

ASE 2021



Why Do Developers Remove **LAMBDA EXPRESSIONS** in Java?

MINGWEI ZHENG, Jun Yang , Ming Wen, Hengcheng Zhu , Yepang Liu , Hai Jin

zmw12306@gmail.com



Huazhong University of
Science and Technology

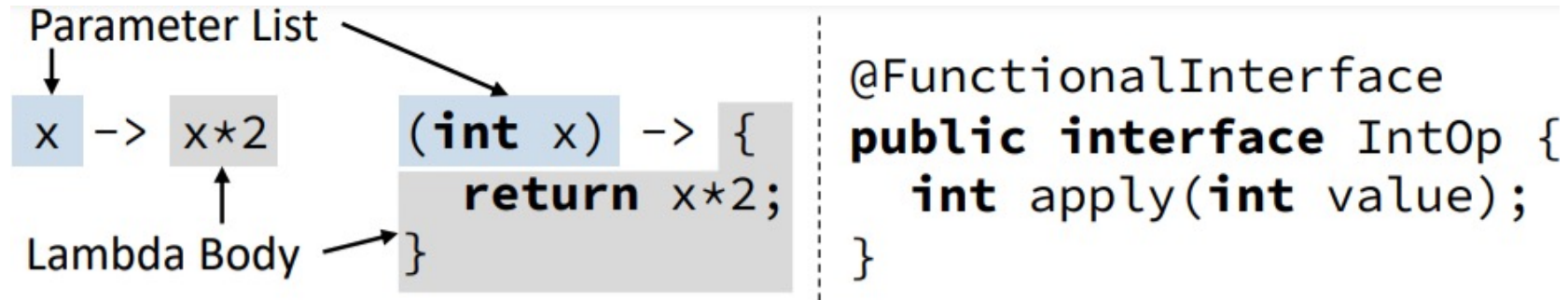


Southern University of
Science and Technology



The Hong Kong University of
Science and Technology

Java Lambda Expressions



- **Lambda Expression:**

parameter list **+** the arrow token (\rightarrow) **+** lambda body.

- **Functional Interface:**

an interface that has just one abstract method. (aside from the methods of Object)



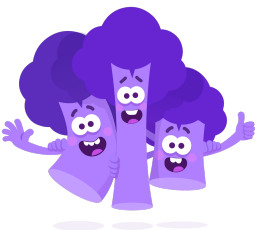
Related Work

- Mazinianian et al. “Understanding the *use of lambda expressions* in Java”.
[OOPSLA 2017]

Thus far, all research works focus on the usages of Java lambda expressions, how about the misuse of lambda expressions in Java?

compreension of java programs ? . [SBES 2019]

- ...



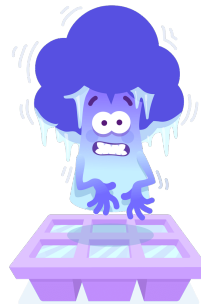
Misuse of Lambda Expressions

Misuse: a lambda expression is used inappropriately which causes side effects or even induces bugs.

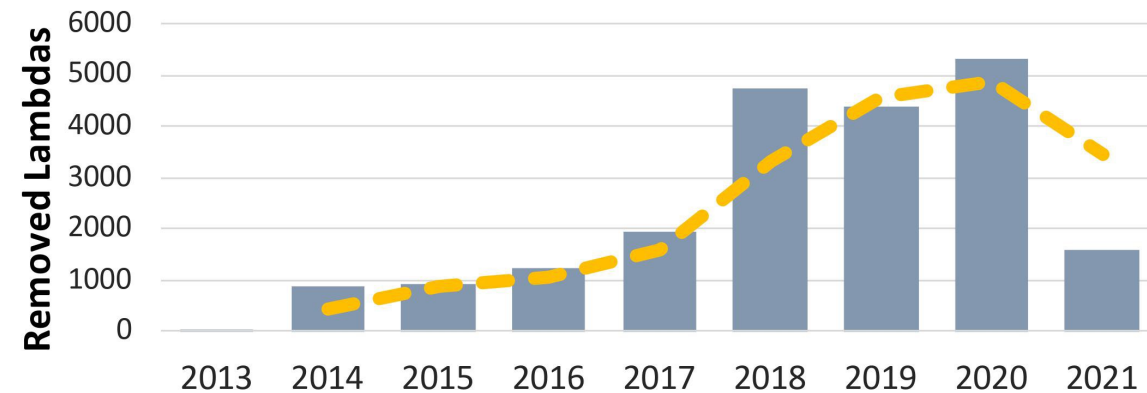
significance

```
synchronized DocumentsWriterDeleteQueue  
advanceQueue(int maxNumPendingOps) {  
    return new DocumentsWriterDeleteQueue(infoStream,  
        generation + 1, seqNo + 1, () -> nextSeqNo.get() -  
1);  
}
```

500 bytes of memory leakage on each full flush (LUCENE-9478)

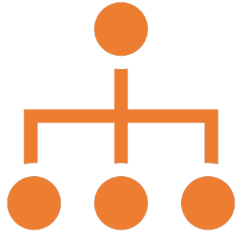


pervasiveness



Trend of lambda removals

What We Explored?



RQ1: What lambda expressions are more frequently removed by developers?



RQ2: Why do developers remove lambda expressions in practice? What are the reasons behind and impacts?



RQ3: What are the migration patterns of the inappropriate usages of lambda expressions?

Our Empirical Study

Quantitative study

- Collected 3,662 removed lambdas and 31,288 kept ones
- Understand the characteristics of removed lambda expressions

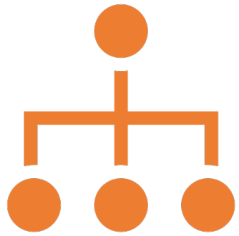


Qualitative study

- Collected 117 real-world issues and conducted a user study
- Explored reasons, impacts, and migration patterns of lambda removals



What Did We Explore?



RQ1: What lambda expressions are more frequently removed by developers?

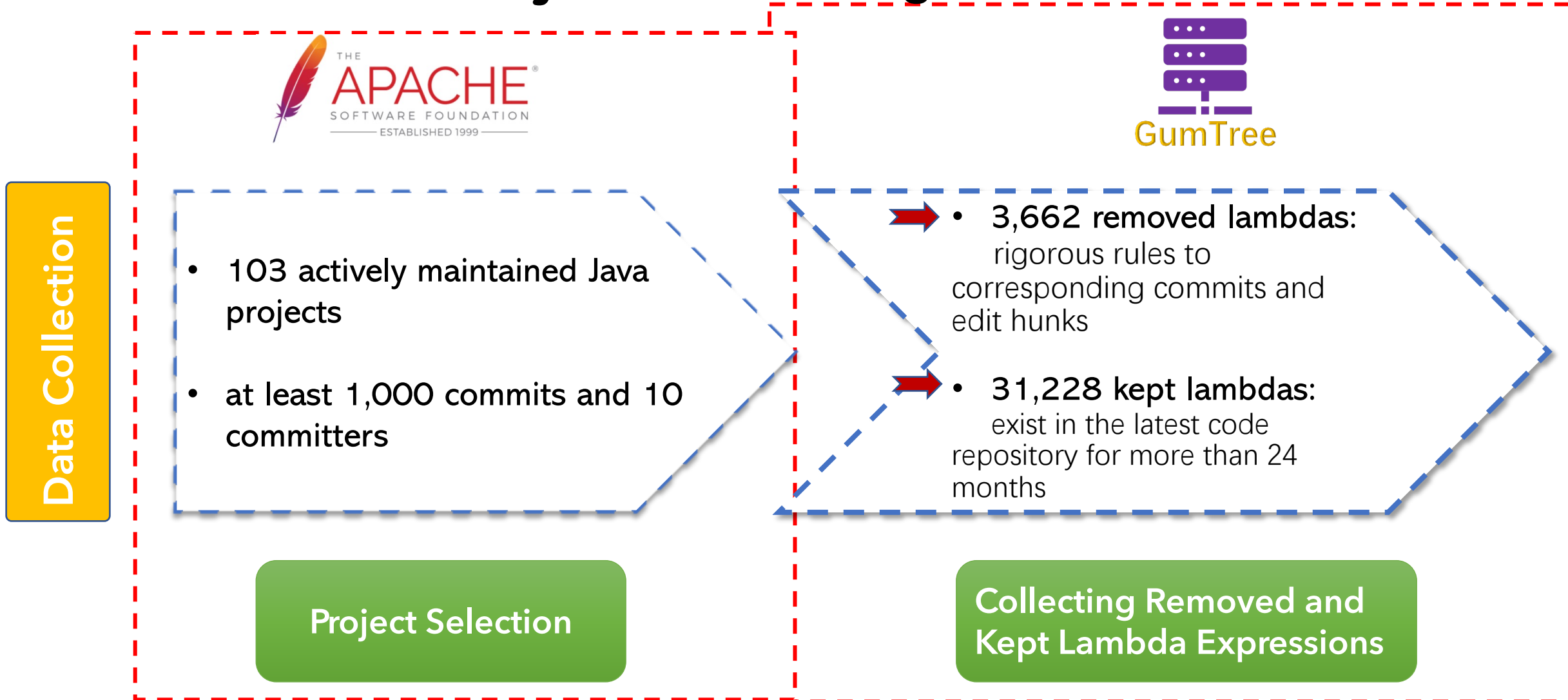


RQ2: Why do developers remove lambda expressions in practice? What are the reasons behind and impacts?



RQ3: What are the migration patterns of the inappropriate usages of lambda expressions?

Quantitative Study: characterizing the removed lambdas



Compare Removed Lambdas and Kept Lambdas

Lifetime of removed lambdas

The time period the lambda expression has stayed in the project.



The usages of functional interfaces

Built-in functional interfaces

VS

Customized functional interfaces

The complexity of lambdas

Parameter number

Lines of code

Lambda body length

Variable number

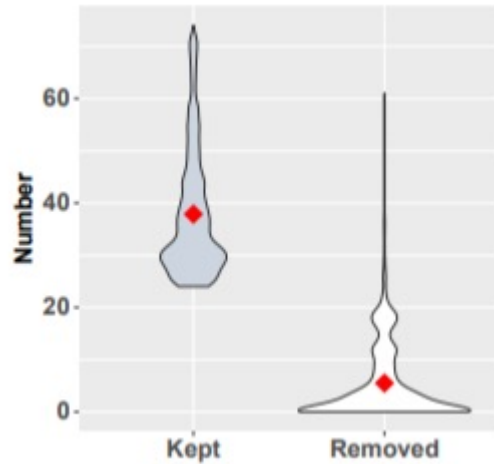
The contexts of lambdas

method APIs: built-in APIs
or self-defined APIs?



Characteristics from 4 Perspectives

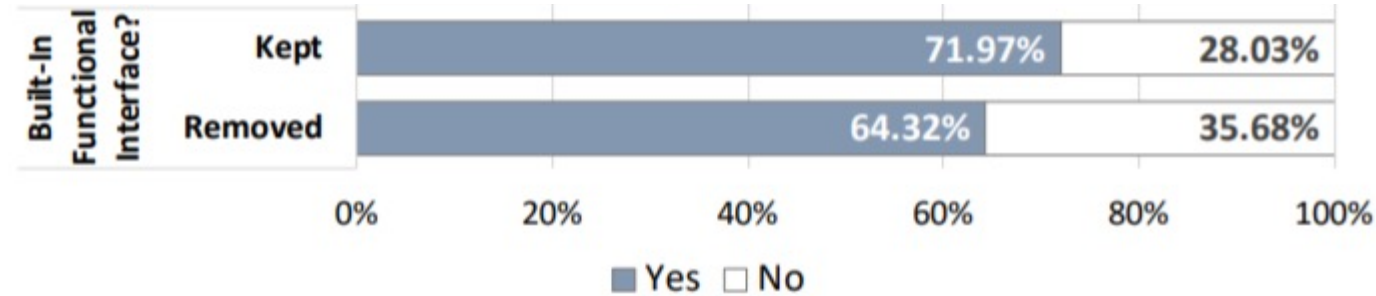
➔ • Lifetime of removed lambdas



38.26% no longer than one month

20.23% more than one year

➔ • The usage of functional interface

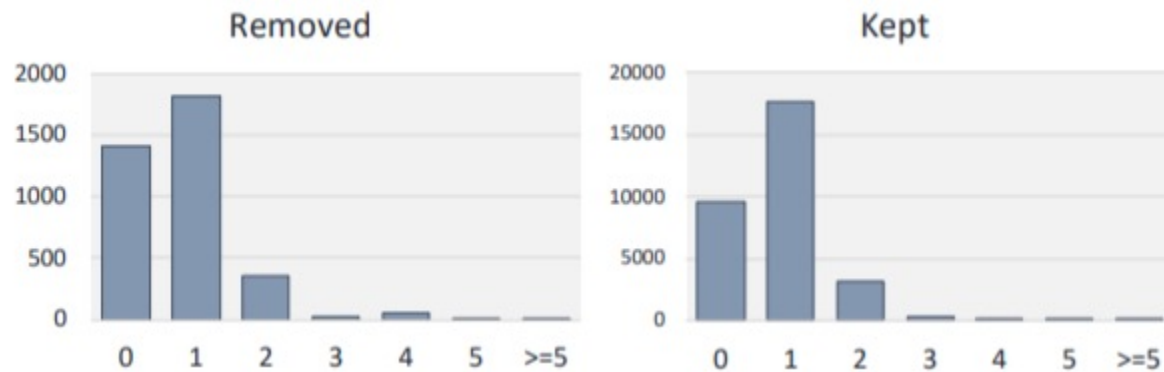


Lambda expressions built on top of customized functional interfaces are more likely to be removed.

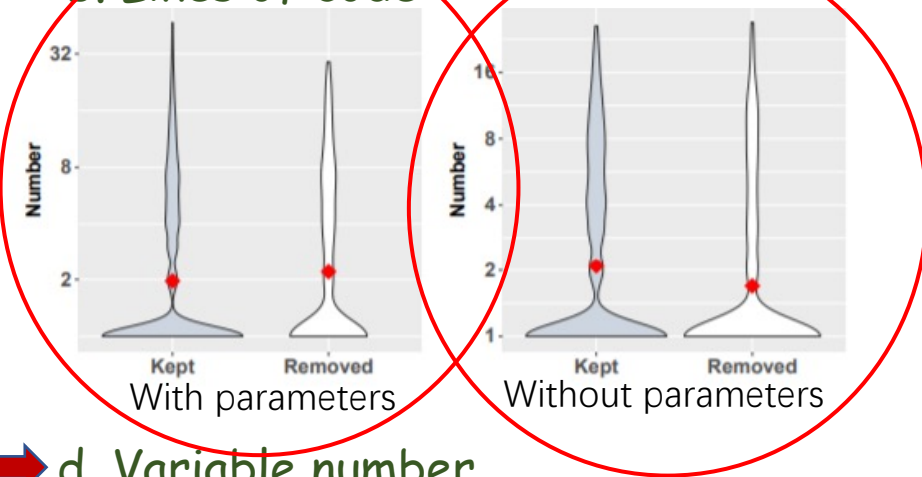
Characteristics from 4 Perspectives

- The complexity of lambda expressions

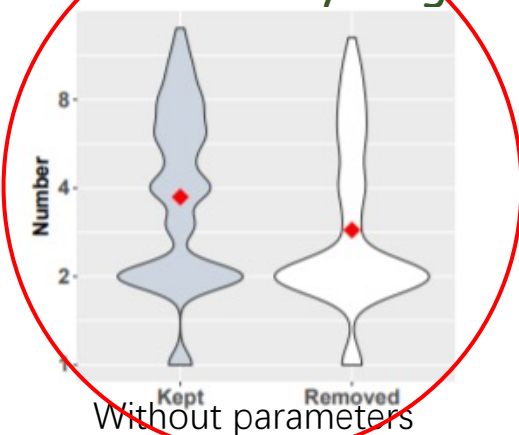
➔ a. Parameter number



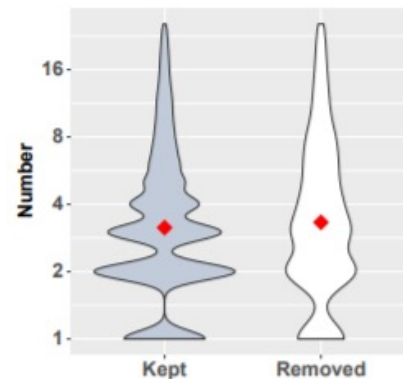
b. Lines of code



c. Lambda body length



➔ d. Variable number



4.89 VS 5.49

Characteristics from 4 Perspectives

- The contexts of lambda expressions

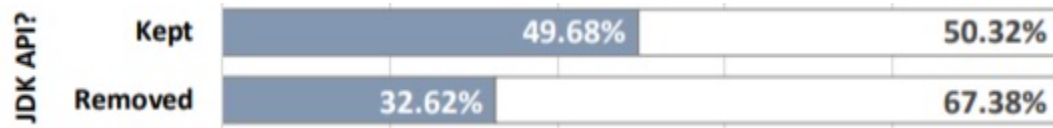
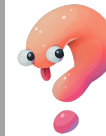


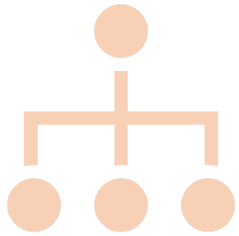
Table I: APIs that Lambda Expressions Are Passed To

API	Removed	Kept	Difference
<code>Iterable.forEach</code>	13.78%	12.65%	1.13%
<code>Stream.map</code>	8.49%	13.93%	-5.44%
<code>Map.computeIfAbsent</code>	7.69%	4.57%	3.12%
<code>Optional.ifPresent</code>	7.05%	2.07%	4.98%
<code>Stream.forEach</code>	5.29%	4.96%	0.33%
<code>Stream.filter</code>	5.29%	14.20%	-8.91%
<code>Collectors.toMap</code>	4.49%	4.70%	-0.21%
<code>Map.forEach</code>	4.01%	3.50%	0.51%
<code>ExecutorService.submit</code>	3.69%	2.59%	1.10%
<code>Optional.map</code>	3.53%	1.51%	2.02%
<code>IntStream.forEach</code>	2.40%	1.48%	0.92%



Lambda expressions that are built on top of customized functional interfaces, passed to self-defined method invocations are more likely to be removed.

What Did We Explore?



RQ1: What lambda expressions are more frequently removed by developers?

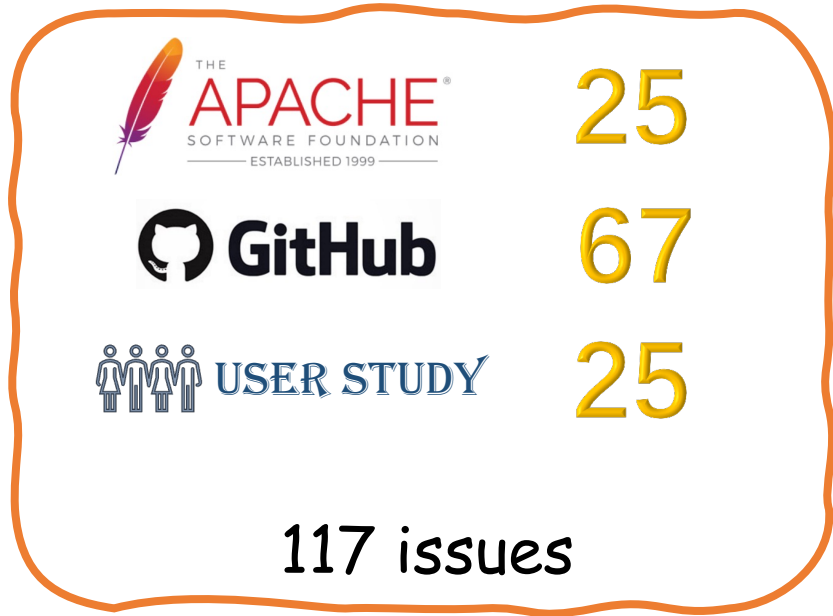


RQ2: Why do developers remove lambda expressions in practice? What are the reasons behind and impacts?

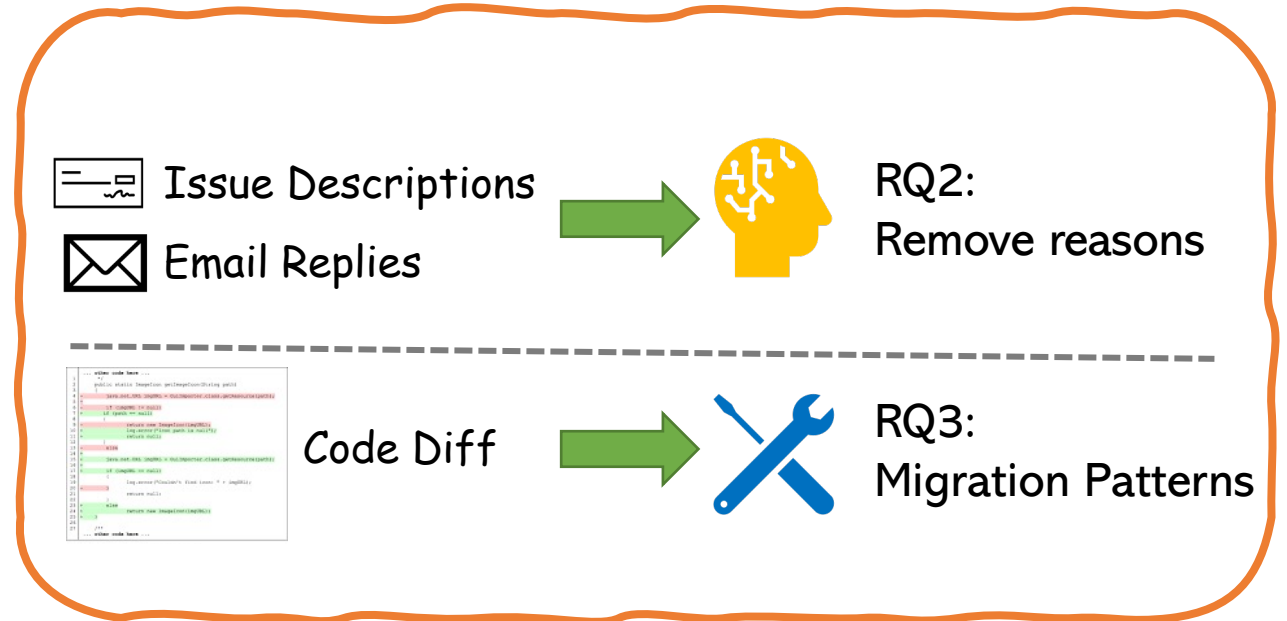


RQ3: What are the migration patterns of the inappropriate usages of lambda expressions?

Qualitative Study: concerns and actions of developers

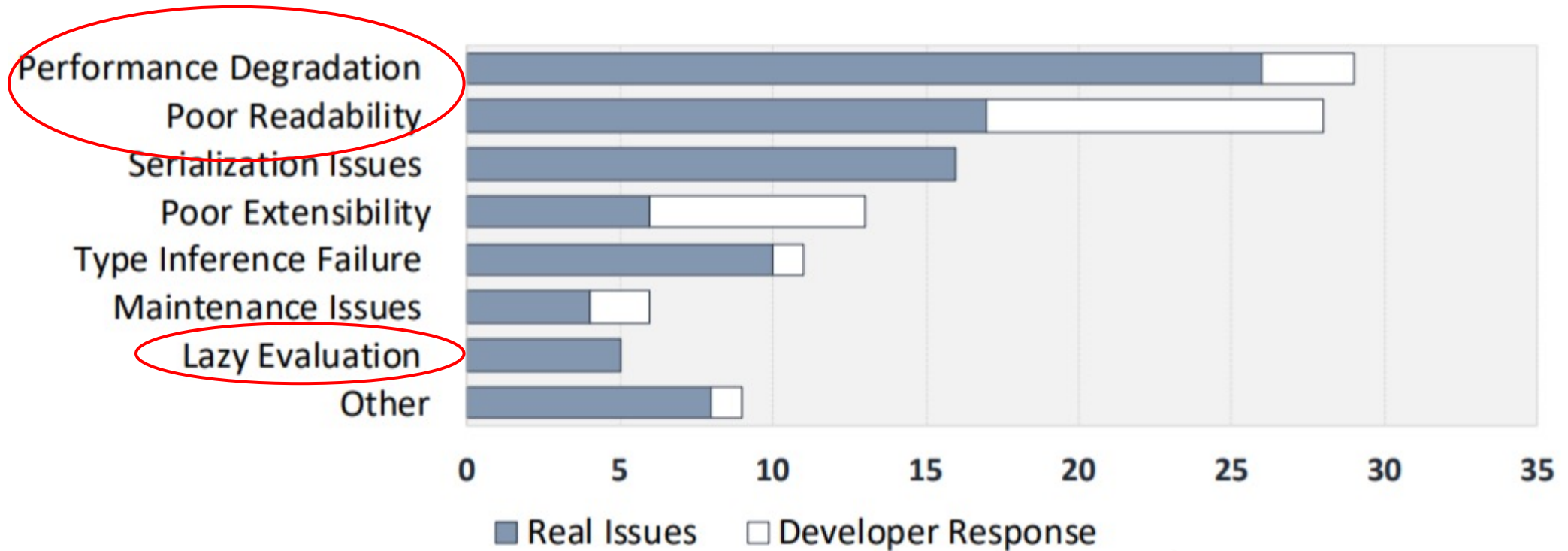


Data Collection



Manual Analysis

Reasons of Removing Lambda Expressions



7 major reasons for removal



Reason 1: Performance Degradation (29/117)

- excessive object allocations, significant garbage collection
- memory leaks

```
226 - private static <T> Stream<T> cyclingShuffledStream(Collection<T> collection)
227 - {
228 -     List<T> list = new ArrayList<>(collection);
229 -     Collections.shuffle(list);
230 -     return Stream.generate(() -> list).flatMap(List::stream);
223 +     ImmutableList.Builder<InternalNode> distribution = ImmutableList.builderWithExpectedSize(bucketCount);
224 +     for (int i = 0; i < bucketCount; i++) {
225 +         distribution.add(shuffledNodes.get(i % shuffledNodes.size()));
226 +     }
227 +     return distribution.build();
```

Commit f555aa6, project presto

Reason 2: Poor Readability (28/117)

- long body or complex logic, anonymous

```
-         .collect(Collectors.groupingBy(url -> {  
-             if (UrlUtils.isConfigurator(url)) {  
-                 return CONFIGURATORS_CATEGORY;  
-             } else if (UrlUtils.isRoute(url)) {  
-                 return ROUTERS_CATEGORY;  
-             } else if (UrlUtils.isProvider(url)) {  
-                 return PROVIDERS_CATEGORY;  
-             }  
-             return "";  
-         }));  
+         .collect(Collectors.groupingBy(this::judgeCategory));
```

```
+     private String judgeCategory(URL url) {  
+         if (UrlUtils.isConfigurator(url)) {  
+             return CONFIGURATORS_CATEGORY;  
+         } else if (UrlUtils.isRoute(url)) {  
+             return ROUTERS_CATEGORY;  
+         } else if (UrlUtils.isProvider(url)) {  
+             return PROVIDERS_CATEGORY;  
+         }  
+         return "";  
+     }
```

Commit#9e9517d, Project Dubbo



A developer in project **Apache Calcite**, *"I generally dislike long lambdas due to bad readability"*

Reason 7: Lazy Evaluation (5/117)




- delay the evaluation of an expression until its value is needed

```
- // Lazily create a blackboard that contains all non-generated columns.
- final Supplier<Blackboard> bb = () -> {
-     RexNode sourceRef = rexBuilder.makeRangeReference(scan);
-     return createInsertBlackboard(table, sourceRef,
-         table.getRowType().getFieldNames());
- };
+ final RexNode sourceRef = rexBuilder.makeRangeReference(scan);
+ final Blackboard bb = createInsertBlackboard(table, sourceRef,
+     table.getRowType().getFieldNames());
- list.add(ief.newColumnDefaultValue(table, f.getIndex(), bb.get()));
+ list.add(ief.newColumnDefaultValue(table, f.getIndex(), bb));
```

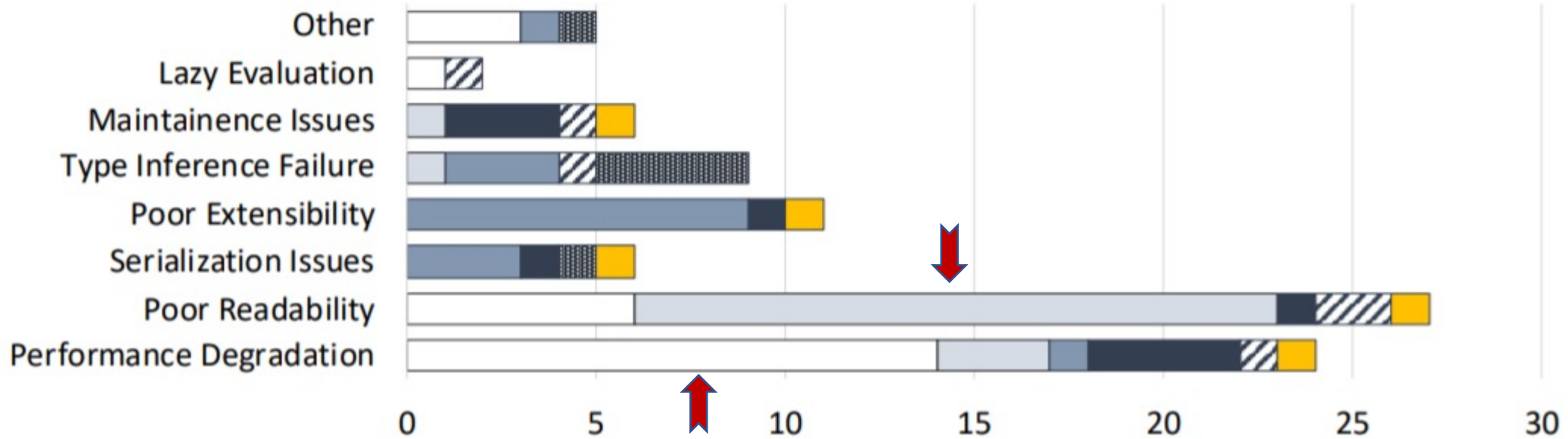
Common Migration Patterns

- 7 major migration patterns

Table II: Migration Patterns of Removing Lambda Expressions

ID	Types	Description	Frequency
	Lambda passed to new Java 8 API methods ⇒ Conventional methods	Replace new Java 8 API, i.e., <code>Collections</code> , <code>Stream</code> and <code>Optionals</code> , with conventional <code>for loop</code> , <code>do while</code> , <code>enhanced for loops</code> , <code>if else</code> , etc.	24
	Lambda ⇒ Method reference	Lambda expressions are refactored into method reference to improve readability or performance. In some cases, the lambda body is too large and should be first extracted into another function and then being invoked with method reference.	22
	Lambda ⇒ Anonymous class	Lambda expressions are refactored into anonymous class.	17
4	Lambda ⇒ Inner class instance	The behavior existed in lambda expressions are wrapped into an newly defined inner class.	10
5	Method with lambdas are replaced with a new method	The method to which the lambda expression is passed, no longer exists and is replaced with a new method which does not accept lambdas any more.	6
6	Adding a type cast	Adding a type to provide more type information for type inference and overload resolution or implementing <code>Serializable</code> .	6
7	Existing method was changed to accept no lambdas	Parameters of the existing method are changed to accept no lambdas any more. The corresponding parameters are either changed to a new one which enclose the behavior of lambda expressions, or deleted (logics in removed lambdas are implemented in following code).	5

Correlation between Lambda Removal Reasons and Migration Patterns



Most Common (24/104): ~~(Lambda) passed to new Method & API methods~~ \Rightarrow Conventional methods

17/22 ~ improve readability

Actionable Advice for Using Lambda Expressions

- Avoid using lambdas in performance-critical code.

Performance degradation!!!

```
- Optional.ofNullable(CDI.current().getBeanManager().getExtension(TomEESecurityExtension.class))
-     .map(TomEESecurityExtension::hasAuthenticationMechanisms)
-     .filter(has -> has.equals(true))
-     .ifPresent(has -> AuthConfigFactory.getFactory()
-         .registerConfigProvider(new TomEESecurityAuthConfigProvider(),
-             null, null,
-             "TomEE Security JSR-375"));
+ final TomEESecurityExtension securityExtension =
+     CDI.current().getBeanManager().getExtension(TomEESecurityExtension.class);
+
+ if (securityExtension.hasAuthenticationMechanisms()) {
+     AuthConfigFactory.getFactory()
+         .registerConfigProvider(new TomEESecurityAuthConfigProvider(),
+             "http", ctx.getVirtualServerName() + " " + ctx.getContextPath(),
+             "TomEE Security JSR-375");
+ }
```

ng.
it#99d6f10,
t TomEE
td. The
ions"-----

Actionable Advice for Using Lambda Expressions

- Specify the type

Lambdas do not

"lambda methods
involved. An easy
specified explicitly

```
1 public void removeHivePluginFrom(Iterable<Drillbit>
2     drillbits) throws PluginException {
3     try {
4         drillbits.forEach(bit -> {
5             try {
6                 bit.getContext().getStorage().remove(pluginName);
7             } catch (PluginException e) {
8                 throw new RuntimeException("...", e);
9             }
10        });
11    } catch (RuntimeException e) {
12        throw (PluginException) e.getCause();
13    }
14 }
```

when Java generics are
the type has to be
/

- Avoid lambdas

Throwing a checked

Listing 8: Throwing a Checked exception with a Wrapper

- Avoid constructors

Bad readability

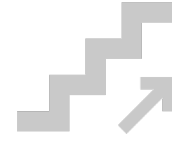
"a lambda expression
into a named function

```
1 public void removeHivePluginFrom(Iterable<Drillbit>
2     drillbits) throws PluginException {
3     for (Drillbit drillbit : drillbits) {
4         drillbit.getContext().getStorage().remove(name);
5     }
6 }
```

the code if broken out
n's reply

Listing 9: Throwing a Checked Exception with For Loop

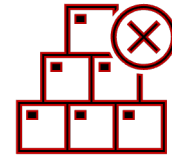
Discussion and Future Work



- ➔ • **Lambda Removal Recommender.**
 - detect certain lambda misuses and recommend a more appropriate implementation



- ➔ • **Study on more Java Functional Idioms.**
 - other functional idioms (i.e., Collections, Stream, and Optionals)



- ➔ • **Applicability to other Programming Languages.**
 - other Object-Oriented languages, such as C++, Python, etc.



Conclusion

- ➔ • A quantitative and a qualitative study on the inappropriate usages of Java lambda expressions
- ➔ • Explored 4 code-level characteristics of removed lambdas
- ➔ • Summarized 7 major reasons for lambda removals and 7 major migration patterns
- ➔ • 5 pieces of actionable advice
 - Dataset available at: <https://github.com/CGCL-codes/LambdaMisuse>

ASE 2021



Thanks!



Huazhong University of
Science and Technology



Southern University of
Science and Technology



The Hong Kong University of
Science and Technology