Analysis of the Adoption of Try-With-Resources in Java Projects

Carolyn Seglem
Department of Computer Science,
Arkansas State University
Jonesboro, Arkansas
carolyn.seglem@smail.astate.edu

Noah Whited
Department of Computer Science,
Arkansas State University
Jonesboro, Arkansas
noah.whited@smail.astate.edu

Kaleb Burdin
Department of Computer Science,
Arkansas State University
Jonesboro, Arkansas
kaleb.burdin@smail.astate.edu

Ryouji Sezai
Department of Computer Science,
Arkansas State University
Jonesboro, Arkansas
ryouji.sezai@smail.astate.edu

ABSTRACT

The motivation behind this project was to figure out the real life usage and implementation of the try-with-resources language feature within Java. With the adoption of this feature, developers can more elegantly close their resources while making their code more ridge. The problem was there was no way to determine if this feature was ever actually being implemented in real projects. Our approach to this was to expand upon a project provided to us in our class Software Engineering. Our results drew us to believe that the try-with-resources language feature is not being widely used. Although it does reduce the lines of code, we could not find sufficient evidence supporting its widespread use.

1 INTRODUCTION

Traditionally, try and catch blocks may be followed by a finally block, which allows a programmer to designate code that will execute every time a try-catch structure is entered, regardless of any exceptions that are encountered. Often, it is used for manually closing resources that have been opened before or within the try block. Though finally blocks are generally sufficient for most cases of resource management, they are frequently forgotten by programmers. Additionally, any exception thrown in a finally block will have the unfortunate effect of suppressing the exception thrown in the try block. To address this problem, programmers would often add a second try-catch statement within the finally block; that was till the try-with-resources statement made an appearance.

The try-with-resources statement was introduced in Java SE 7. It was designed to address the deficiencies associated with performing cleanup in the finally block. Rather than declare a resource within the body of a try block, programmers can instead declare that resource inside parentheses following the try keyword. At the end of the try block, the resource is automatically released, without the need for the programmer to manually close it. Just like a traditional try block, a try-with-resources block may be followed by catch and finally blocks; the resource will be closed before the catch or finally blocks begin to execute. To the average programmer, the try-with-resources statement seems to be the perfect replacement to the clunky and annoying traditional try-finally structure but there is still debate whether using the try-with-resources statement is beneficial or not.

Prior to our study, we had no knowledge of any other existing study about whether using the try-with-resources structure is actually redundant to programmers who are already used to closing resources in a finally block and who attentively close all resources as both structures have the exact same functionality. This issue became more debatable as Java 7 introduced the addSuppressed() and getSuppressed() methods in the *Throwable* class, allowing the programmer to handle suppressed exceptions. Because this can be used to resolve the problem of suppressed exceptions caused by the *finally* class, the try-with-resources statement became more redundant. To truly find the perspective of programmers, we conducted a study on 7 open source projects.

For this study, we modified a provided framework to analyze try statements, created *TryStatement*- an extension of the provided *ASTNode* class- which contain a try statement's resources, its catch, and it's finally blocks as attributes, and recorded whether there were finally blocks or resources in each try statement of the provided open

source project codes. With that information, we would figure out whether there were more try-finally statements or try-with-resources statements being used and whether programmers were converting their traditional try-finally statements into try-with-resources statements and vice versa. From our projects, we found that most programmers used the traditional try-finally structure and rarely any of the programmers converted their try-finally statements to try-with-resources statements, let alone use try-with-resources statements at all. We did find, though, that the try-with-resources statements effectively reduced the number of lines of code in each of the open source projects that used the language feature.

This paper made the following contributions:

- A study of the use of the try-with-resources language feature in real world open source projects
- Evidence showing that most programmers continue to use the traditional try-finally statement instead of the try-with-resources statement
- Evidence showing that most programmers chose not to convert existing try-finally statements to try-with-resources statements

2 RELATED WORKS

In the vast field of Computer Science/Computer Engineering, there has always been the want, even a need, to find the most effective ways to produce and run any one line of code. There have been countless studies over the past few decades to not only find the most effective way to reduce runtime but to improve readability which both flow back into creating the most efficient code possible. Try-with-resources is one such language feature that ties into these countless research studies as it was designed not only to improve readability but to reduce human error in the closure of resources. So, the question that rests with the community is: “Do such features like try-with-resources truly create a path for more efficient code?” Three such studies sought to answer this question.

2.1. In the paper “Mining Billions of AST Nodes to Study Actual and Potential Usage of Java Language Features” by Robert Dyer [1], researchers analyzed the extent to which various Java language features have been adopted by programmers and asked two questions relevant to our study: “How frequent is each feature used?” and “Was old code converted to use a new language form?” This study focused on a combined total of eighteen features, including our research interest, the try-with-resources language feature. The goals of this paper were remarkably similar to our study but simply in a more vast scope with their eighteen language features as our study focused merely on the try-with-resources language feature. The study found that only 0.21% of projects and 0.1% of files used a try-with-resources statement and only 99 files across 17 projects converted try-finally statements to a try-with-resources structure. This study was excellent in providing a broad understanding of how programmers utilized new language features but fell short when it came to each individual language feature as the wide scope prevents it from having any time to truly delve into how a particular feature tends to be used. Though our project only focuses on a single feature, it will have the advantage of being able to discuss try-with-resources in far greater depth.

2.2. In the paper “Performance of Lambda Expressions in Java 8” by A. Ward and D. Duego [2], researchers studied the effectiveness of a new language feature introduced in Java 8, lambda expressions. Ward and Duego found that this new language feature was able to create more concise and efficient code by removing the need for a standard while or for loop which both require more code to be written. Before the introduction of lambda functions, tasks such as reduction, filtering, collecting, mapping, passing in functions and predicates, and calling class methods required lines and lines of code but lambda expressions were able to reduce these tasks down to one simple line of code, reduce the number of lines of code and increasing efficiency of the program. This language feature is comparable to that of the language feature we studied, try-with-resources, as both were designed to reduce the number of lines of code and to increase efficiency. Now, the study by A. Ward and D. Duego isn’t the perfect example of a related work when it comes to our project as it focuses on a different language feature entirely but it has a great resemblance to our language feature, allowing us to produce a similar study.

2.3. In the paper “Extending the JastAdd Extensible Java Compiler to Java 7” by Jesper Oqvist and Gorel Hedin [3], researchers discuss their experiences from extending JastAddJ to support Java 7; more specifically, they discuss how the try-with-resources statement and the diamond operator could be implemented and the difference in the Java 7 compiler regarding code size, compiler time, and memory usage. The JastAddJ framework contains a class modeling the regular try statement (classified as the *TryStmt* class) was later extended, with the node *TryWithResources*, to support static semantics for try-with-resources. Oqvist and Hedin noted that there are new kinds of possible compile time errors for try-with-resources statements such as declaring a resource of a type that does not implement AutoCloseable – an

interface that automatically declares the close method used to close a resource instead of . The JastAddJ compiler collects compile time error messages through calling the method typeCheck on all the nodes which is implemented for the class ResourceDeclaration in order to check for these compile time errors. That is, the JastAddJ compiler removes one of the biggest disadvantages of try-with-resources where the main feature, automatic resource management by auto closing resources, by type checking the resources passed into the try-with-resources statement. Oqvist and Hedin deemed the JasAddJ compiler superior over the javac compiler due to this error checking but they didn't go more in depth in their research to determine if a finally block could also check these types of errors and thus making our project much more important to determine whether a try-with-resources statement isn't as efficient as a simple finally block.

In regards to try-with-resources, programmers in the Computer Science/Computer Engineering field simply have either focused too shallowly on studying the usage of the language feature or too broad on the subject as a whole. Being composed of all undergraduate college students, our group is no shape or form experts in this area but through our extensive research and implementation on multiple real world open source projects, we have the needed knowledge to effectively discuss and present our findings in regards to the three expertly written related works. These three studies helped present promising examples for our group to compare our results and to depthen our studies though they lack the data that our group has become dedicated to find by either being too broad subject matter wise or simply by not going in depth enough. Those lacking features are what enabled our to become a necessity not only for us but to our community as well.

3 BACKGROUND

For our project we choose to study the syntactic sugar known as try-with-resources in Java. Introduced in July 2011, try-with-resources was aimed to reduce lines of code, improve readability, and more importantly prevent programmers from making trivial mistakes.

The traditional way of closing resources such as files in Java is to include a finally block of code at the end of your try statement. That way no matter what happens in the try or catch block the resources are always closed. This can lead to developers forgetting to close their resources causing time consuming and trivial errors that could otherwise be easily prevented. The finally block can increase the lines of code by an unnecessarily large amount while making the code less legible.

Is try-with-resources widely used and has it helped remove the need for a finally block? These were our research questions we aimed for this study. We modeled our framework around the software provided to us by Dr. Kim. Using a Java server, client, and AST nodes we were able to answer these questions.

4 IMPLEMENTATION

We selected seven open-source projects written primarily in Java. These projects were Bouncy Castle, Byte Buddy, jMonkey Engine, Geoserver, Graphhopper, Jetty, and Lucene. We downloaded the full history of each project from its Git repository and stored them on our local machines. Table ? shows each project together with its start and end dates (as of the time of the writing of this report), lines of code, and the number of try-with-resources statements detected in our analysis.

Our language analysis framework was composed of two primary components: a Java server and a client written in Python. This framework was derived from the work of Parnin et al. [4] and Kim and Yi [5].

Within the Java server, we used the org.eclipse.jdt.core.dom package, which is part of the Eclipse Java Development Tools (JDT), to examine the code from the open-source projects in the form of an Abstract Syntax Tree [6]. Visitor classes within our server were responsible for accepting and performing analysis on all instances of their respective language feature. All instances of TryStatement nodes within the AST were sent to our TryVisitor class, which allowed us to examine each try statement in the code and produce Boolean values indicating whether it had resources, catch clauses, or a finally block. The server also extracted detailed information about the try statement's location within the code, the time at which the change was committed, and the developer who had committed the code to the repository.

The results of each try statement analysis were returned to the client in string format. The client parsed the information from the server and added the data for each try statement to a table in our database. Within the client, we also examined how each try statement instance changed over time. For each revision in the project's history, we compared the properties of each try statement found in the files directly before and directly after the revision. Specifically, we searched for instances in which a developer removed a finally block from a try statement and replaced it with a try-with-resources structure. We created a separate table in our database to record the location and revision number of each such replacement. The total number of entries in this database is representative of the total number of times the

programmers in these projects converted a traditional try-catch-finally structure into a try-with-resources structure.

At each revision, our framework calculated the number of try statements that made use of the resources structure and the number of try statements that made use of a finally block. This data was stored in .tsv files on our local machines.

5 ANALYSIS

5.1 RQ1: Is Try-With-Resources Widely Used?

5.1.1 Number of Instances in Code. Every instance of a try structure located within the Java code of our open-source projects was stored within our database along with information about its attributes. At each revision, our framework calculated the number of try statements that made use of the resources structure and the number of try statements that made use of a finally block. Using the GNU Octave software, we plotted the number of try statements with finally blocks against the number of try-with-resources statements. The results are visible in Figure 1.

Try-with-resources was introduced in Java 1.7, which was released in 2011. However, most of the projects we studied did not begin to make use of try-with-resources until several years later, often in the period between 2014 and 2016. Jetty saw a comparably early adoption of try-with-resources, with its first instance appearing in 2012.

5.1.2 Number of Project Contributors Using Try-With-Resources. Using the information stored in our database, we next counted the number of unique users whose contributions to the open-source project included a try-with-resources statement. The project with the greatest number of contributors who utilized try-with-resources was Geoserver. Fittingly, it was also the project with the greatest total number of contributors. Jetty had the greatest percentage of contributors who used try-with-resources, with approximately 9% of contributors adding at least one try-with-resources statement. Byte Buddy, which had no instances of try-with-resources at all, predictably had no contributors who made use of try-with-resources. Our results for each project are detailed below in Table 2.

Table 2. Contributors Who Used Try-With-Resources

Project	Contributors Adding Try-with-resources	Total Contributor Count
Byte Buddy	0	62
Geoserver	21	398
Graphhopper	3	108
Jetty	15	170
jMonkey Engine	5	196

Our analysis showed that in each project, an extremely small subset of developers was responsible for all instances of try-with-resources used within the code. This leads us to conclude that try-with-resources remains a language feature that is not widely used among the general population.

5.2 RQ2: Has Try-With-Resources Helped Remove the Need for a Finally Block?

Even in revisions dating as late as 2020, we found that projects continue to utilize traditional try-catch-finally structure far less frequently than try-with-resources. In six of the seven projects we analyzed, there remained a wide disparity between the number of finally blocks and the number of try-with-resources statements. One of these projects, Byte Buddy, was found to not have even one instance of try-with-resources. Jetty was the only project that formed the exception to this rule. In late 2018, the number of try-with-resources statements in Jetty surpassed the number of finally blocks. The disparity between Jetty's number of try-with-resources statements and try statements with finally blocks has only increased since 2018.

Our client code compared the open-source projects before and after every revision to detect instances in which a developer converted a try-catch-finally structure to a try-with-resources structure. Our results showed that this is an extremely rare practice. No conversions were detected in four of our seven open-source projects, while the maximum number of conversions totaled to 63 in Lucene. The number of conversions to try-with-resources for each project can be seen below in Table 3.

Table 3. Conversions From Finally Blocks to Try-With-Resources

Project	Number of Conversions
Bouncy Castle	0
Byte Buddy	0
Geoserver	23
Graphhopper	0
Jetty	34
jMonkey Engine	0
Lucene	63

From our results, we can infer that developers have been slow to adopt try-with-resources for newly written code, and they are extremely reluctant to update old code to replace finally blocks with try-with-resources. Part of this is likely due to the late introduction of try-with-resources to the Java language. By the time try-with-resources had been introduced to Java, the Java language was already 16 years old. We assume that most developers by that point had become accustomed to the traditional structures of the language and could see little to no need to adjust to a new structure which serves primarily as syntactic sugar. The increase in the usage of try-with-resources in recent years may then be correlated with the emergence of a new generation of programmers who learned Java after the release of try-with-resources. From our analysis, we can see that there is a gradual increase in the rate at which developers are beginning to utilize try-with-resources. If our predictions are correct, we expect that the rate of usage of try-with-resources will continue to increase in the future.

Of course, it should not be expected that the introduction of try-with-resources should lead to the eradication of finally blocks altogether. In practice, finally blocks are used almost exclusively for ensuring that resources are closed, but developers have used them for other purposes as well. Moreover, it is extremely inconvenient for developers to search through millions of lines of code, replacing structures that already work well. However, considering the advantages try-with-resources can offer compared to finally blocks, it seems that developers are neglecting a significant opportunity to improve their code quality.

6 CONCLUSION

Our group has analyzed the usage of try-with-resources in 7 open source projects. We found out that only a few developers in each project were responsible for all uses of this feature, and in almost all projects, it was used far less than try-finally structures. The one of the reasons is they were extremely reluctant to convert previously existing try-finally structures into try-with-resources. However, the results from our analysis leads us to believe that try-with-resources reduce the total line of codes. In addition, the number of developers using it are increasing over time. Therefore, we hope try-with-resources can replace try-finally to improve code quality in the future.

REFERENCES

[2] A. Ward & D. Deugo, Performance of Lambda Expressions in Java 8. *Int'l Conf. Software Eng. Research and Practice | SERP'15 |*. 2015.

[4] Chris Parnin, Christian Bird, and Emerson Murphy-Hill. 2011. Java Generics Adoption: How New Features are Introduced, Championed, or Ignored. *MSR 2011: Proceedings of the 8th International Working Conference on Mining Software Repositories*, 3-12. DOI: https://doi.org/10.1145/1985441.1985446

[5] Donghoon Kim and Gangman Yi. 2014. Measuring Syntactic Sugar Usage in Programming Languages: An Empirical Study of C# and Java Projects. *Advances in Computer Science and its Applications. Lecture Notes in Electrical Engineering,* vol. 279*,* 279-284. DOI: https://doi.org/10.1007/978-3-642-41674-3_40

[1] Dyer, R., Rajan, H., Nguyen, H., and Nguyen, T. N. May 2014. Mining Billions of AST Nodes to Study Actual and Potential Usage of Java Language Features. *ICSE 2014: Proceedings of the 36th International Conference on Software Engineering*, 779-790. DOI: https://doi.org/10.1145/2568225.2568295

[6] IBM Knowledge Center. Class ASTNode. Retrieved December 8, 2020 from https://www.ibm.com/support/knowledgecenter/en/SS5JSH_9.5.0/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/ASTNode.html.

[3] Jesper Oqvist and Gorel Hedin. September 2013. Extending the JastAdd extensible Java compiler to Java 7. *Proceeding of the 2013 International Conference on Principle and Practices of Programming on the Java Platform: Virtual machines, Languages, and Tools*. 147-152. DOI: https://doi.org/10.1145/2500828.2500843

Project Name	Start Date	End Date	Lines of Code	Try-With-Resources Expressions
Bouncy Castle	February, 2013	November, 2020	1,113,102	60
Byte Buddy	November, 2013	November, 2020	162,635	0
Geoserver	April, 2001	December, 2020	1,568,767	340
Graphhopper	April, 2012	November, 2020	123,823	1,622
Jetty	August, 1998	July, 2019	1,396,226	2,836
jMonkey Engine	March, 2011	December 2020	822,149	720
Lucene	September, 2001	December, 2020	726,777	93,762