An Empirical Assessment of the Adoption of Java Generics and Lambda Expressions

Daniella Angelos
Computer Science Department
University of Brasília
Brasília, Brazil

Thiago Cavalcanti
Computer Science Department
University of Brasília
Brasília, Brazil

Vinícius Correa
Computer Science Department
University of Brasília
Brasília, Brazil

Rodrigo Bonifácio
Computer Science Department
University of Brasília
Brasília, Brazil

ABSTRACT

Modernizing legacy systems towards the evolution of the underlining programming language has been reported as a challenging task. For this reason, developers often prefer to maintain the use of existing constructs instead of using new language features. This scenario of language evolution might be well exemplified using two remarkable releases of Java programming language— Java SE 5.0 (2004) and Java SE 8 (2014), for instance, in which considerable language improvements were proposed. Java SE 5.0 introduced parametric polymorphism to the language (using Java Generics) and Java SE 8 introduced Lambda Expressions. In this paper we empirically investigate the adoption of both features by considering open-source Java systems. In the case of Java Generics, differently from existing reasearch works, we compare the adoption of this language feature by considering two distinct groups. In the first group we only consider systems whose initial development started before the release of Java SE 5.0. In the second group we consider systems whose initial development started at least five years after the release of Java SE 5.0. In the case of Lambda Expressions, we contribute as one the first attempts to empirically characterize the common usage of that language construct. Our results reveal that...

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

SAC'16, April 4-8, 2016, Pisa, Italy
Copyright 2016 ACM 978-1-4503-3739-7/16/04...$15.00
http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

Keywords

Java Generics; lambda expressions; language evolution

1. INTRODUCTION

Programming languages have to evolve to better address both techonology trends and developers needs. For instance, the Java programming language, which might be considered a reasonably recent language, presents features and constructs that differ significantly from its initial release in 1996. This kind of language evolution leads to a lot of dicussion regarding how this changes are embraced, used in practice, or ignored by the community of developers []. One significant change to the Java language was the introduction of parametric polymorphism using Java Generics in 2004, which provides larger support for classes and methods generalization as well as improved support for software evolution. At that time, Java Generics was considered a significant improvement because it would simplify software construction by allowing the design of generic behavior using parametric types.

Several years later, Parnin et al. carried out an investigation to understand how Java Generics had been adopted in practice []. In their study, they found out that over half of the projects and developers did not use generics at that time, and for those that did, the use was consistently narrow. They discovered, empirically, that generics were almost entirily used to either hold or traverse collections of objects in a type safe manner. Nevertheless, the mentioned work did not answer a relevant question: is there any difference in the adoption of Java Generics when comparing two groups of systems based on the release date of this feature: one group of projects whose development started before Java SE 5.0 and one group of projects whose initial releases started after Java SE 5.0? Answering to this question might give ....

In this paper we first replicate the work of Pet et al. [] trying to answer aditional research questions **??**.

More recently, in 2014 a new version of the Java language was released (Java SE 8), introducing a long-waited feature that addresses some (limited) support for functional programming mechanisms: Lambda Expressions. In this paper we also characterize the adoption of that Java programming language construct, which might guide software developers to better understand the most common situations where Lambda Expressions should be used. In summary, the contributions of this paper are two-fold

- We replicate an existing study that investigates the adoption of Java Generics in open-source systems []. Differently from the original work [], our research also aims at understanding whether developers of recently developed systems embrace the use of Generics more extensively than developers of fully developed systems—whose initial releases started several years before Sun Microsystems launched Java SE 5.0 in 2004.
- We characterize how Java developers are using Lambda Expressions, a new Java language construct introduced in 2014 (Java SE 8). To the best of our knowledge, there is no other empirical study that investigates this issue.

Roadmap. The remaining of this paper is organized as follows. In the next section . . .

2. OVERVIEW

2.1 Generics

In 2004 Sun Microsystems introduced generics in benefit to remove ambiguity in collections like not specify the type and not doing so much cast in the collections. Elimination of casts. The following code snippet without generics requires casting:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

When re-written to use generics, the code does not require casting:

```
List<String> list = new ArrayList<String>()
    ;
list.add("hello");
String s = list.get(0);   // no cast
```

2.2 Lambda

Like intent to help write a better code *Java 8.* A significantly change focus in Collection API with addition of *Streams* that allow us to write a better code about collections-processing at a hight level of abstraction. A good solution provide for *Stream* is a trouble about concurrency that solve this with simple loops for and while was necessary a hard work and many boilerplate while *parallelStream* is a elegant way to this.

The difference between *Collection* and *Stream* is that *Collection* is an data structure in memory while *Stream*is a conceptually fixed data structure that computed elements by demand. When we using *Collection Interface* require iteration provide by user while *Stream Interface* uses internal iteration. Such as real example extracted Jetty 9.3.0:

```
List<String> actual = new ArrayList
    <>();
for (Path path:finder.getHits()){
 actual.add(hb.toShortForm(path.
     toFile()));
}

List<String> actual = finder.
    getHits().stream().map(hb.
    toShortForm(Path::toFile)).
    collect(toList());

for(String parent:module.
    getParentNames()){
 System.out.printf("Depend:%s%n",
     parent);
}

module.getParentNames().stream().
    forEach(parent->System.out.
    printf("Depend:%s%n",parent));
```

Based in this examples we have interested to research cases like this to find real opportunities of refactoring to prove that a adoption lambda expression in Java projects wold be a simple way to re-write a code and discovery if community has adopted Lambda to create a better code.
Like this we have 2 questions about lambda to guide this work: **RQ1:** How was adoption of Lambda by the community?
RQ2: There are any construction with Lambda in a projects?

3. RELATED WORK

After a release of new language feature, manifold studies are published related to use and benefits of it. This section gives an overview of researches about Generics and Lambda Expression.

3.1 Generics claims and empirical studies

There are two basic types of works about Generics, the first one explore claims about its use in several contexts, and the benefits it brings. The second type performs empirical studies and tries to verify the claims and hypothesis made by some of the articles in the first type. As the question we want answered is related to the use of this feature by developers a decade after its release, we'll give a breef only in a few works on empirical study.

mencionar Mining Billions of AST Nodes to Study Actual and Potential Usage of Java Language Features

The aforementioned *Java Generics Adoption: How New Features are Introduced, Championed, or Ignored* empirical study about the use of Generics in open source communities is the main result we have related to Java Generics use until now. Several claims were tested in 20 “most used” projects at that time, but their focus were only in three of these projects: JEDIT, ECLIPSE-CS and SQUIRREL-SQL. Their study concluded that Generics wasn't being used in a relevant manner. mencionar outros estudos que citam este trabalho . Our goal is to replicate part of this study and avaliate if, even a few years later, the use of Generics is still restricted to either hold or traverse collections of objects in a type safe manner, or if developers are using it more broadly.

3.2 Lambda Expressions

4. INVESTIGATION

4.1 Projects studied

We choose 27 open source projects with the last public release in 2015. Here are we choose *Ant*, *Hibernate orm*, *Openmeetings*, *Tomcat*, *Archiva*, *Jclouds*, *Platform Community*, *UniversalMediaServer*, *Bonita-Tools*, *Jenkins*,*Postgree jdbc*, *Wicket*,*Cassandra*, *Jmeter*, *Sonar*, *Wildfly*,*Checkstyle*, *Log4j*, *Spark*, *Woden*, *Crawler4j* ,*Maven*, *Spring*, *Zookeeper*,*Eclipse*, *MyFaces* and *Storm* we choose this because are constantly update and has a large adoption on open source community.

4.2 Methodology

We start doing static analyzing in all *.java* files in open source projects that we choose, to collected data around how often the community generics and lambda expressions at the time was generated an AST to each file source. Based on structure and abstraction provided from Eclipse JDT we choose this API to support generates a parse tree to each file and creation of each visitor [1]. We choose to use the framework Spring to inject the visitors such as dependence. Therefore like an ideal configuration in visitor [1] when found a ideal construction as base from this paper, this is save to analysis with R language [**?**].

The Static Analyses has a visitor to search Lambda Expression because we have a intent to discovery how community has been accepted this feature so waited. Another hand, we cant leave explain how community has ignored an opportunity to make a simple refactoring in large scale when not use Lambda to replace a Enhanced For to iterate a Collection. Like it each occur of Lambda Expression and Enhanced For are save in a csv file to analysis marking source file and line of start and project name end of each construction founded.

5. RESULTS

5.1 Lambda

After analysis around 3.5M LOC, we founded 438 cases to Lambda inside unit test. We concluded that a large use in unit test probability because developers work with anonymous methods provide for Lambda and to increase a productive and isn't necessary create methods to make tests.

Another relevant point is that developers ignore a real opportunities to refactoring iteration in Collections with Stream and prefer use *Enhanced For* to do it. In this research we founded 18821 cases in 23 projects to convert *Enhanced For* to *Stream* interface provide that avoid some boilerplate and open opportunities to do concurrency in Collections through **parallelStream**, in figure 5.1 occur about Enhanced.

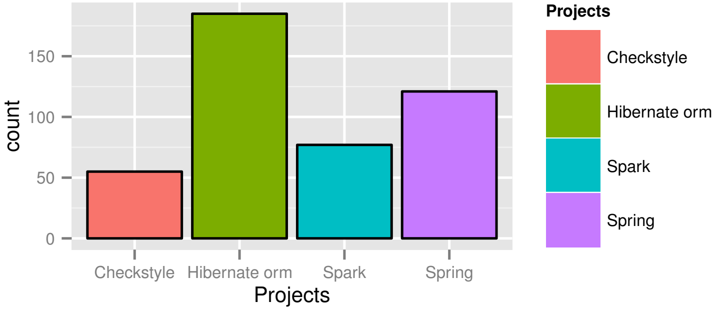

Figure 1: Use of Lambda in unit tests.

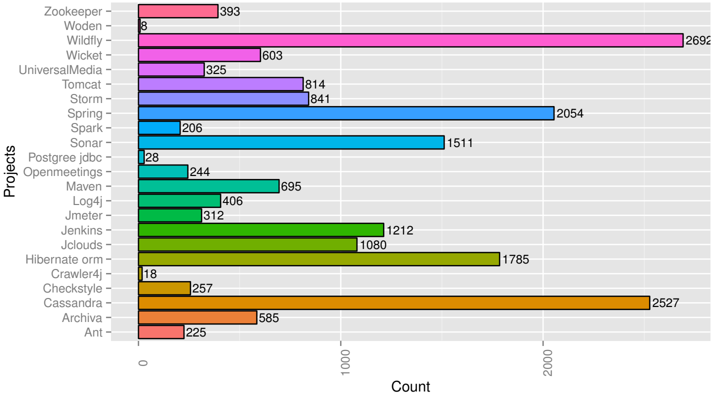

Figure 2: Oportunites to replace For by Lambda.

6. FUTURE WORK

Generics ????

Maybe with a large adoption of community for lambda become possible the the use of language Scale was increased. However that a paradigm function was incorporated in Java and was a good work discovery if it can replace constructors in Scala. Another point is that Lambda provide solution more elegant to concurrency in a loop avoid boilerplate and generated a construct more clear but the big problems is how test Lambda Expression with a elegant form showed this construction are anonymous methods.

7. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

8. ADDITIONAL AUTHORS

9. REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A.2.7 A Caveat for the TeX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.

C. REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.