASS_AND_CLASS_CONSTANT	equals() breaks in subclass due to class literal
424	IP_PARAMETER_IS_DEAD_BUT_OVERWRITTEN	parameter value ignored and overwritten
414	CO_SELF_NO_OBJECT	likely unintentional covariant compareTo() defined
403	FE_FLOATING_POINT_EQUALITY	incorrect comparison of floating point values
364	NP_NULL_ON_SOME_PATH_EXCEPTION	exception path guarantees null value
356	DMI_INVOKING_TOSTRING_ON_ARRAY	toString() method on array is not useful
292	RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE	null check after variable already deref.
256	UC_USELESS_VOID_METHOD	void method appears to have no side effects
233	ICAST_IDIV_CAST_TO_DOUBLE	incorrect double division
232	ES_COMPARING_PARAMETER_STRING_WITH_EQ	string comparison with == instead of equals()
218	EQ_SELF_NO_OBJECT	defines covariant version of equals()
218	SF_SWITCH_NO_DEFAULT	switch statement has no default case
216	EC_NULL_ARG	equals(null) should always return false
201	SA_FIELD_SELF_ASSIGNMENT	e.g., field_x = field_x; (probable typo)
194	SF_DEAD_STORE_DUE_TO_SWITCH_FALLTHROUGH	overwritten value in switch, fallthrough
194	SF_SWITCH_FALLTHROUGH	switch needs breaks or returns
187	UUF_UNUSED_PUBLIC_OR_PROTECTED_FIELD	public or protected field is never used
182	EQ_OVERRIDING_EQUALS_NOT_SYMMETRIC	overriding superclass equals not symmetric
173	EC_ARRAY_AND_NONARRAY	comparison b/w array and not array
168	RV_RETURN_VALUE_IGNORED	return value not saved
158	EQ_SELF_USE_OBJECT	defines covariant version of equals()
157	DB_DUPLICATE_BRANCHES	same code in multiple conditional statements
157	NP_ALWAYS_NULL	dereference of null pointer
136	BC_IMPOSSIBLE_CAST	always incorrect cast
131	RpC_REPEATED_CONDITIONAL_TEST	e.g., if (x == 0 || x == 0)...(probable typo)
123	SA_FIELD_DOUBLE_ASSIGNMENT	e.g., field_x = field_x = 12; (probable typo)
113	SA_LOCAL_SELF_ASSIGNMENT_INSTEAD_OF_FIELD	field and param of same name, set to param
103	GC_UNRELATED_TYPES	adding unrelated Object type to collection
102	SA_LOCAL_SELF_ASSIGNMENT	e.g. x = x (probable typo)

Table 4.1: FindBugs warnings from Fall 2021 Data

warnings; they are included in this percentage if they encountered any warning in any of their submissions to an assignment. However, we cannot guarantee that students viewed their code to read the warning message.

Lastly, when we examined individual students throughout all of their coursework (across multiple classes), we found that 922/1,000 unique students saw a FindBugs warning at some point in the coursework that Web-CAT graded. This means that only about 8% of students never encountered FindBugs in any submission. The most frequent errors are shown in Figure 4.4.

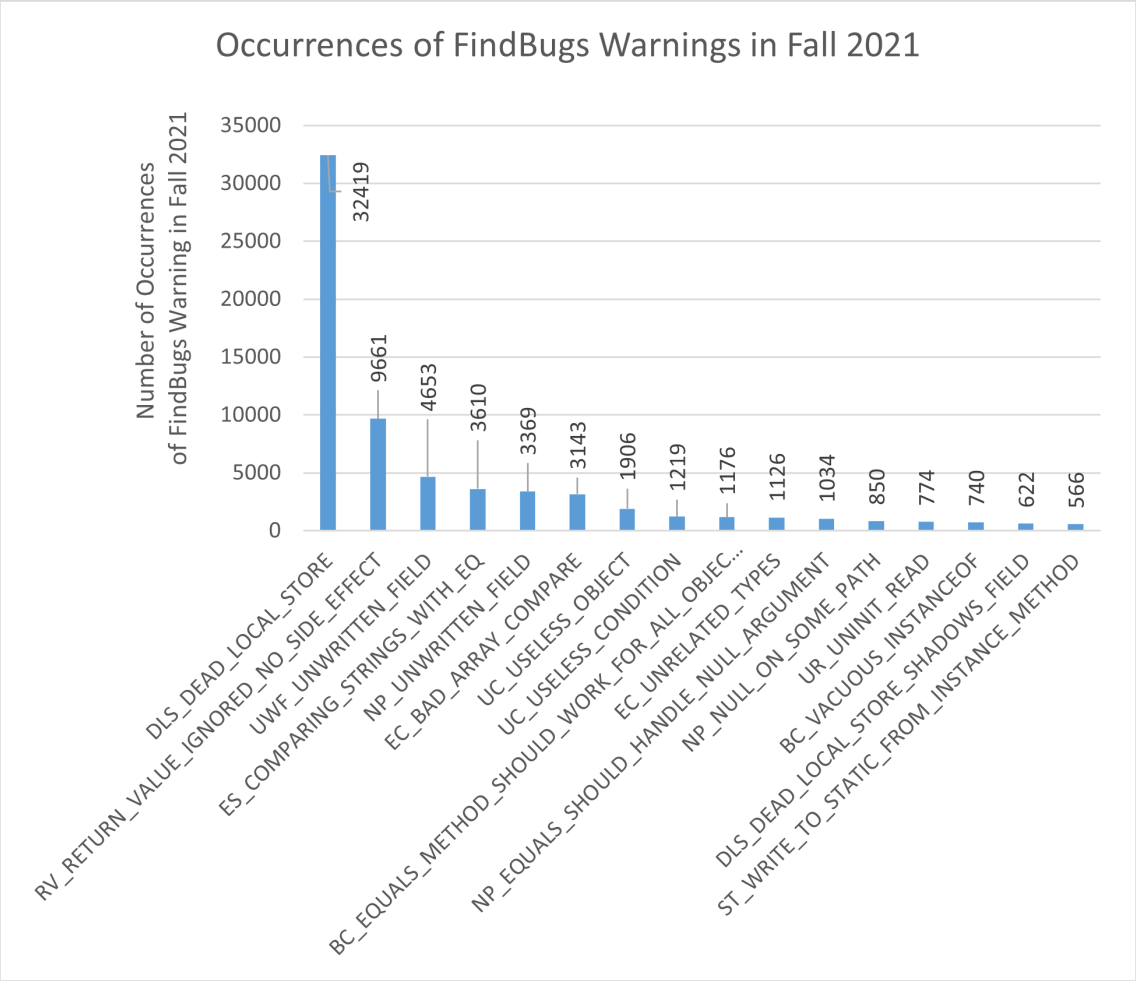

Figure 4.4: Frequency of FindBugs Warnings in Fall 2021

Looking at the values for individual submissions, histories, and individual students, we can see that FindBugs has a high potential impact on our students while they are completing their coursework. Even though FindBugs was designed as a tool to help professional developers, we can see that it still generates warnings that are triggered by student programmers.

4.2.2 RQ2: Do the FindBugs warnings correlate with incorrectness in student solutions?

To determine if FindBugs warnings were related to correctness in student code, we looked at the lifetimes of FindBugs warnings. We used the linux `diff` command to synchronize line positions across successive submissions by a student, then considered Warnings that reoccurred on the same line after synchronization to be the same warning recurring. To look at the impact of FindBugs warnings on correctness, we looked at the score difference between the first appearance of a warning and the first submission in which it was missing. We ran an ANOVA test which produced an F ratio of 177.9 with $p < 0.0001$ based on whether or not a warning was fixed. Fixing the warning or not had an impact on the percentage of instructor reference tests passed. The mean gains are shown in Figure 4.5. Which specific FindBugs warning code was given also had a significant impact, with an F ratio of 4.15 and $p < 0.0001$.

When looking at the individual warning codes, some had more of an impact on a student's score than others. When we refer to gain, we are referring to the increase in percentage of instructor reference tests passed. Table 4.2 shows the warnings with low p values and the difference in gain between an unfixed submission and a fixed submission. It also shows the Cohen's d value, which is the difference in gain over the standard deviation of the gain.

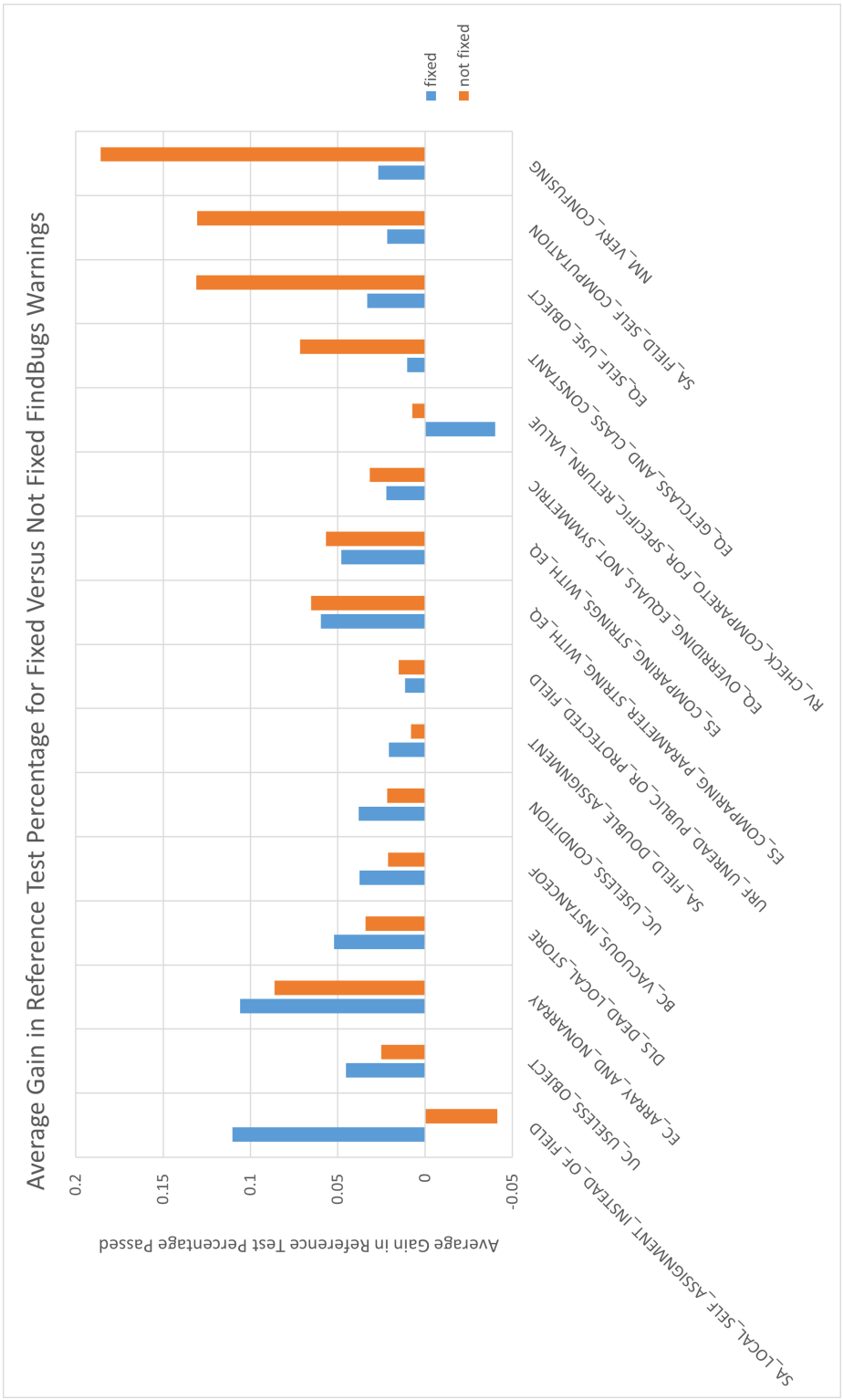

Figure 4.5: Average Gains When a FindBugs Warning Was Not Fixed Versus Fixed for Low p Values

Error Code	Gain w/o Fix	Gain w/ Fix	Gain Diff	Cohen's d	Prob > |t|
UC_USELESS_OBJECT	0.0251	0.0453	0.0203	0.1624	0.0112
EC_ARRAY_AND_NONARRAY	0.0863	0.1059	0.0196	0.0918	< 0.0001
DLS_DEAD_LOCAL_STORE	0.0341	0.0522	0.0180	0.1442	0.0177
BC_VACUOUS_INSTANCEOF	0.0214	0.0376	0.0162	0.2127	0.0132
UC_USELESS_CONDITION	0.0218	0.0379	0.0161	0.1924	0.0057
SA_FIELD_DOUBLE_ASSIGNMENT	0.0082	0.0209	0.0127	0.1327	0.0094
URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD	0.0151	0.0115	-0.0036	-0.0666	0.0104
ES_COMPARING_PARAMETER_STRING_WITH_EQ	0.0654	0.0598	-0.0056	-0.0334	0.0131
ES_COMPARING_STRINGS_WITH_EQ	0.0569	0.0479	-0.0090	-0.0600	< 0.0001
EQ_OVERRIDING_EQUALS_NOT_SYMMETRIC	0.0318	0.0222	-0.0096	-0.1006	0.0312
SA_LOCAL_SELF_ASSIGNMENT_INSTEAD_OF_FIELD	-0.0414	0.1103	-0.0426	-0.3464	0.0414
RV_CHECK_COMPARETO_FOR_SPECIFIC_RETURN_VALUE	0.0074	-0.04	-0.0474	-0.4867	0.0418
EQ_GETCLASS_AND_CLASS_CONSTANT	0.0716	0.0103	-0.0613	-0.3645	0.0294
EQ_SELF_USE_OBJECT	0.1311	0.0332	-0.0980	-0.5067	< 0.0001
SA_FIELD_SELF_COMPUTATION	0.1304	0.0217	-0.1087	-0.4480	0.0269
NM_VERY_CONFUSING	0.1858	0.0270	-0.1589	-0.7284	0.0035

Table 4.2: FindBugs warnings with Significant Impact on Student Score

Some of the warning codes produce results opposite to what we would expect, having a lower gain when the warning is removed from a submission than when it remains in a submission. However, there are a few possible explanations for these values. First, students will ignore some errors while they are trying to fix others. We cannot tell which error a student was trying to fix between submissions. There is also no incentive for students to fix warnings that do not cause a score deduction. In the CS 2114 feedback style, students would have to click through each individual code file after receiving full points on reference tests to see if there were any FindBugs warnings remaining that did not cause deductions. For the warnings to be most effective, students need to see them and edit their code accordingly. The list of warnings that do not causes deductions is shown in Table 3.3. All other warnings that were in the FindBugs "Correctness" category resulted in a -2 point deduction each, while warnings in the "Style" or "Bad Practice" categories resulted in a -1 point deduction each. The "Style" warnings for DLS_DEAD_LOCAL_STORE_SHADOWS_FIELD, ICAST_IDIV_CAST_TO_DOUBLE, ICAST_INTEGER_MULTIPLY_CAST_TO_LONG, and EQ_DOESNT_OVERRIDE_EQUALS were promoted to -2 point deductions because of their strong association with program correctness problems.

4.2.3 RQ3: Are FindBugs warnings associated with greater struggling on an exercise, in terms of time spent, submissions made, or final score?

To determine if students were struggling on assignments, we used elapsed time between first and last submissions, total number of submissions, and a student's final score on an assignment. We wanted to see if students who encountered FindBugs warnings in their final submissions were having more trouble completing assignments than those who were not.

We found that 1,465/4,454 final submissions (or 32.9%) in the Fall 2021 semester had a FindBugs warning in the final submission.

First, we investigated if the percentage of instructor reference tests students passed was impacted by the presence of a FindBugs warning in the final submission. Final submissions that contained a FindBugs warning passed an average of 86.0% of instructor reference tests. In contrast, final submissions without a FindBugs warning passed 92.0% of tests on average. This effect was found to be significant when an ANOVA test was run, producing an F ratio of 173.0 with $p < 0.0001$. This is shown in Figure 4.6.

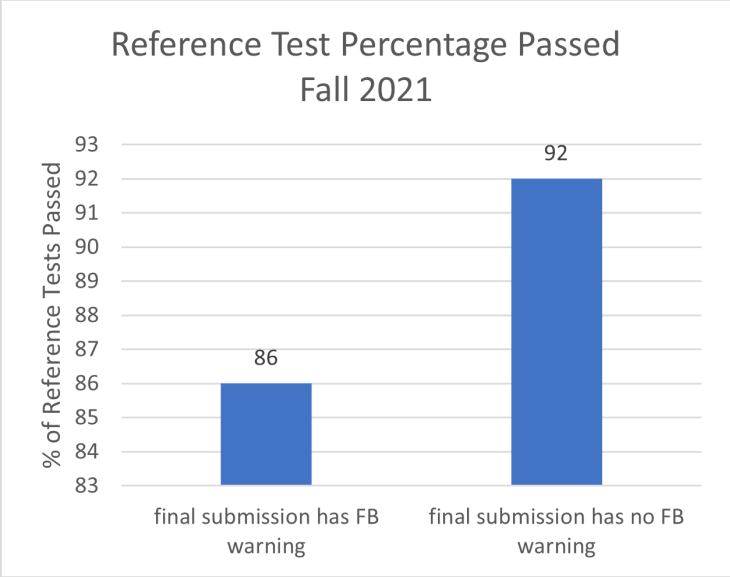

Figure 4.6: Reference Test Percentages of Final Submissions With and Without FindBugs Warnings

Which assignment a student was working on also had an impact on their final score. The CS 1114 assignments had averages anywhere from 97.1% to 69.7% with fluctuations up and down in reference test percentage averages between projects with no constant pattern. The CS 2114 assignments had gradually decreasing averages on scores for assignments over the course of the semester from 97.5% to 87.0%. The overall average on CS 1114 assignments was 88.5% and in CS 2114, 93.2%.

We also examined the elapsed time between the first and last submissions a student made to an assignment. CS 1114 assignments took students an average of 20.5 hours, and CS 2114 assignments took an average of 26.9 hours. Final submissions containing a FindBugs warning also took students longer to complete. A final submission with a FindBugs warning had an elapsed time of, on average, 24.3 hours, while a final submission without a FindBugs warning had an elapsed time of 21.6 hours on average. This is shown in Figure 4.7. The ANOVA test for the significance of these values produced an F ratio of 4.57, $p = 0.0325$.

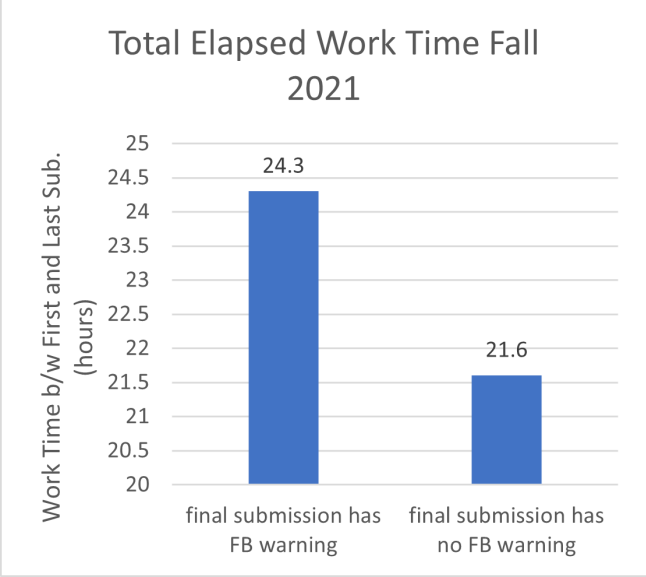

Figure 4.7: Total Elapsed Time Between First and Last Submissions With and Without FindBugs Warnings

Lastly, we looked at how many submissions a student made. In CS 1114, the average number of submissions was 5.4, and in CS 2114, the average was 7.9 submissions per assignment.

Students who had a FindBugs warning in their final submission made an average of 7.2 submissions, while those without a FindBugs warning made 5.7 submissions on average. An ANOVA test showed that the presence of warnings was statistically significant with an F ratio of 26.2, $p < 0.0001$. The average number of submissions are shown in Figure 4.8. The CS 1114 assignments fluctuated in terms of number of submissions required, but the CS 2114

assignments had an increasing trend. This is likely due to the assignments in the CS 2114 class becoming more complex throughout the semester.

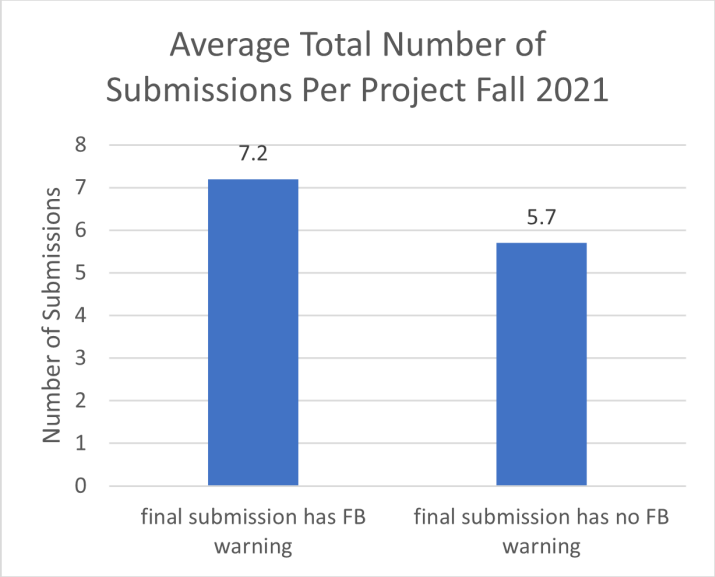

Figure 4.8: Average Total Number of Submissions Per Project With and Without FindBugs Warnings

4.2.4 RQ4: Does providing modified FindBugs feedback cause students to make fewer coding errors and fix them more quickly?

Note that this discussion of results will include some comparisons of CS 3114. This is because, although the averages are off due to the discussion of CS 3114 group work challenges with running FindBugs in section 4.2, the general trends are still important.

One thing we were hoping FindBugs would help with is the total amount of time taken for students to complete their assignments. We hoped that providing feedback in the form of FindBugs warnings would enable students to fix errors in their code more quickly and in turn pass instructor reference tests more quickly. While not all final submissions pass

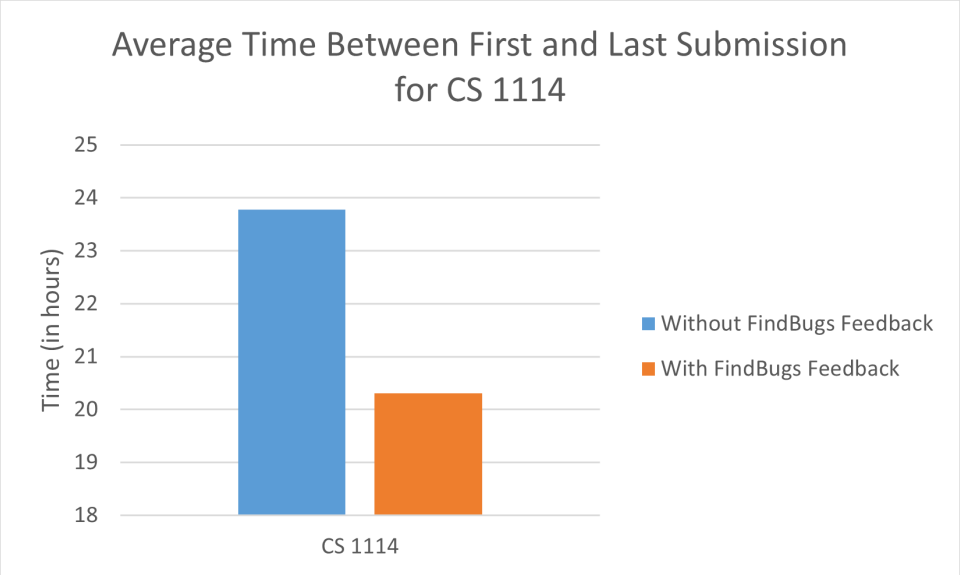

Figure 4.9: Time Difference Between Historical and Fall 2021 Data for CS 1114

every reference tests, students generally try to pass as many as possible before their final submission.

Figures 4.9, 4.10, and 4.11 show the average times between the first and last submissions to assignments by course. We compared the time taken from our historic (Fall 2017 - Spring 2019) data and our Fall 2021 data. In all three undergraduate CS courses (1114, 2114, and 3114), there is a decrease in the average elapsed time between first and last submissions when FindBugs is included in the Web-CAT feedback. This indicates that FindBugs could have an impact on how long it takes students to solve their programming assignments. Note that the CS 3114 data is based on the submission of a single group member counting as multiple submissions to account for the other group members.

Figures 4.12, 4.13, and 4.14 show the difference between the historical data and the Fall 2021 data in terms of number of submissions made to an assignment. Students who are receiving better feedback on submissions will likely make fewer submissions because they will hopefully solve their problems more quickly with the help of FindBugs feedback. Again, the graph

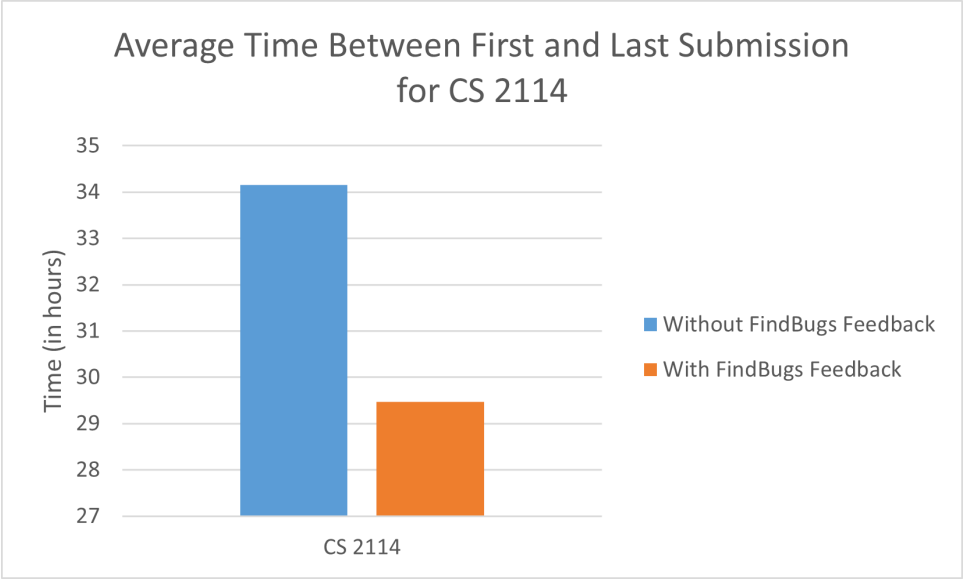

Figure 4.10: Time Difference Between Historical and Fall 2021 Data for CS 2114

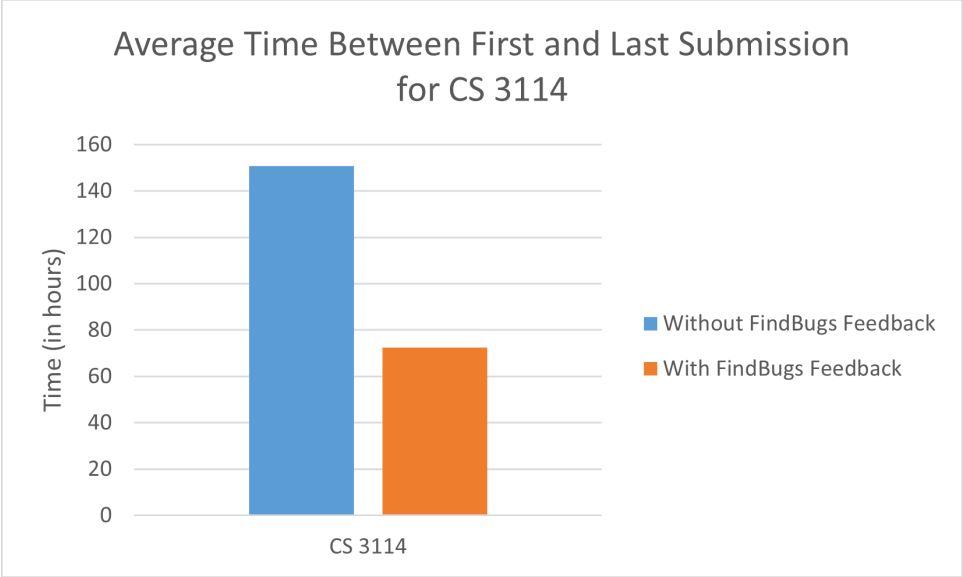

Figure 4.11: Time Difference Between Historical and Fall 2021 Data for CS 3114

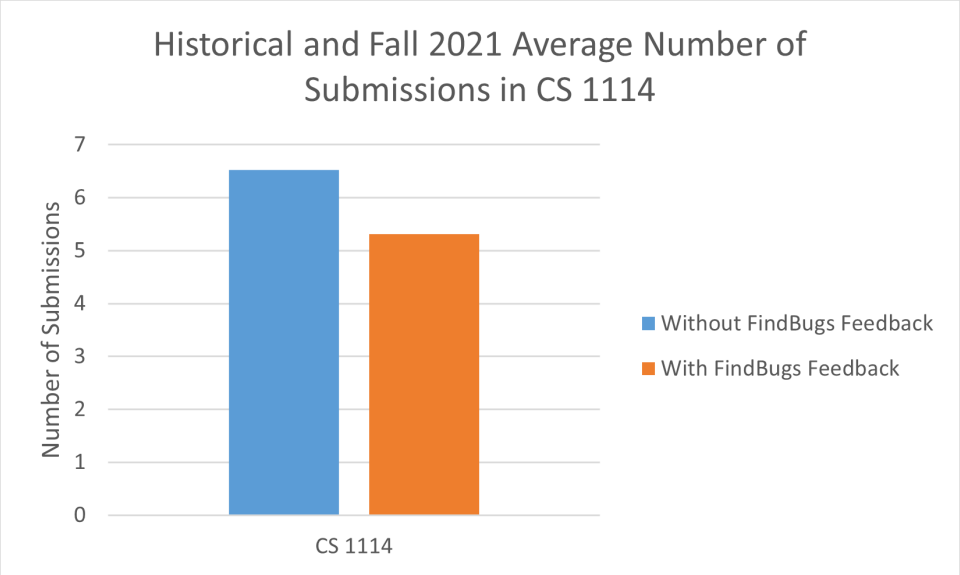

Figure 4.12: Historical and Fall 2021 Average Number of Submissions in CS 1114

indicates that there is a decrease in the number of submissions made to assignments in all three undergraduate courses when FindBugs was included in the Web-CAT feedback.

The replication study in Senger et al. [19] examined three possible indicators of struggling: longer times to complete an assignment, more submissions made, and lower final score. The historic data from Fall 2017 to Spring 2019 was used as our non-feedback group, and the Fall 2021 data was used as the feedback group. In addition to examining the impact of FindBugs feedback with respect to individual histories, we studied the overall effect of feedback using both sets of data.

In terms of final score, we found that feedback and course impact the final scores students receive on assignments. Students in CS 1114 received an average score of 92.5% on their final submissions without feedback, while students in CS 2114 averaged a 93.6%. Interestingly, scores seemed to decrease with the inclusion of feedback. CS 1114 students averaged 88.6% with feedback and CS 2114 students averaged 93.0%. The ANOVA test for the course showed an F ratio of 51.2 with a $p < 0.0001$ and the presence of feedback had an F ratio of 73.7

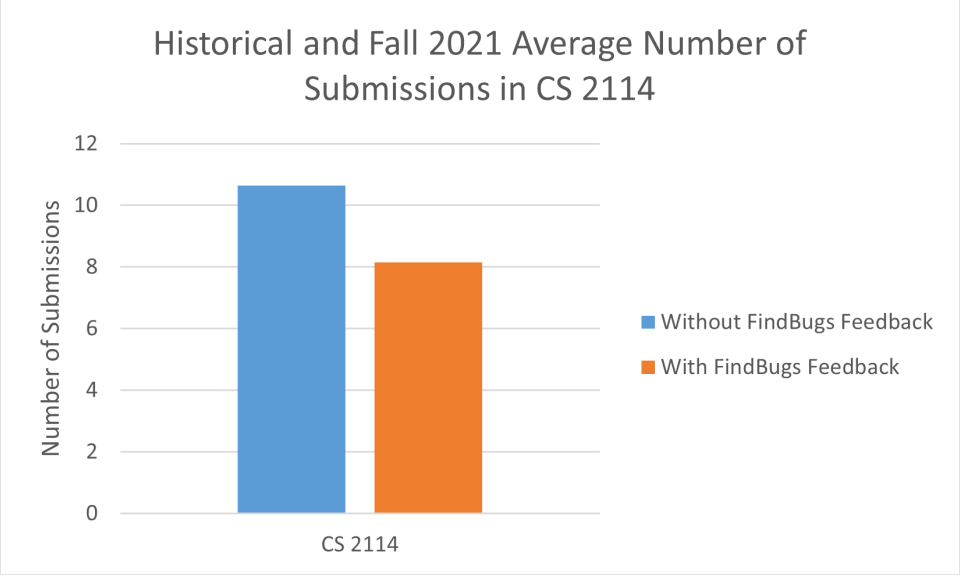

Figure 4.13: Historical and Fall 2021 Average Number of Submissions in CS 2114

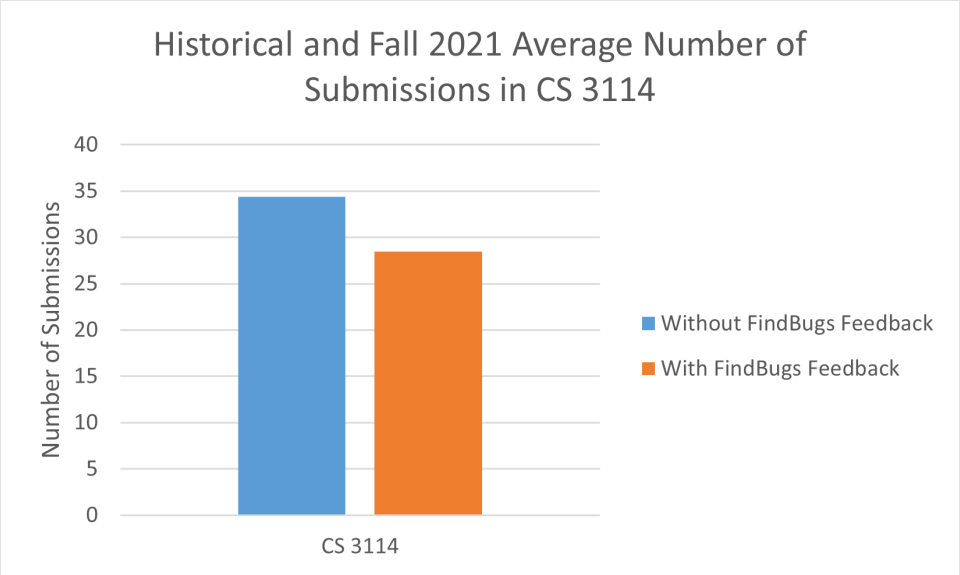

Figure 4.14: Historical and Fall 2021 Average Number of Submissions in CS 3114

with a $p < 0.0001$. There are a few possible reasons for the decrease in scores with feedback. The first is the impact that COVID could have had on students in recent semesters. The online coursework could have resulted in lower scores as more students seem to struggle with online learning. The difficulty of the coursework and assignments also fluctuates depending on the semester and the instructor. Finally, students could still not be seeing the feedback in either feedback style, or they could be choosing to ignore it in favor of working on the hints provided for problem coverage points.

Both course and presence of feedback also had an effect on the total elapsed time between the first and last submissions a student made to an assignment. The course had an F ratio of 223.4 with a $p < 0.0001$. The CS 1114 students had an average of about 22.9 hours between their first and last submissions while the CS 2114 students averaged about 33.3 hours between their first and last submissions. The presence of feedback in Web-CAT also had an impact on the amount of time taken to complete assignments. Feedback had an F ratio of 23.8 with a $p < 0.0001$.

Finally, the inclusion of FindBugs feedback and the course also had an influence on the number of submissions students required to complete assignments. The ANOVA test revealed an F ratio of 705.9 where $p < 0.0001$. As we would expect, students in CS 1114 made fewer submissions than students in CS 2114, likely due to the difficulty and length of assignments increasing from CS 1114 to CS 2114. The CS1 class had an average of 6.2 submissions to an assignment while the CS2 class had an average of 10.2.

On the historical assignments where students did not receive feedback, an average of 8.5 submissions were required while those where students did get feedback only required 6.4 on average. Adding the FindBugs feedback to Web-CAT was intended to help students identify problems more quickly instead of trying to guess-and-check their solution with the grader. These results seem to support the idea that students were able to recognize their problems

faster.

While none of these factors is indicative of student struggling on its own, we assumed that these three attributes could help determine if students were struggling and if the feedback was helping students resolve their problems more quickly, more accurately, and with fewer submissions.

Chapter 5

Experiences and Lessons Learned

This chapter discusses some personal experience from helping students as a teaching assistant with the FindBugs tool active and some thoughts on the Senger et al. [19] replication study now that we have been using this tool with students.

5.1 Experience from Helping Students with Coding Issues

In my experience helping students with coding issues in multiple CS2 courses, I have found that students will often approach a teaching assistant whenever their code compiles with an error or warning because they do not understand what it means. Because I have more practice, the errors are typically easy to resolve; however, there is definite frustration from students when they do understand the source of the error because they already knew it was wrong, but they just could not find it based on the error message alone.

It is important that students are shown the most important feedback first. Often, FindBugs errors are buried in many style and formatting errors, meaning students may not see them until they start to resolve those issues in their code. We are starting to use a different style of Web-CAT interface to direct students to the most important problems first. The CS 2114 and CS 3114 classes used the old feedback style of Web-CAT, which is partially shown in

Figure 1.2. CS 1114 has begun using a new user interface that directs students to the more problematic issues first and only highlights minor errors after the main functionality and testing problems have been resolved.

An example of the CS 2114 and CS 3114 style of feedback can be seen in Figure 5.1. This is a method in student code from CS 2114. The student has many CheckStyle problems, as well as salmon-colored lines that represent code uncovered by test cases. In the middle of all of this is a FindBugs warning. Unless a student was actively searching for FindBugs feedback in specific source files, it is unlikely they would see this message in a timely manner that would help them fix their code.

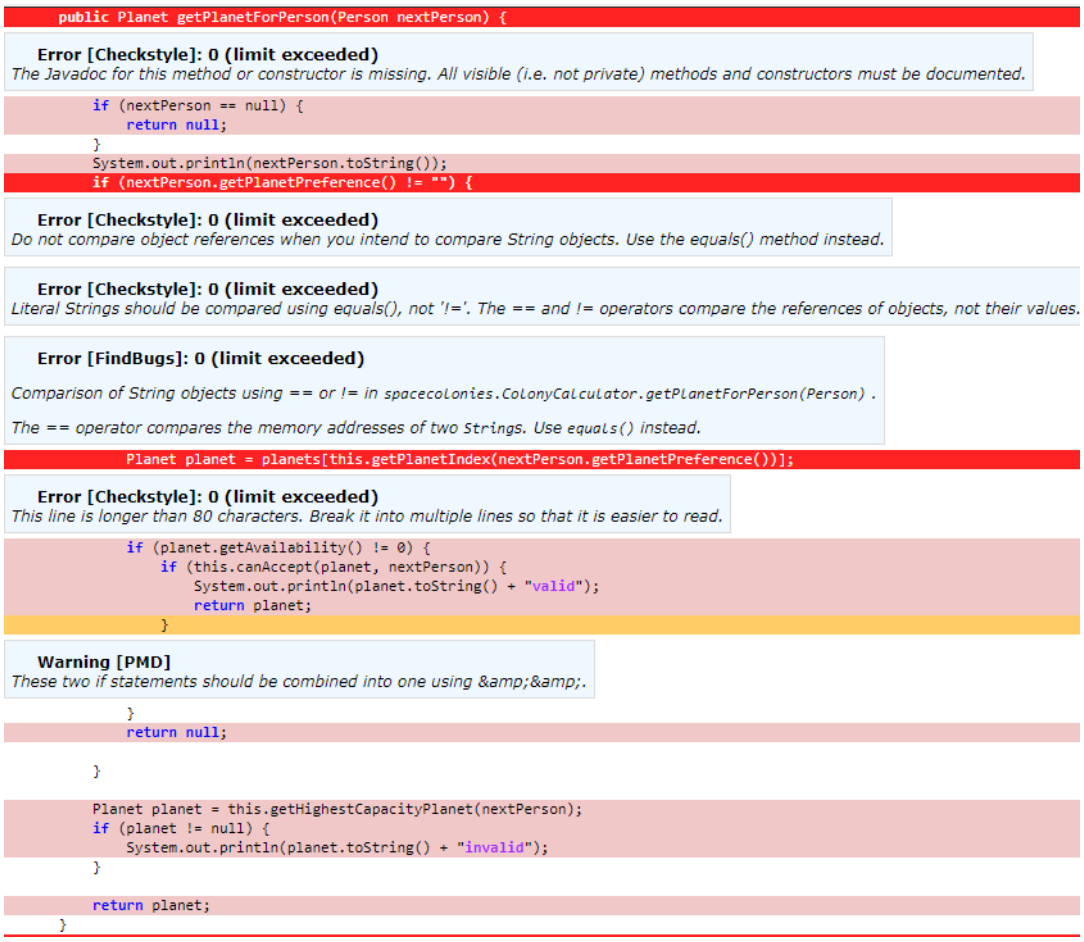

Figure 5.1: An example of feedback in old style

An example of the newer CS 1114 feedback can be seen in Figure 5.2. Note that this is the same student code run with a different feedback system. Attention is immediately brought to the FindBugs warnings because they indicate potential coding flaws. It is much easier to find the problems with the code when not having to sift through all of the stylistic feedback.

5.2 Thoughts on FindBugs Replication Study

Throughout Fall 2021, I have been working as a graduate teaching assistant (GTA) for one of our CS2 courses, CS 2114 Software Design and Data Structures. This has enabled me to see the types of warnings students frequently get from FindBugs, and it has given me a new perspective on the paper written in Summer 2021.

We found a small (although significant) difference in the correctness score students received depending on if they had a FindBugs warning in their final submission or not. However, I have seen some FindBugs warnings in submissions that are correctly marked as problematic, but the instructor reference tests do not necessarily test. For example, a common FindBugs error is students using == instead of .equals() to compare String objects. However, Java can recognize when the same String is created twice and, instead of making a new object, it will just add another reference to the already-created object. Unfortunately, this means that instructor tests that rely on String comparison may not detect the == versus .equals() problem. This indicates that any tests relying on the equality of objects should use an object type other than String to ensure that == represents identity and .equals() represents equality.

Other problems that represent bad practice may not cause errors in reference tests but are still bad habits for student programmers to incorporate into their code. One example of this is the MF_METHOD_MASKS_FIELD warning, which advises against using a local

variable in a method with the same name as a field either in this class or in a superclass. Technically, it is possible to write code that functions correctly with this problem. However, this is not something we want new programmers to learn. Further modification of the class or upkeep in the future may be hindered by the masking of a field by a local variable. This indicates another benefit of using FindBugs in the classroom to detect issues that instructor reference tests may not.

5.3 Examples from Student Code

5.3.1 Null Pointer Detection Based on While Loop Condition

A student in CS 2114 this semester was attempting to use a combined condition as their while loop condition. This condition was attempting to end the iteration through a linked list when the student had gotten to the node at position 'count'. The loop condition was as follows:

```
while (currentNode != null && count < index)
{. . .}
```

The student then attempted to dereference currentNode (likely to retrieve data) after the while loop finished to return the value in the node at 'index' in the list. The way their code was structured meant that the condition currentNode != null would never be met because they were selecting valid, random node from a list. However, FindBugs was able to see that if their loop broke on the condition currentNode != null, currentNode would then be null, and dereferencing it would cause a NullPointerException. The specific error code was NP_NULL_ON_SOME_PATH. While this would not be the case for the student code, it

is important that FindBugs recognizes this as a potential problem. This error could have also led the student to remove the unnecessary condition in their code, which makes the function of the code easier to understand for an outside observer.

5.3.2 Uninitialized Variable Detection

FindBugs can also detect problems in frontend classes. In a CS 2114 project, the students use a random number generator in their frontend to select items from a Bag data structure. This random number generator is declared as a field so it can be used throughout the frontend class.

This student tried to call a method on the random generator as follows:

```
int size = randomGenerator.nextInt(9) + 6;
```

However, the constructor in which the student tried to run this code had yet to initialize the random generator object. No feature of Web-CAT other than FindBugs detected this error, which would have caused a NullPointerException if the student ran their code. Our students sometimes do not realize that Web-CAT does not test the frontend of their code, so FindBugs can potentially detect problems the students may not have noticed before submitting otherwise.

5.3.3 Ignored Return Value

FindBugs can also detect when code calls a method that returns a value and then ignores that value. This frequently occurs in String methods because of the assumption that those methods modify the existing String (e.g. substring()). However, this can occur in other cases as well.

An example comes from the fourth project in CS 2114. Figure 5.3 shows part of a student's attempt at a compareTo() method that uses the compareTo() method of another class to compute its value. On line 116, the student attempts to return the comparison between the names of two Planet objects if their capacity and availability are equal. However, the student simply calls the compareTo() for the String class without storing or returning the information. Because this line will never change the execution of the code, the student is likely to get an incorrect result from this method. FindBugs recognizes that the return value is not used and gives the warning RV_RETURN_VALUE_IGNORED.

5.3.4 Local Variable Shadows Field

In CS 1114, students sometimes misunderstand how fields and local variables interact. Figure 5.4 shows a student's constructor. This class has fields that match the names of all the parameters in the constructor. To access a field correctly in this case, we must call 'this' (e.g. this.name = name) so the code knows to store into the field. However, this student's code sets the local variables equal to themselves and does not modify the field values. This could lead to unexpected behavior where none of the fields are set in methods called after the constructor. FindBugs was able to catch this problem and gave the warning DLS_DEAD_LOCAL_STORE_SHADOWS_FIELD.

5.4 Threats to Validity

There are several potential threats to validity for this work. The data for this study was collected over a period of four and a half years, with some of the semesters in the middle not recorded. Although we hope it averaged out, there is a possibility that the assignments or

instructors for the courses changed in difficulty throughout data collection. There have also been attempts to improve the Web-CAT display for CS 1114, and there has been work on the content of the CS 2114 course. There are also some students who submit assignments much more than their peers, which could skew the number of appearances of warnings (if students did not fix them between submissions) and the amount of submissions in which warnings appeared.

Most of the threats to validity from the Senger et al. study [19] are also still possible threats to validity, as the Fall 2021 results are a direct continuation of that study and the Edwards et al. study [9]. However, the work times between submissions from the Senger et al. study were not measured for the Fall 2021 data, which instead used time between the first student submission and the last student submission. The potential problems of FindBugs warnings appearing in multiple submissions due to students not attempting to fix them and the measure of struggling being assumed from three indicators are still relevant for this study.

Potential Coding Bugs

src/spacecolonies/PersonTest.java

The field 'person2' should be declared private. Only constants (static final fields) are allowed to be public. If this field is intended to represent a constant value, declare it to be static and final.

```
14.     Person person1;
15.     Person person2;
16.     public void setUp() {
```

src/spacecolonies/PersonTest.java

The field 'person1' should be declared private. Only constants (static final fields) are allowed to be public. If this field is intended to represent a constant value, declare it to be static and final.

```
13.
14.     Person person1;
15.     Person person2;
```

src/spacecolonies/ColonyCalculator.java

Literal Strings should be compared using equals(), not '!='. The == and != operators compare the references of objects, not their values. Using the equals() (or equalsIgnoreCase()) method will compare the values of the Strings instead.

```
63.         System.out.println(nextPerson.toString());
64.         if (nextPerson.getPlanetPreference() != "") {
65.             Planet planet = planets[this.getPlanetIndex(nextPerson.getPlanetPreference())];
```

src/spacecolonies/PlanetTest.java

This method does not contain any code. Should it be doing something? Or can it be removed? If you need to keep it, add a comment to the body explaining why it is empty.

```
43.
44.     }
45.
```

src/spacecolonies/ColonyCalculator.java

Do not compare object references when you intend to compare String objects. Use the equals() method instead.

```
63.         System.out.println(nextPerson.toString());
64.         if (nextPerson.getPlanetPreference() != "") {
65.             Planet planet = planets[this.getPlanetIndex(nextPerson.getPlanetPreference())];
```

Figure 5.2: An example of feedback in new style

```
112     @Override
113     public int compareTo(Planet other) {
114         if (this.getCapacity() == other.getCapacity()) {
115             if(this.getAvailability() == other.getAvailability()) {
116                 this.getName().compareTo(other.getName());
117             }
118             if(this.getAvailability() < other.getAvailability()) {
119                 return 1;
120             }
121             else {
122                 return -1;
123             }
124         }
```

Figure 5.3: Example of RV_RETURN_VALUE_IGNORED

```
33      public Unit(String name, int points, int quality, int defense)
34      {
35          super();
36          name = name;
37          points = points;
38          quality = 2;
39          defense = 2;
40          /*# Do any work to initialize your class here. */
41      }
```

Figure 5.4: Example of DLS_DEAD_LOCAL_STORE_SHADOWS_FIELD

Chapter 6

Conclusions

This work examined the presence of FindBugs warnings in student code and their impact on code correctness, time to completion, and number of attempts. We found that while some FindBugs warnings do not necessarily indicate a problem, they are still important for students to fix. We modified the feedback messages for FindBugs warnings that would be difficult for novice programmers to understand by simplifying the language and removing the need for excess background knowledge of computer science concepts. Including FindBugs has led students to write code for their assignments more effectively.

In this work, we implemented FindBugs into the Web-CAT automated grading system to provide more static-analysis tool feedback to student programmers. The findings of the research questions and future work in this area are discussed below.

6.1 Discussion of Research Questions

RQ1 addressed the FindBugs warnings that were relevant to students. From previous work in the Edwards et al. study [9] and the Senger et al. study [19], we started with several sets of warnings that were useful in some context to students. There was a difference between the short-form programming context from the Edwards et al. study and the long-form programming assignments examined by Senger et al. As this work is a continuation of Senger et al., that set of warnings was found to be relevant to students. Not all warnings

from the Fall 2017 to Spring 2019 time period appeared during Fall 2021, likely due to the different errors only occurring on a few submissions, possibly all for one student during that four semester period. We would not expect to see the uncommon errors every semester. A list of the most common FindBugs warnings from student code in Fall 2021 can be found in Figure 4.1.

In RQ3, we wanted to determine if FindBugs warnings indicated that students were struggling with assignments. We found that students without a FindBugs warning in their final submission received higher scores on assignments, took less time to complete their work, and made fewer submissions to assignments. These indicators show that FindBugs warnings are related to a student's success on an assignment, even though it is not possible for us to measure struggling directly.

Finally, RQ4 addressed how receiving FindBugs feedback influenced the number of submissions students made and how long they took to complete assignments. We found general downward trends across all classes in both number of submissions and elapsed time between first and last submissions when FindBugs warnings were added to the Web-CAT feedback. This could indicate that FindBugs is helping students complete their assignments more effectively.

6.2 Future Work

This work is just a piece of the ongoing work in the Virginia Tech Computer Science department to improve the quality of education. There are several directions forward from this work that could produce interesting results.

Firstly, it would be interesting to do some more work on analyzing which of the FindBugs

errors are the most important in terms of students having a correct submission. In the replication study [19], we had a set of four warnings that appeared only in submissions that failed to pass all instructor reference tests. This set included:

- RE_BAD_SYNTAX_FOR_REGULAR_EXPRESSION
- EQ_ALWAYS_TRUE
- VA_FORMAT_STRING_BAD_CONVERSION
- MF_CLASS_MASKS_FIELD

We assumed that if a warning only ever appeared in an incorrect submission, then that warning was a likely culprit for causing the failure of at least one reference test. However, these warnings represented a small percentage of the submissions with a warning in the strong set; one or more of those four warnings only occurred in 341 submissions out of the roughly 315,000 submissions with a warning from the strong set.

Some of the warnings that make up the other 55 errors of the strong set are likely to cause problems for passing reference tests as well. For example, ES_COMPARING_STRINGS_WITH_EQ (which uses == to compare String objects instead of .equals()) seems likely to cause a reference test to fail. Only 10% of submissions with this error passed all the reference tests, and 17,780 submissions contained this error. It seems as if this warning is very important because most submissions did not pass if it was included. However, this could be a flawed assumption; if this warning appeared in conjunction with another warning that was actually the cause of the error, then this message is not as important. Work needs to be done to determine how interrelated the FindBugs warnings are in submissions to figure out which warnings are actually the most problematic, not just the most frequent.

It is also important to look at the FindBugs warnings we did not include in our analysis.

We kept FindBugs from flagging some errors in the Bad Practice, Correctness, and Style categories because they caused problems for our instructors or because they are not what is taught at this university. Future research could examine if any of these excluded warnings represent gaps in our teaching at the undergraduate and graduate levels as we attempt to prepare students for the workforce. What we may see as a trivial error to exclude because it is not relevant to us (like requiring a hashCode() method with every equals() method) may represent an important coding flaw in a software company.

If we did decide to include more of the FindBugs messages in the future, we would want to modify those to also be understandable. Some of the warnings that could potentially get added later include those that caused student submissions to fail but did not occur frequently enough for us to include. We required that a FindBugs message appear in over 500 incorrect submissions or in 25 submissions where none of the submissions were correct before we recognized it in the strong set. However, some errors could just be harder to create or only appear in a higher level class like CS 3114, which would mean they occur in fewer submissions.

Additionally, some of the messages for which we have already written modified feedback may benefit from more examples or external links to pages that show the problem in action in another piece of code. It could also be useful to review student submissions with specific errors to determine in which contexts students are creating those errors. Then we could modify our feedback to more directly address the problems that a student is likely having.

As shown in Figures 5.1 and 5.2, the Web-CAT feedback varies depending on the class. Updating the score interface for students in CS 2114 and CS 3114 could make it easier for them to see and fix FindBugs warnings.

Bibliography

[1] Bruce M. Adcock, Paolo Bucci, Wayne D. Heym, Joseph E. Hollingsworth, Timothy J. Long, and Bruce W. Weide. 2007. Which pointer errors do students make? *Proceedings of the 38th SIGCSE technical symposium on Computer science education* (2007).

[2] ApacheAnt 2020. ApacheAnt website. https://ant.apache.org/.

[3] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE-WGR '19)*. Association for Computing Machinery, New York, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[4] Matthew Butler and Michael Morgan. 2007. Learning challenges faced by novice programming students studying high level and low feedback concepts. (01 2007).

[5] CheckStyle 2018. CheckStyle website. http://checkstyle.sourceforge.net/.

[6] Tomche Delev and Dejan Gjorgjevikj. 2017. Static analysis of source code written by novice programmers. 825–830. https://doi.org/10.1109/EDUCON.2017.7942942

[7] Anna Eckerdal and Michael Thuné. 2005. Novice Java programmers' conceptions of "object" and "class", and variation theory. *ACM SIGCSE Bulletin* 37, 89–93. https://doi.org/10.1145/1067445.1067473

[8] Stephen Edwards, Nischel Kandru, and Mukund Rajagopal. 2017. Investigating Static Analysis Errors in Student Java Programs. 65–73. https://doi.org/10.1145/3105726.3106182

[9] S. Edwards, Jaime Spacco, and David Hovemeyer. 2019. Can Industrial-Strength Static Analysis Be Used to Help Students Who Are Struggling to Complete Programming Activities?. In *Proceedings of the 52nd Annual Hawaii International Conference on System Sciences*. Computer Society Press. https://doi.org/10125/60221

[10] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. Common Logic Errors Made by Novice Programmers. In *Proceedings of the 20th Australasian Computing Education Conference* (Brisbane, Queensland, Australia) *(ACE '18)*. Association for Computing Machinery, New York, NY, USA, 83–89. https://doi.org/10.1145/3160489.3160493

[11] José Luis Fernandez Fernandez Aleman, Dominic Palmer-Brown, and Chrisina Jayne. 2011. Effects of Response-Driven Feedback in Computer Science Learning. *IEEE Transactions on Education* 54, 3 (2011), 501–508. https://doi.org/10.1109/TE.2010.2087761

[12] FindBugs 2015. FindBugs website. http://findbugs.sourceforge.net/bugDescriptions.html.

[13] J. S. Foster, C. B. Almazan, and N. Rutar. 2004. A Comparison of Bug Finding Tools for Java. In *15th International Symposium on Software Reliability Engineering*. IEEE Computer Society, Los Alamitos, CA, USA, 245–256. https://doi.org/10.1109/ISSRE.2004.1

[14] David Hovemeyer and William Pugh. 2004. Finding Bugs is Easy. *SIGPLAN Not.* 39, 12 (Dec. 2004), 92–106. https://doi.org/10.1145/1052883.1052895

[15] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. 2011. Measuring the Effectiveness of Error Messages Designed for Novice Programmers. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 499–504. https://doi.org/10.1145/1953163.1953308

[16] Marie-Hélène Nienaltowski, Michela Pedroni, and Bertrand Meyer. 2008. Compiler Error Messages: What Can Help Novices? *SIGCSE Bull.* 40, 1 (March 2008), 168–172. https://doi.org/10.1145/1352322.1352192

[17] PMD 2018. PMD website. https://pmd.github.io/.

[18] Yizhou Qian and James D. Lehman. 2019. Using Targeted Feedback to Address Common Student Misconceptions in Introductory Programming: A Data-Driven Approach. *SAGE Open* 9, 4 (2019), 2158244019885136. https://doi.org/10.1177/2158244019885136 arXiv:https://doi.org/10.1177/2158244019885136

[19] Allyson Senger, Stephen Edwards, and Margaret Ellis. to appear. Helping Student Programmers Through Industrial-Strength Static Analysis: A Replication Study. *SIGCSE 2022: ACM Technical Symposium on Computer Science Education* (to appear).

[20] Michael Striewe and Michael Goedicke. 2014. A Review of Static Analysis Approaches for Programming Exercises. In *Computer Assisted Assessment. Research into E-Assessment*, Marco Kalz and Eric Ras (Eds.). Springer International Publishing, Cham, 100–113.

[21] Web-CAT 2019. Web-CAT website. https://web-cat.org/.

Appendices

Appendix A

FindBugs Messages

A.1 FindBugs Messages Used in Web-CAT

Note that the descriptions for all FindBugs messages can be found on the website [12]. The FindBugs messages below are pulled directly from that website.

RE_BAD_SYNTAX_FOR_REGULAR_EXPRESSION

> FindBugs message: "The code here uses a regular expression that is invalid according to the syntax for regular expressions. This statement will throw a PatternSyntaxException when executed."

> Web-CAT uses original message

EQ_ALWAYS_TRUE

> FindBugs message: "This class defines an equals method that always returns true. This is imaginative, but not very smart. Plus, it means that the equals method is not symmetric."

> Web-CAT uses original message

VA_FORMAT_STRING_BAD_CONVERSION

> FindBugs message: "One of the arguments is uncompatible with the corresponding format string specifier. As a result, this will generate a runtime exception when executed. For example, String.format("%d", "1") will generate an exception, since the String "1" is incompatible with the format specifier %d."

> Web-CAT uses original message

MF_CLASS_MASKS_FIELD

> FindBugs message: "This class defines a field with the same name as a visible instance field in a superclass. This is confusing, and may indicate an error if methods update or access one of the fields when they wanted the other."

> Rewritten message: The field name in this class matches a visible field of the same name in the superclass. Change one of them so you don't get confused.

DLS_DEAD_LOCAL_STORE

> FindBugs message: "This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used."

> Rewritten message: This local variable has a value, but the value was never used. Make sure the value is used where intended, or remove it if it is not needed.

ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD

> FindBugs message: "This instance method writes to a static field. This is tricky to get correct if multiple instances are being manipulated, and generally bad practice."

> Rewritten message: An object typically represents an instance of a class. For example, a Car can have more than one Tire. If you write to a static field from one of the Tires, it will update the value for all of them. When you have multiples of an object, avoid updating static fields.

RV_RETURN_VALUE_IGNORED_NO_SIDE_EFFECT

> FindBugs message: "This code calls a method and ignores the return value. However our analysis shows that the method (including its implementations in subclasses if any) does not produce any effect other than return value. Thus this call can be removed."

> Rewritten message: Immutable objects like Strings are not updated when a method is called on them. If you want the updated value, you will have to assign the result of the method to a variable.

EQ_COMPARETO_USE_OBJECT_EQUALS

> FindBugs message: "This class defines a compareTo(...) method but inherits its equals() method from java.lang.Object. Generally, the value of compareTo should return zero if and only if equals returns true. If this is violated, weird and unpredictable failures will occur in classes such as PriorityQueue. In Java 5 the PriorityQueue.remove method uses the compareTo method, while in Java 6 it uses the equals method."

> Rewritten message: When you define a compareTo() method, you should also override the equals() method from Object. compareTo() should return 0 if equals returns true.

ES_COMPARING_STRINGS_WITH_EQ

> FindBugs message: "This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by two different String objects. Consider using the equals(Object) method instead."

> Rewritten message: The == compares the memory addresses of two Strings. Consider using equals() instead.

EC_BAD_ARRAY_COMPARE

> FindBugs message: "This method invokes the .equals(Object o) method on an array. Since arrays do not override the equals method of Object, calling equals on an array is the same as comparing their addresses. To compare the contents of the arrays, use java.util.Arrays.equals(Object[], Object[]). To compare the addresses of the arrays, it would be less confusing to explicitly check pointer equality using ==."

> Rewritten message: The equals() method compares the memory addresses of arrays in-

stead of the contents of the arrays. Consider using Arrays.equals() or write your own loop comparison.

UWF_UNWRITTEN_FIELD

> FindBugs message: "This field is never written. All reads of it will return the default value. Check for errors (should it have been initialized?), or remove it if it is useless."

> Rewritten message: This field never gets assigned a value. Check if it should be initialized or if it can be removed.

RR_NOT_CHECKED

> FindBugs message: "This method ignores the return value of one of the variants of java.io.InputStream.read() which can return multiple bytes. If the return value is not checked, the caller will not be able to correctly handle the case where fewer bytes were read than the caller requested. This is a particularly insidious kind of bug, because in many programs, reads from input streams usually do read the full amount of data requested, causing the program to fail only sporadically."

> Rewritten message: A call to read() from an InputStream is not checked. Be sure to check the return value to avoid making assumptions about what it contains.

UWF_NULL_FIELD

> FindBugs message: "All writes to this field are of the constant value null, and thus all reads of the field will return null. Check for errors, or remove it if it is useless."

> Rewritten message: Every assignment to this field is null. Make sure it is assigned a value before you use it or remove it.

NP_NONNULL_PARAM_VIOLATION

> FindBugs message: "This method passes a null value as the parameter of a method which must be non-null. Either this parameter has been explicitly marked as @Nonnull, or analysis

has determined that this parameter is always dereferenced."

> Rewritten message: The parameter in this method must not be null. It is always dereferenced.

NP_UNWRITTEN_FIELD

> FindBugs message: "The program is dereferencing a field that does not seem to ever have a non-null value written to it. Unless the field is initialized via some mechanism not seen by the analysis, dereferencing this value will generate a null pointer exception."

> Rewritten message: This field is never assigned a value other than null, so dereferencing it will cause a NullPointerException.

UC_USELESS_OBJECT

> FindBugs message: "Our analysis shows that this object is useless. It's created and modified, but its value never go outside of the method or produce any side-effect. Either there is a mistake and object was intended to be used or it can be removed."

> Rewritten message: This object is created using the new operator and possibly modified, but it does not do anything in this method or in a different method. Make sure this object is used; otherwise, it can be removed.

NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT

> FindBugs message: "This implementation of equals(Object) violates the contract defined by java.lang.Object.equals() because it does not check for null being passed as the argument. All equals() methods should return false if passed a null value."

> Rewritten message: Include a check for if the parameter to equals() is null.

NP_NULL_ON_SOME_PATH_EXCEPTION

> FindBugs message: "A reference value which is null on some exception control path is dereferenced here. This may lead to a NullPointerException when the code is executed.

Note that because FindBugs currently does not prune infeasible exception paths, this may be a false warning.

Also note that FindBugs considers the default case of a switch statement to be an exception path, since the default case is often infeasible."

> Rewritten message: When handling an exception, a value that will be dereferenced is null. This may cause a NullPointerException.

ES_COMPARING_PARAMETER_STRING_WITH_EQ

> FindBugs message: "This code compares a java.lang.String parameter for reference equality using the == or != operators. Requiring callers to pass only String constants or interned strings to a method is unnecessarily fragile, and rarely leads to measurable performance gains. Consider using the equals(Object) method instead."

> Rewritten message: The == compares the memory addresses of two Strings. Consider using equals() instead. One of the String being compared is a parameter.

RV_ABSOLUTE_VALUE_OF_RANDOM_INT

> FindBugs message: "This code generates a random signed integer and then computes the absolute value of that random integer. If the number returned by the random number generator is Integer.MIN_VALUE, then the result will be negative as well (since Math.abs(Integer.MIN_VALUE) == Integer.MIN_VALUE). (Same problem arised for long values as well)."

> Web-CAT uses original message

SF_SWITCH_NO_DEFAULT

> FindBugs message: "This method contains a switch statement where default case is missing. Usually you need to provide a default case."

> Web-CAT uses original message

BC_EQUALS_METHOD_SHOULD_WORK_FOR_ALL_OBJECTS

> FindBugs message: "The equals(Object o) method shouldn't make any assumptions about the type of o. It should simply return false if o is not the same type as this."

> Rewritten message: To override equals(), the parameter must be of type Object. If the class of the Object and this do not match, equals() will return false.

EQ_GETCLASS_AND_CLASS_CONSTANT

> FindBugs message: "This class has an equals method that will be broken if it is inherited by subclasses. It compares a class literal with the class of the argument (e.g., in class Foo it might check if Foo.class == o.getClass()). It is better to check if this.getClass() == o.getClass()."

> Web-CAT uses original message

EC_UNRELATED_TYPES

> FindBugs message: "This method calls equals(Object) on two references of different class types and analysis suggests they will be to objects of different classes at runtime. Further, examination of the equals methods that would be invoked suggest that either this call will always return false, or else the equals method is not be symmetric (which is a property required by the contract for equals in class Object)."

> Rewritten message: This equals() comparison is comparing Objects of two different class types. Double check the arguments to equals().

NP_NULL_ON_SOME_PATH

> FindBugs message: "There is a branch of statement that, if executed, guarantees that a null value will be dereferenced, which would generate a NullPointerException when the code is executed. Of course, the problem might be that the branch or statement is infeasible and that the null pointer exception can't ever be executed; deciding that is beyond the ability of FindBugs."

> Web-CAT uses original message

RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE

> FindBugs message: "A value is checked here to see whether it is null, but this value can't be null because it was previously dereferenced and if it were null a null pointer exception would have occurred at the earlier dereference. Essentially, this code and the previous dereference disagree as to whether this value is allowed to be null. Either the check is redundant or the previous dereference is erroneous."

> Rewritten message: This variable has already been dereferenced somewhere in the code. If the code got to this point, it means the value was non-null at the time and cannot be null here unless it was modified. Make sure the earlier dereference was appropriate or remove this null check.

UC_USELESS_CONDITION

> FindBugs message: "This condition always produces the same result as the value of the involved variable was narrowed before. Probably something else was meant or condition can be removed."

> Rewritten message: This condition always gives the same result. The value of the variable may have been narrowed to always match the condition in a previous statement or you may be using the = operator instead of == by mistake. Update the condition or remove it, as it is not necessary.

UR_UNINIT_READ

> FindBugs message: "This constructor reads a field which has not yet been assigned a value. This is often caused when the programmer mistakenly uses the field instead of one of the constructor's parameters."

> Rewritten message: The constructor is trying to use the value of a field, but the field has not been assigned yet. Initialize the field before you use it in the constructor.

ICAST_IDIV_CAST_TO_DOUBLE

> FindBugs message: "This code casts the result of an integral division (e.g., int or long division) operation to double or float. Doing division on integers truncates the result to the integer value closest to zero. The fact that the result was cast to double suggests that this precision should have been retained. What was probably meant was to cast one or both of the operands to double before performing the division. Here is an example:

int x = 2; int y = 5; // Wrong: yields result 0.0 double value1 = x / y;

// Right: yields result 0.4 double value2 = x / (double) y;"

> Rewritten message: When using integers in a division statement, Java will automatically use integer division. This results in the truncated value where the decimal part is removed. Instead of using, for example, 2, use 2.0 on the bottom of the division.

BC_VACUOUS_INSTANCEOF

> FindBugs message: "This instanceof test will always return true (unless the value being tested is null). Although this is safe, make sure it isn't an indication of some misunderstanding or some other logic error. If you really want to test the value for being null, perhaps it would be clearer to do better to do a null test rather than an instanceof test."

> Web-CAT uses original message

RV_RETURN_VALUE_IGNORED_BAD_PRACTICE

> FindBugs message: "This method returns a value that is not checked. The return value should be checked since it can indicate an unusual or unexpected function execution. For example, the File.delete() method returns false if the file could not be successfully deleted (rather than throwing an Exception). If you don't check the result, you won't notice if the method invocation signals unexpected behavior by returning an atypical return value."

> Web-CAT uses original message

FE_FLOATING_POINT_EQUALITY

> FindBugs message: "This operation compares two floating point values for equality. Because floating point calculations may involve rounding, calculated float and double values may not be accurate. For values that must be precise, such as monetary values, consider using a fixed-precision type such as BigDecimal. For values that need not be precise, consider comparing for equality within some range, for example: if (Math.abs(x - y) < .0000001). See the Java Language Specification, section 4.2.4."
> Rewritten message: Floating point and double calculations can involve rounding which makes equality hard to check. In test classes, use assertEquals(value1, value2, precision) to test equality for doubles and floats. In normal Java classes, check a range: e.g. if (Math.abs(x-y) <.000001).

ICAST_INTEGER_MULTIPLY_CAST_TO_LONG

> FindBugs message: "This code performs integer multiply and then converts the result to a long, as in:

long convertDaysToMilliseconds(int days) return 1000*3600*24*days; If the multiplication is done using long arithmetic, you can avoid the possibility that the result will overflow. For example, you could fix the above code to:

```
long convertDaysToMilliseconds(int days) { return 1000L*3600*24*days; }
```

or

```
static final long MILLISECONDS\_PER\_DAY = 24L*3600*1000;
```

> Web-CAT uses original message

EQ_SELF_USE_OBJECT

> FindBugs message: "This class defines a covariant version of the equals() method, but

inherits the normal equals(Object) method defined in the base java.lang.Object class. The class should probably define a boolean equals(Object) method."

> Rewritten message: The equals() method in this class is overloading, not overriding the equals() from Object. The method header should be boolean equals(Object obj).

DMI_INVOKING_TOSTRING_ON_ARRAY

> FindBugs message: "The code invokes toString on an array, which will generate a fairly useless result such as [C@16f0472. Consider using Arrays.toString to convert the array into a readable String that gives the contents of the array."

> Rewritten message: Using toString() on an array prints out its memory location, not its contents. Try using Arrays.toString().

UC_USELESS_VOID_METHOD

> FindBugs message: "Our analysis shows that this non-empty void method does not actually perform any useful work. Please check it: probably there's a mistake in its code or its body can be fully removed."

> Web-CAT uses original message

DB_DUPLICATE_BRANCHES

> FindBugs message: "This method uses the same code to implement two branches of a conditional branch. Check to ensure that this isn't a coding mistake."

> Web-CAT uses original message

NP_NULL_PARAM_DEREF_NONVIRTUAL

> FindBugs message: "A possibly-null value is passed to a non-null method parameter. Either the parameter is annotated as a parameter that should always be non-null, or analysis has shown that it will always be dereferenced."

> Web-CAT uses original message

OS_OPEN_STREAM

> FindBugs message: "The method creates an IO stream object, does not assign it to any fields, pass it to other methods that might close it, or return it, and does not appear to close the stream on all paths out of the method. This may result in a file descriptor leak. It is generally a good idea to use a finally block to ensure that streams are closed."

> Rewritten message: When you open an IO stream object, be sure to close it when you are done.

IP_PARAMETER_IS_DEAD_BUT_OVERWRITTEN

> FindBugs message: "The initial value of this parameter is ignored, and the parameter is overwritten here. This often indicates a mistaken belief that the write to the parameter will be conveyed back to the caller."

> Rewritten message: A parameter of this method was overwritten without using its value in the method. Create a new variable instead of assigning over the parameter value.

SA_FIELD_DOUBLE_ASSIGNMENT

> FindBugs message: "This method contains a double assignment of a field; e.g.

int x,y; public void foo() x = x = 17; Assigning to a field twice is useless, and may indicate a logic error or typo."

> Web-CAT uses original message

DLS_DEAD_LOCAL_STORE_SHADOWS_FIELD

> FindBugs message: "This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used. There is a field with the same name as the local variable. Did you mean to assign to that variable instead?"

> Web-CAT uses original message

BC_UNCONFIRMED_CAST

> FindBugs message: "This cast is unchecked, and not all instances of the type casted from can be cast to the type it is being cast to. Check that your program logic ensures that this cast will not fail."

> Web-CAT uses original message

EQ_DOESNT_OVERRIDE_EQUALS

> FindBugs message: "This class extends a class that defines an equals method and adds fields, but doesn't define an equals method itself. Thus, equality on instances of this class will ignore the identity of the subclass and the added fields. Be sure this is what is intended, and that you don't need to override the equals method. Even if you don't need to override the equals method, consider overriding it anyway to document the fact that the equals method for the subclass just return the result of invoking super.equals(o)."

> Web-CAT uses original message

NP_LOAD_OF_KNOWN_NULL_VALUE

> FindBugs message: "The variable referenced at this point is known to be null due to an earlier check against null. Although this is valid, it might be a mistake (perhaps you intended to refer to a different variable, or perhaps the earlier check to see if the variable is null should have been a check to see if it was non-null)."

> Web-CAT uses original message

RV_RETURN_VALUE_IGNORED

> FindBugs message: "The return value of this method should be checked. One common cause of this warning is to invoke a method on an immutable object, thinking that it updates the object. For example, in the following code fragment,

String dateString = getHeaderField(name); dateString.trim(); the programmer seems to be

thinking that the trim() method will update the String referenced by dateString. But since Strings are immutable, the trim() function returns a new String value, which is being ignored here. The code should be corrected to:

String dateString = getHeaderField(name); dateString = dateString.trim();"

> Rewritten message: Immutable objects like Strings are not updated when a method is called on them. If you want the updated value, you will have to assign the result of the method to a variable.

URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD

> FindBugs message: "This field is never read. The field is public or protected, so perhaps it is intended to be used with classes not seen as part of the analysis. If not, consider removing it from the class."

> Rewritten message: A field is assigned public or protected when another class needs access to it. This field has not been used outside of the class. Consider removing it or switching it to private if it is only used within this class.

SF_SWITCH_FALLTHROUGH

> FindBugs message: "This method contains a switch statement where one case branch will fall through to the next case. Usually you need to end this case with a break or return."

> Rewritten message: End each case in a switch statement with a break or a return to prevent multiple pieces of it from being executed.

UUF_UNUSED_PUBLIC_OR_PROTECTED_FIELD

> FindBugs message: "This field is never used. The field is public or protected, so perhaps it is intended to be used with classes not seen as part of the analysis. If not, consider removing it from the class."

> Rewritten message: This field is visible to other classes, but it is not used. Remove it if

it is unnecessary.

RpC_REPEATED_CONDITIONAL_TEST

> FindBugs message: "The code contains a conditional test is performed twice, one right after the other (e.g., x == 0 || x == 0). Perhaps the second occurrence is intended to be something else (e.g., x == 0 || y == 0)."

> Rewritten message: The same conditional statement is used twice. Remove the repeated condition or update it to reflect actual intent.

CO_SELF_NO_OBJECT

> FindBugs message: "This class defines a covariant version of compareTo(). To correctly override the compareTo() method in the Comparable interface, the parameter of compareTo() must have type java.lang.Object."

> Rewritten message: To correctly override compareTo(), use the header: int compareTo(Object obj).

NS_DANGERGOUS_NON_SHORT_CIRCUIT

> FindBugs message: "This code seems to be using non-short-circuit logic (e.g., & or |) rather than short-circuit logic (&& or ||). In addition, it seem possible that, depending on the value of the left hand side, you might not want to evaluate the right hand side (because it would have side effects, could cause an exception or could be expensive.

Non-short-circuit logic causes both sides of the expression to be evaluated even when the result can be inferred from knowing the left-hand side. This can be less efficient and can result in errors if the left-hand side guards cases when evaluating the right-hand side can generate an error."

> Rewritten message: Using only one | or & means that conditional statements will not short circuit. Even if the first condition fails, the second will be evaluated. The second

condition may cause an exception or other side effects.

DMI_HARDCODED_ABSOLUTE_FILENAME

> FindBugs message: "This code constructs a File object using a hard coded to an absolute pathname (e.g., new File("/home/dannyc/workspace/j2ee/src/share/com/sun/enterprise/deployment");"

> Rewritten message: This code constructs a File object using a hard coded to an absolute pathname (e.g., new File("/home/dannyc/workspace/j2ee/src/share/com/sun/enterprise/deployment"); Use a relative path to allow other computers to run your code.

NP_ALWAYS_NULL

> FindBugs message: "A null pointer is dereferenced here. This will lead to a NullPointerException when the code is executed."

> Rewritten message: A null pointer is dereferenced here. This will lead to a NullPointerException when the code is executed. Check that all Objects are initialized correctly.

NP_NULL_PARAM_DEREF

> FindBugs message: "This method call passes a null value for a non-null method parameter. Either the parameter is annotated as a parameter that should always be non-null, or analysis has shown that it will always be dereferenced."

> Web-CAT uses original message

NP_TOSTRING_COULD_RETURN_NULL

> FindBugs message: "This toString method seems to return null in some circumstances. A liberal reading of the spec could be interpreted as allowing this, but it is probably a bad idea and could cause other code to break. Return the empty string or some other appropriate string rather than null."

> Rewritten message: toString() should not return null. Try using an empty String ("") instead.

EC_NULL_ARG

> FindBugs message: "This method calls equals(Object), passing a null value as the argument. According to the contract of the equals() method, this call should always return false."
> Rewritten message: Comparing any Object to null will always return false. This comparison is not necessary. Make sure you have initialized all your Objects.

IL_INFINITE_RECURSIVE_LOOP

> FindBugs message: "This method unconditionally invokes itself. This would seem to indicate an infinite recursive loop that will result in a stack overflow."
> Rewritten message: This recursive loop does not terminate. Check that a base case is included and that the recursion will eventually reach that base case.

SA_FIELD_SELF_ASSIGNMENT

> FindBugs message: "This method contains a self assignment of a field; e.g.

```
int x;
public void foo() {
  x = x;
}
```

Such assignments are useless, and may indicate a logic error or typo."
> Web-CAT uses original message

A.2 Excluded Messages

Excluded in Initial Short-Form Programming Exercises Study:

- IM_BAD_CHECK_FOR_ODD

- DMI_USELESS_SUBSTRING
- RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE
- NM_METHOD_NAMING_CONVENTION
- HE_EQUALS_USE_HASHCODE

Excluded Correctness Warnings:

- IJU_SETUP_NO_SUPER
- IJU_TEARDOWN_NO_SUPER
- ME_ENUM_FIELD_SETTER
- RV_ABSOLUTE_VALUE_OF_HASHCODE
- NP_ALWAYS_NULL_EXCEPTION
- HE_SIGNATURE_DECLARES_HASHING_OF_UNHASHABLE_CLASS
- UWF_NULL_FIELD

Excluded Bad Practice Warnings:

- VA_FORMAT_STRING_USES_NEWLINE
- DM_EXIT
- NM_CLASS_NAMING_CONVENTION
- HE_EQUALS_NO_HASHCODE
- NM_CLASS_NOT_EXCEPTION

- HE_INHERITS_EQUALS_USE_HASHCODE
- CNT_ROUGH_CONSTANT_VALUE
- IT_NO_SUCH_ELEMENT
- CN_IMPLEMENTS_CLONE_BUT_NOT_CLONEABLE
- DMI_RANDOM_USED_ONLY_ONCE
- NM_FIELD_NAMING_CONVENTION
- DE_MIGHT_IGNORE
- RV_NEGATING_RESULT_OF_COMPARETO
- RC_REF_COMPARISON_BAD_PRACTICE_BOOLEAN
- CN_IDIOM_NO_SUPER_CALL
- RR_NOT_CHECKED
- EQ_COMPARETO_USE_OBJECT_EQUALS

Excluded Dodgy Code (Style) Warnings:

- NP_NULL_ON_SOME_PATH_MIGHT_BE_INFEASIBLE
- ICAST_QUESTIONABLE_UNSIGNED_RIGHT_SHIFT
- EQ_UNUSUAL
- IA_AMBIGUOUS_INVOCATION_OF_INHERITED_OR_OUTER_METHOD
- EQ_DOESNT_OVERRIDE_EQUALS

Appendix B

HRPP Approval

Division of Scholarly Integrity and Research Compliance
Institutional Review Board
North End Center, Suite 4120 (MC 0497)
300 Turner Street NW
Blacksburg, Virginia 24061
540/231-3732
irb@vt.edu
http://www.research.vt.edu/sirc/hrpp

MEMORANDUM

DATE: November 17, 2021

TO: Cliff Shaffer, Alex Hicks, Rifat Sabbir Mansur, Mostafa Kamel Osman Mohammed, Allyson Senger, Eunoh Cho, Molly Rebecca Domino, Mohammed Fawzi Seddik Farghally, Stephen H Edwards, Zhiyi Li, et. al.

FROM: Virginia Tech Institutional Review Board (FWA00000572)

PROTOCOL TITLE: Community-Building and Infrastructure Design for Data-Intensive Research in Computer Science Education

IRB NUMBER: **17-1095**

Effective November 16, 2021, the Virginia Tech Institution Review Board (IRB) approved the Amendment request for the above-mentioned research protocol.

This approval provides permission to begin the human subject activities outlined in the IRB-approved protocol and supporting documents.

Plans to deviate from the approved protocol and/or supporting documents must be submitted to the IRB as an amendment request and approved by the IRB prior to the implementation of any changes, regardless of how minor, except where necessary to eliminate apparent immediate hazards to the subjects. Report within 5 business days to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.

All investigators (listed above) are required to comply with the researcher requirements outlined at:

https://secure.research.vt.edu/external/irb/responsibilities.htm

(Please review responsibilities before beginning your research.)

PROTOCOL INFORMATION:

Approved As:	**Expedited, under 45 CFR 46.110 category(ies) 5,7**
Protocol Approval Date:	**February 19, 2021**
Protocol Expiration Date:	**February 18, 2022**
Continuing Review Due Date*:	**January 28, 2022**

*Date a Continuing Review application is due to the IRB office if human subject activities covered under this protocol, including data analysis, are to continue beyond the Protocol Expiration Date.

ASSOCIATED FUNDING:

The table on the following page indicates whether grant proposals are related to this protocol, and which of the listed proposals, if any, have been compared to this protocol, if required.

SPECIAL INSTRUCTIONS:

This amendment, submitted Ocber 13, 2021, updates research protocol to include description for the cases involving compensation and revised wording regarding amendments. Research personnel was updated to add Allyson Senger and Sara Hooshangi and to remove Ayaan Kazerouni. Recruitment materials were updated to revise Canvas assignment instructions for participation credit in the survey and an email inviting students to participate in an interview session. Consent forms were updated to revise consent form and survey instrument for the CS2114 F21 survey and the consent form for a voluntary interview session. Data collection instruments were updated to include survey instrument for a survey about help-seeking behavior in CS2114 and materials for an interview session.

Date*	OSP Number	Sponsor	Grant Comparison Conducted?
11/12/2017	PSLTFPLY	National Science Foundation (Title: BCC-EHR: Collaborative Research: Community-building and Infrastructure Design for Data-Intensive Research in Computer Science Education)	Not required (VT is not primary inst.)

* Date this proposal number was compared, assessed as not requiring comparison, or comparison information was revised.

If this protocol is to cover any other grant proposals, please contact the HRPP office (irb@vt.edu) immediately.