Ph.D. Proposal (80%) Seminar

Automatic Software Debloating

César Soto Valero

Supervised by: Thomas Durieux, Martin Monperrus, and Benoit Baudry

KTH Royal Institute of Technology

CASTOR
Software Research Centre

WASP
WALLENBERG AI,
AUTONOMOUS SYSTEMS
AND SOFTWARE PROGRAM

cesarsv@kth.se

Agenda

1. Background
2. Debloating Java dependencies (static analysis)
3. Debloating Java bytecode (dynamic analysis)
4. Lessons learned
5. Conclusion
6. Future work

Part #1: Background

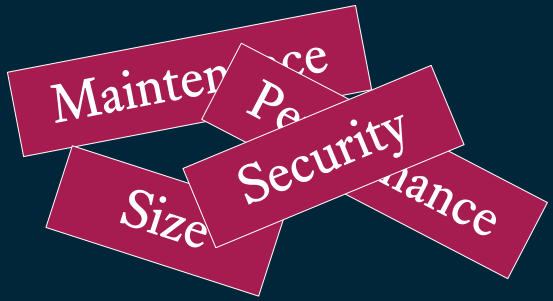
[Software bloat]

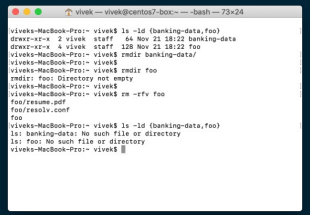
“Software tends to grow over time, whether or not there's a need for it.”

Gerard J. Holzmann. (2015). Code Inflation. *IEEE Software*, 32 (2).

Software bloat

"The term software bloat refers to code that is packaged in an application but that is not necessary for building and running the application."





- [USENIX] RAZOR : A Framework for Post-deployment Software Debloating
- [USENIX] Is Less Really More? Quantifying the Security Benefits of Debloating Web Applications
- [CCS] Slimium: Debloating the Chromium Browser with Feature Subsetting
- [TSE] TRIMMER: An Automated System for Configuration-based Software Debloating
- [USENIX] Debloating Software through Piece-Wise Compilation and Loading

Part #2: Debloating Java dependencies

[Static analysis]

Source code


```java
import com.google.common.base.Joiner;
import org.apache.spark.annotation.Private;

@Private
public class EnumUtil {

  public static <E extends Enum<E>> E parseIgnoreCase(Class<E> clz, String str) {
    E[] constants = clz.getEnumConstants();
    if (str == null) {
      return null;
    }
    for (E e : constants) {
      if (e.name().equalsIgnoreCase(str)) {
        return e;
      }
    }
    throw new IllegalArgumentException(
        String.format("Illegal type='%s'. Supported type values: %s",
            str, Joiner.on(", ").join(constants)));
  }
}
```

core/src/main/java/org/apache/spark/util/EnumUtil.java

Bytecode


```
// class version 52.0 (52)
// access flags 0x21
public class org/apache/spark/util/EnumUtil {

  // compiled from: EnumUtil.java

  // access flags 0x9
  // signature <E:Ljava/lang/Enum<TE;>;>(Ljava/lang/Class<TE;>;Ljava/lang/String;)TE;
  // declaration: E parseIgnoreCase<E extends java.lang.Enum<E>>(java.lang.Class<E>, java.lang.String)
  public static parseIgnoreCase(Ljava/lang/Class;Ljava/lang/String;)Ljava/lang/Enum;
. . .
L9
    LINENUMBER 35 L9
    INVOKESTATIC com/google/common/base/Joiner.on (Ljava/lang/String;)Lcom/google/common/base/Joiner;
    ALOAD 2
    INVOKEVIRTUAL com/google/common/base/Joiner.join ([Ljava/lang/Object;)Ljava/lang/String;
    AASTORE
. . .
```

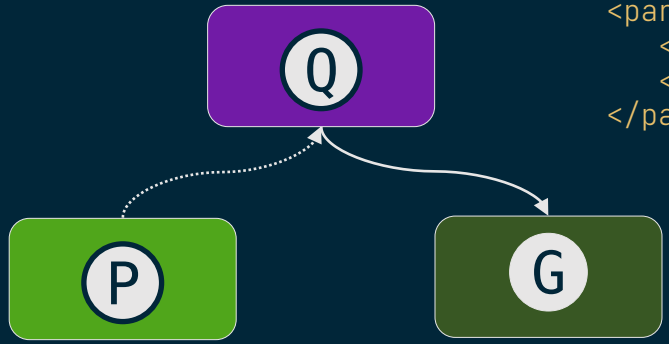
target/classes/org/apache/spark/util/EnumUtil.class

Bytecode


```
. . .
// access flags 0x9
public static on(Ljava/lang/String;)Lcom/google/common/base/Joiner;
 L0
  LINENUMBER 71 L0
  NEW com/google/common/base/Joiner
  DUP
  ALOAD 0
  INVOKESPECIAL com/google/common/base/Joiner.<init> (Ljava/lang/String;)V
  ARETURN
. . .
// access flags 0x11
// signature (Ljava/lang/Iterable<*>;)Ljava/lang/String;
// declaration: java.lang.String join(java.lang.Iterable<?>)
public final join(Ljava/lang/Iterable;)Ljava/lang/String;
 L0
  LINENUMBER 230 L0
  ALOAD 0
  ALOAD 1
  INVOKEINTERFACE java/lang/Iterable.iterator ()Ljava/util/Iterator; (itf)
  INVOKEVIRTUAL com/google/common/base/Joiner.join (Ljava/util/Iterator;)Ljava/lang/String;
  ARETURN
 . . .
```

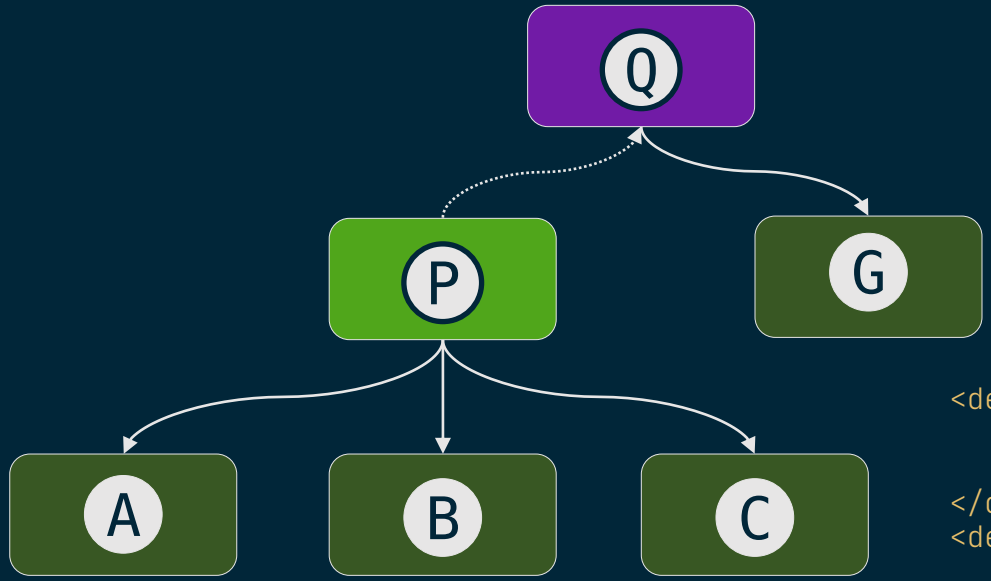
Dependency tree

Dependency tree



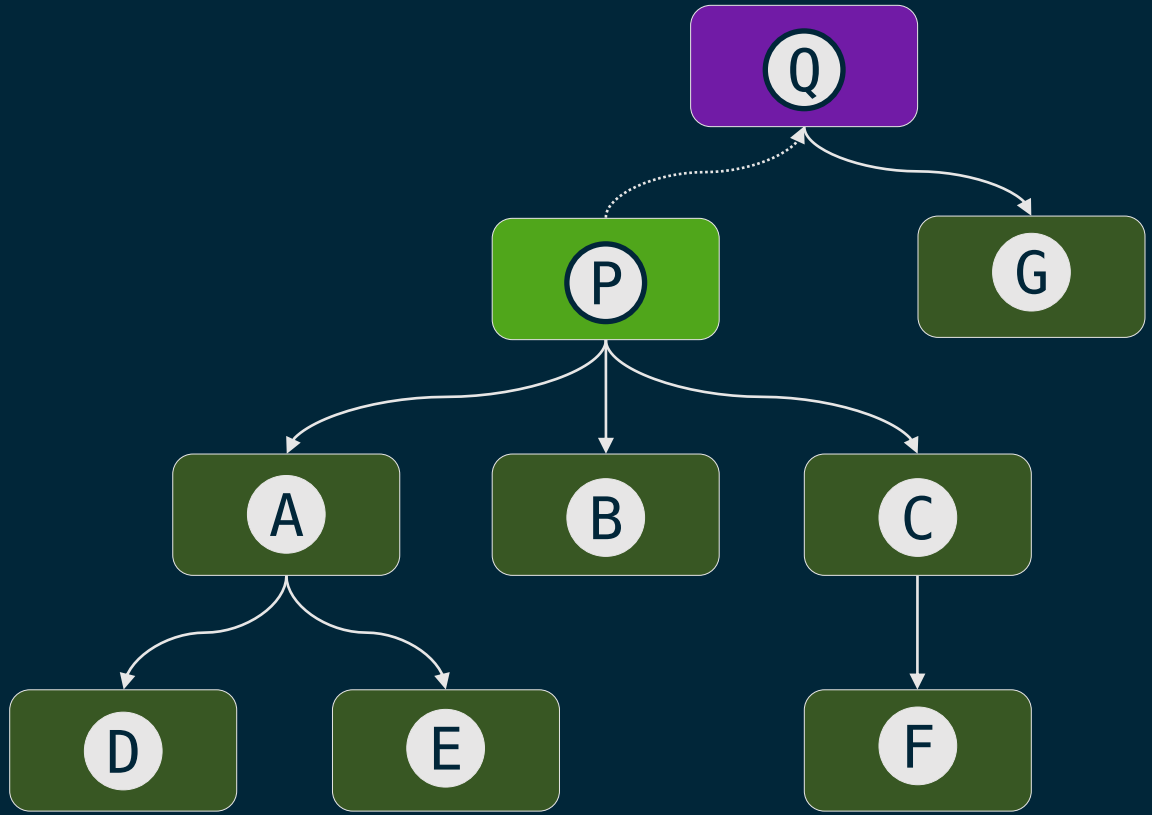

```
<parent>
   <groupId>org.Q</groupId>
   <artifactId>Q</artifactId>
</parent>
```

Dependency tree

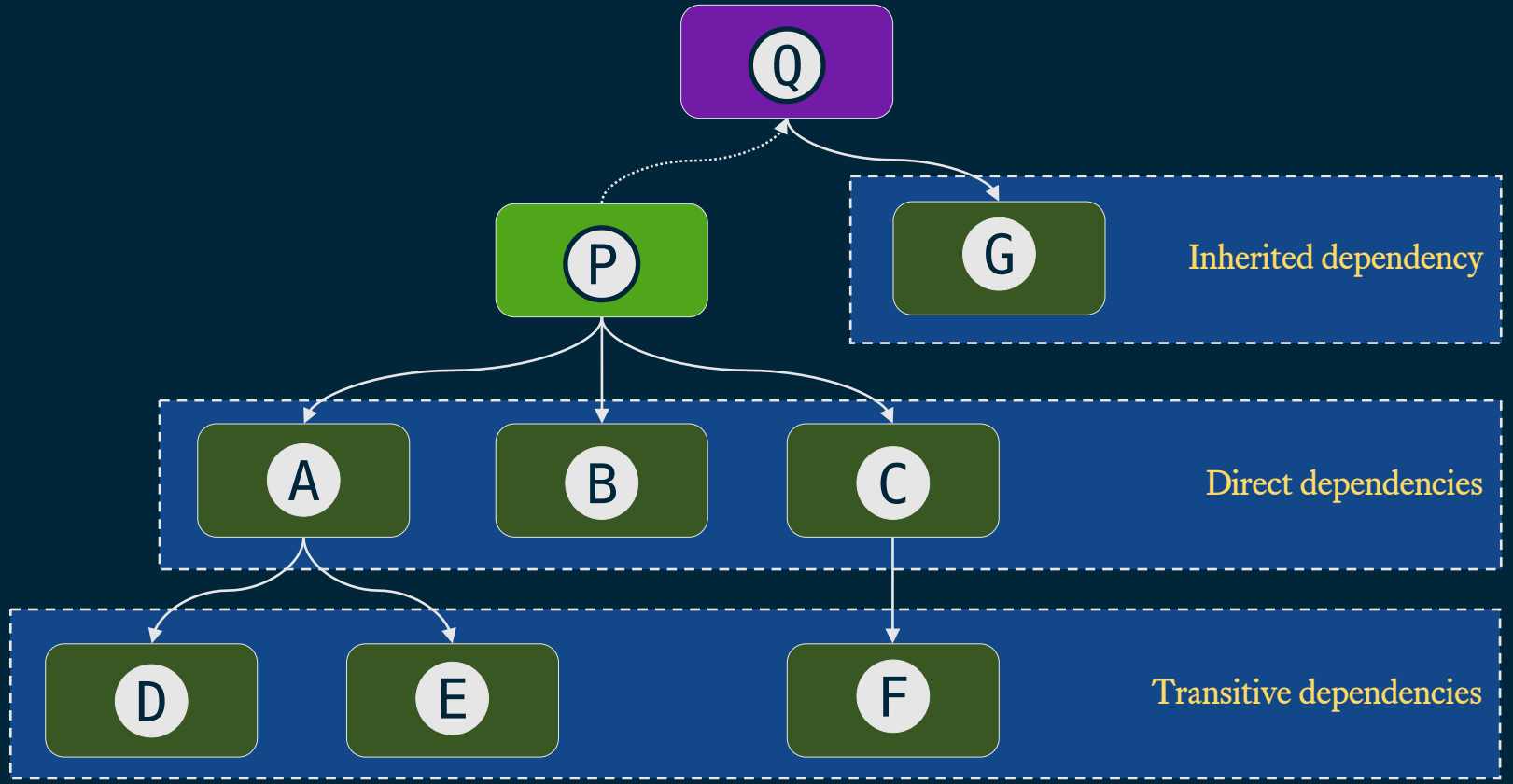



```
<dependency>
    <groupId>org.A</groupId>
    <artifactId>A</artifactId>
</dependency>
<dependency>
    <groupId>org.B</groupId>
    <artifactId>B</artifactId>
</dependency>
<dependency>
    <groupId>org.C</groupId>
    <artifactId>C</artifactId>
</dependency>
```

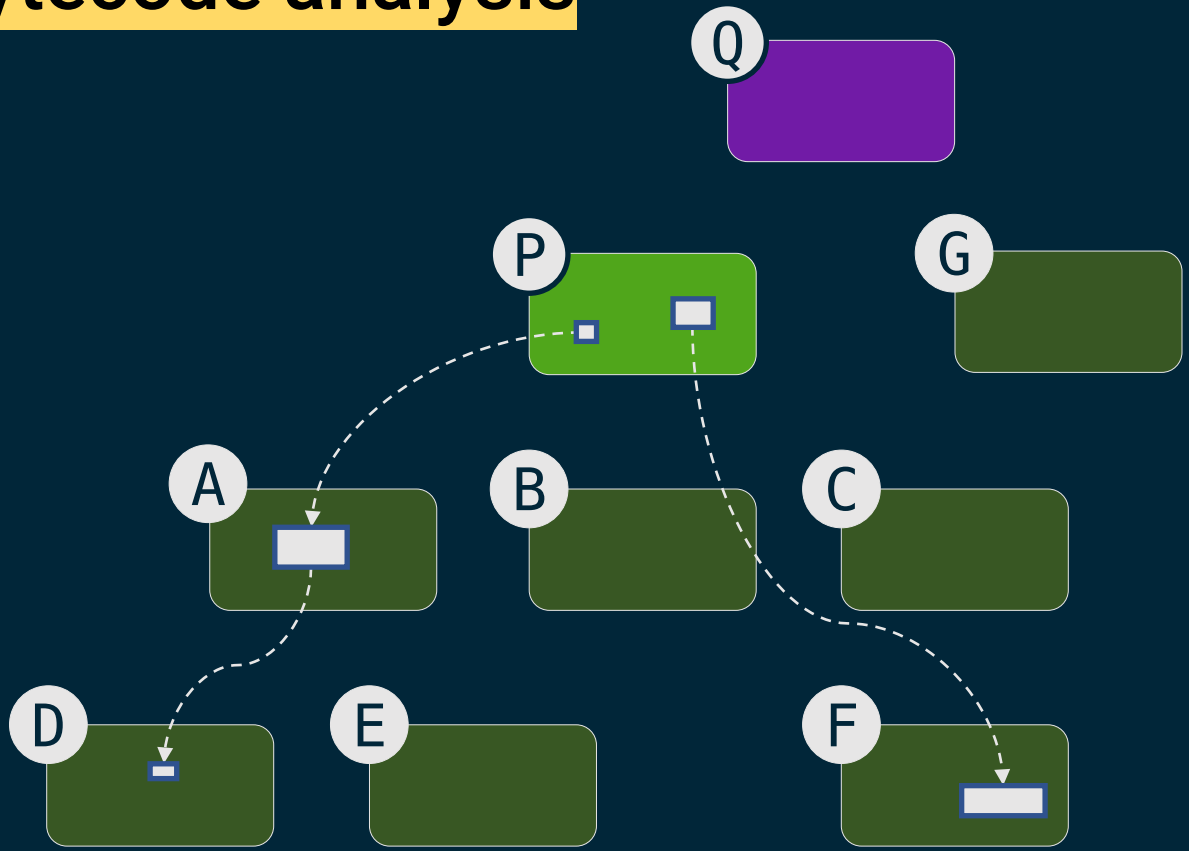
Dependency tree



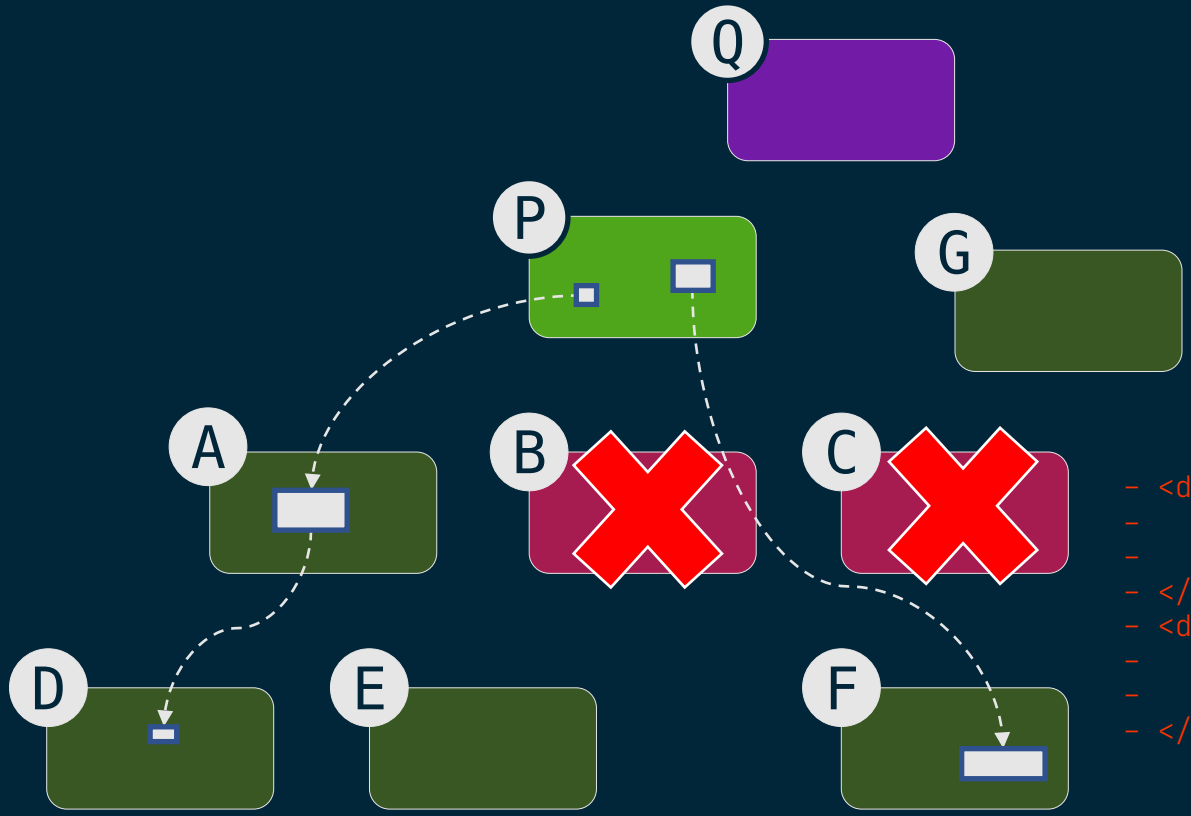
Dependency tree



Bytecode analysis

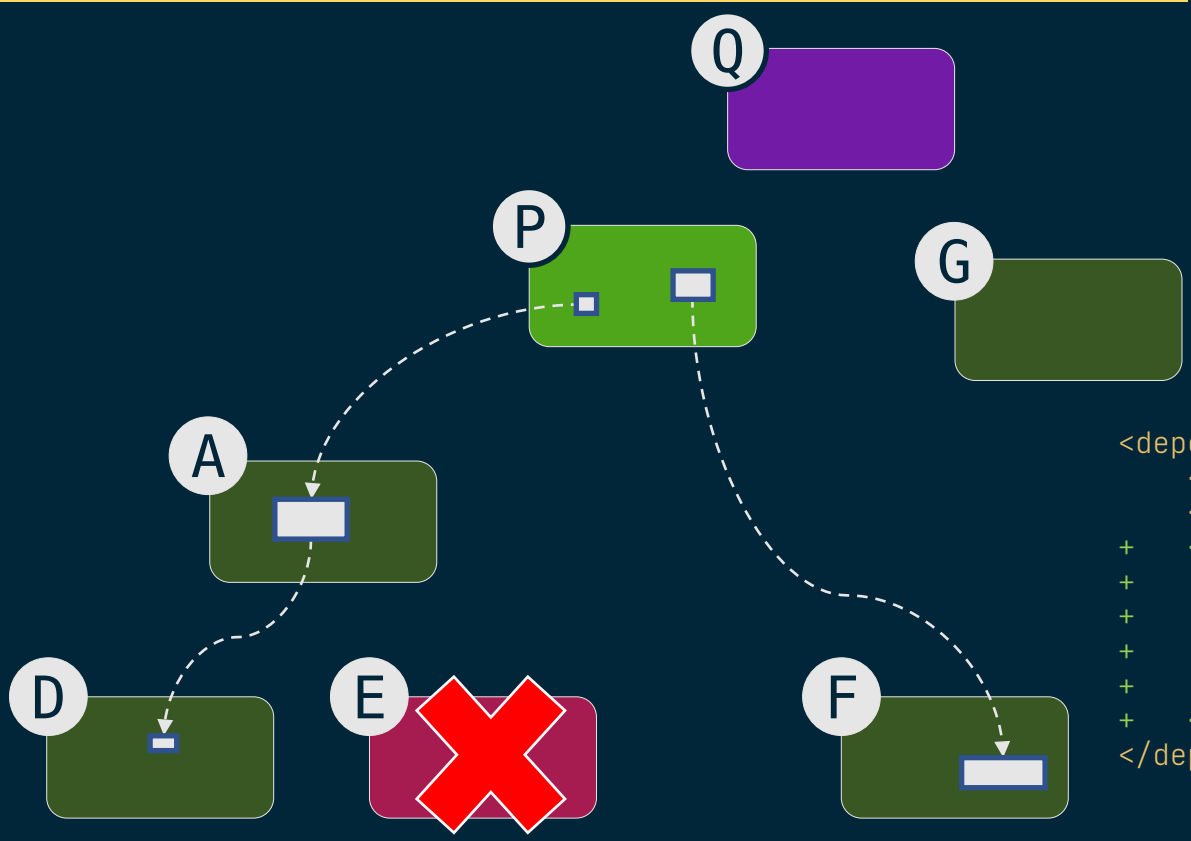


Debloating direct dependencies



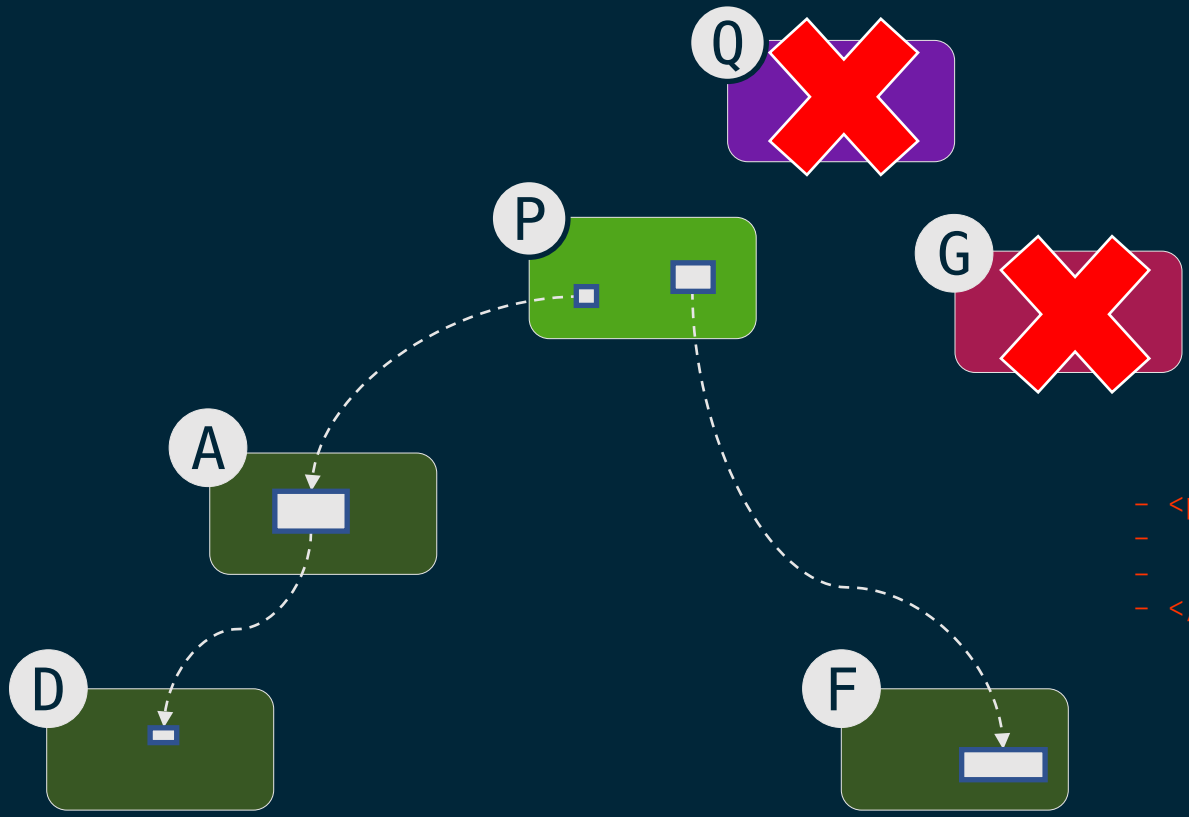

```
- <dependency>
-     <groupId>org.B</groupId>
-     <artifactId>B</artifactId>
- </dependency>
- <dependency>
-     <groupId>org.C</groupId>
-     <artifactId>C</artifactId>
- </dependency>
```

Debloating transitive dependencies



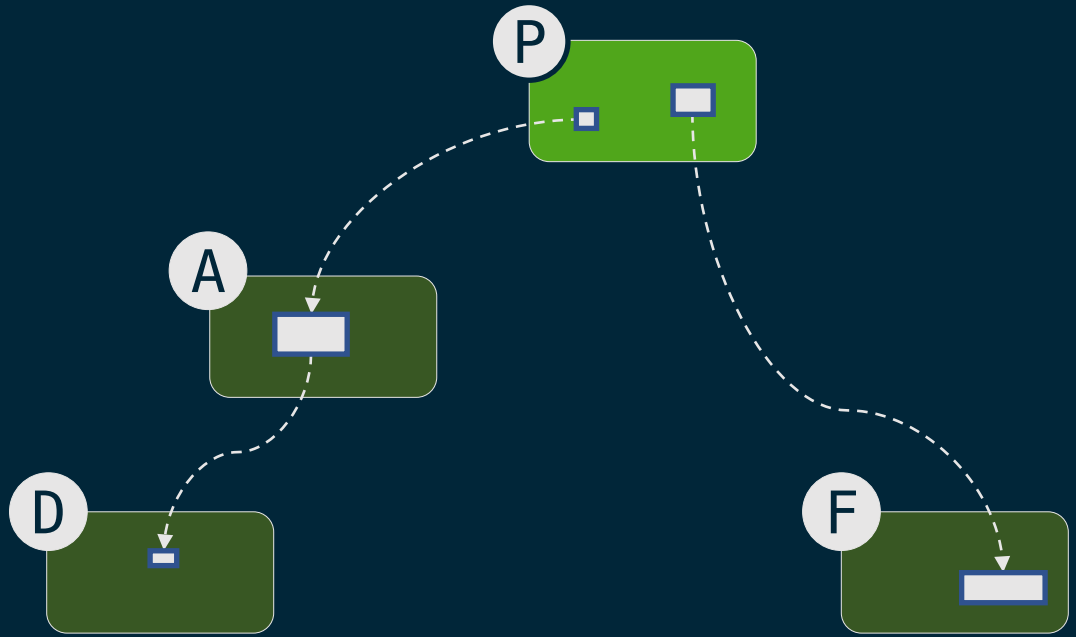

```
<dependency>
    <groupId>org.A</groupId>
    <artifactId>A</artifactId>
+   <exclusions>
+     <exclusion>
+       <groupId>org.E</groupId>
+       <artifactId>E</artifactId>
+     </exclusion>
+   </exclusions>
</dependency>
```

Debloating inherited dependencies

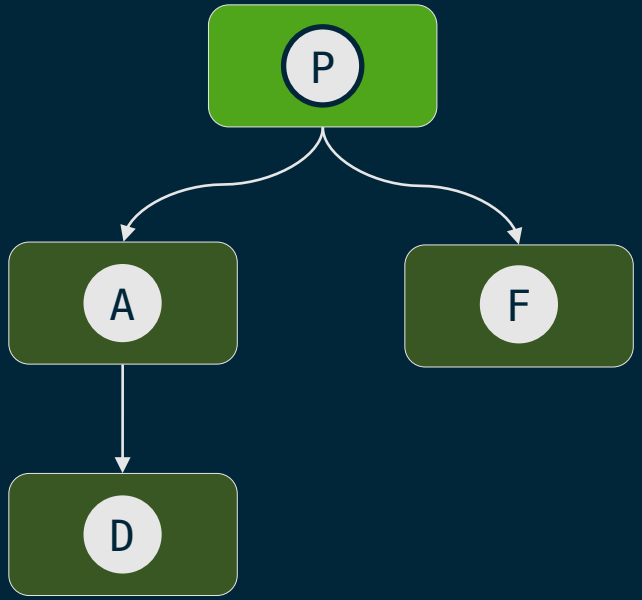



```
- <parent>
-    <groupId>org.Q</groupId>
-    <artifactId>Q</artifactId>
- </parent>
```

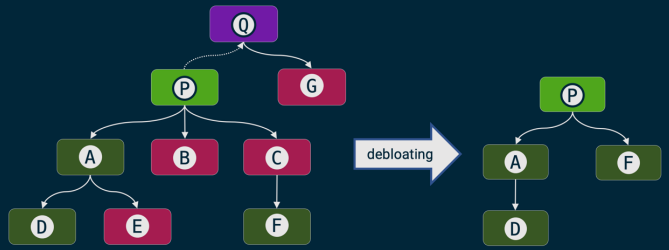
Debloating inherited dependencies



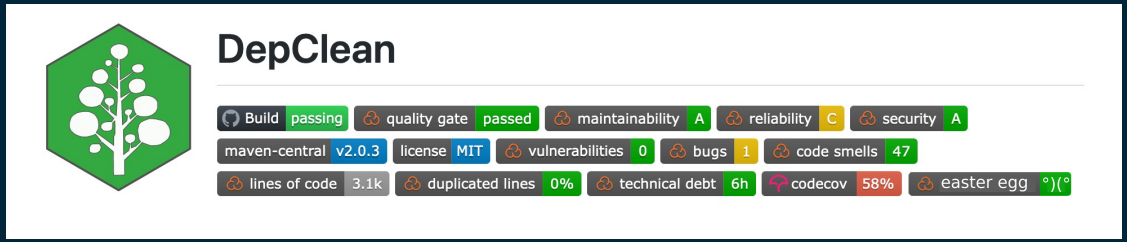
Debloated dependency tree



DepClean

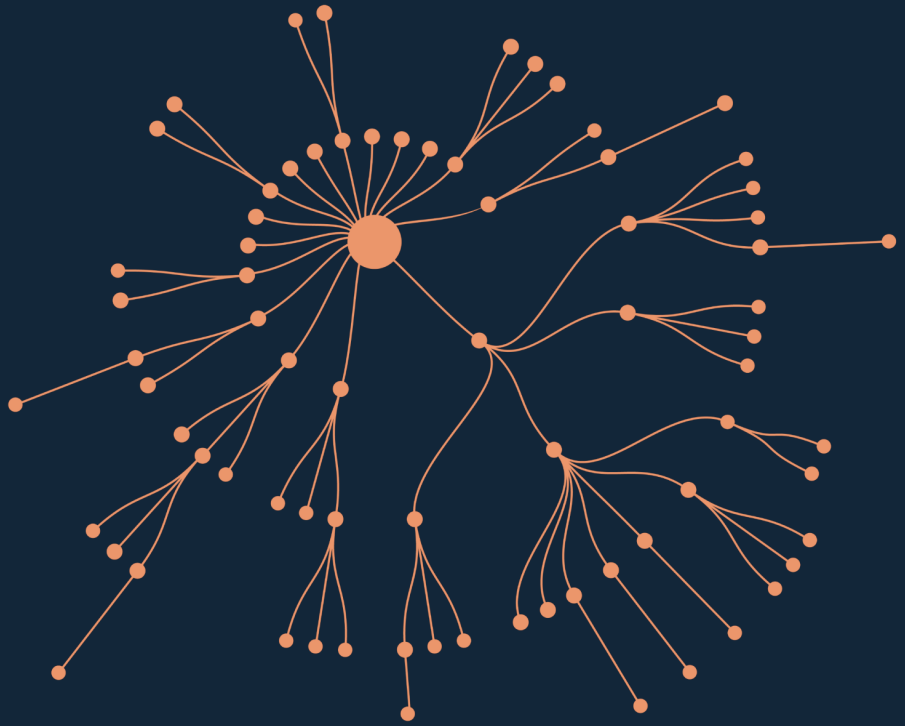


- Detect and report bloated dependencies:
  - In the context of an artifact.
  - On the whole dependency tree.
- Automatic generation of a debloated *pom.xml* file.
- Open source (https://github.com/castor-software/depclean).



RQ: How much dependency bloat exists out there?

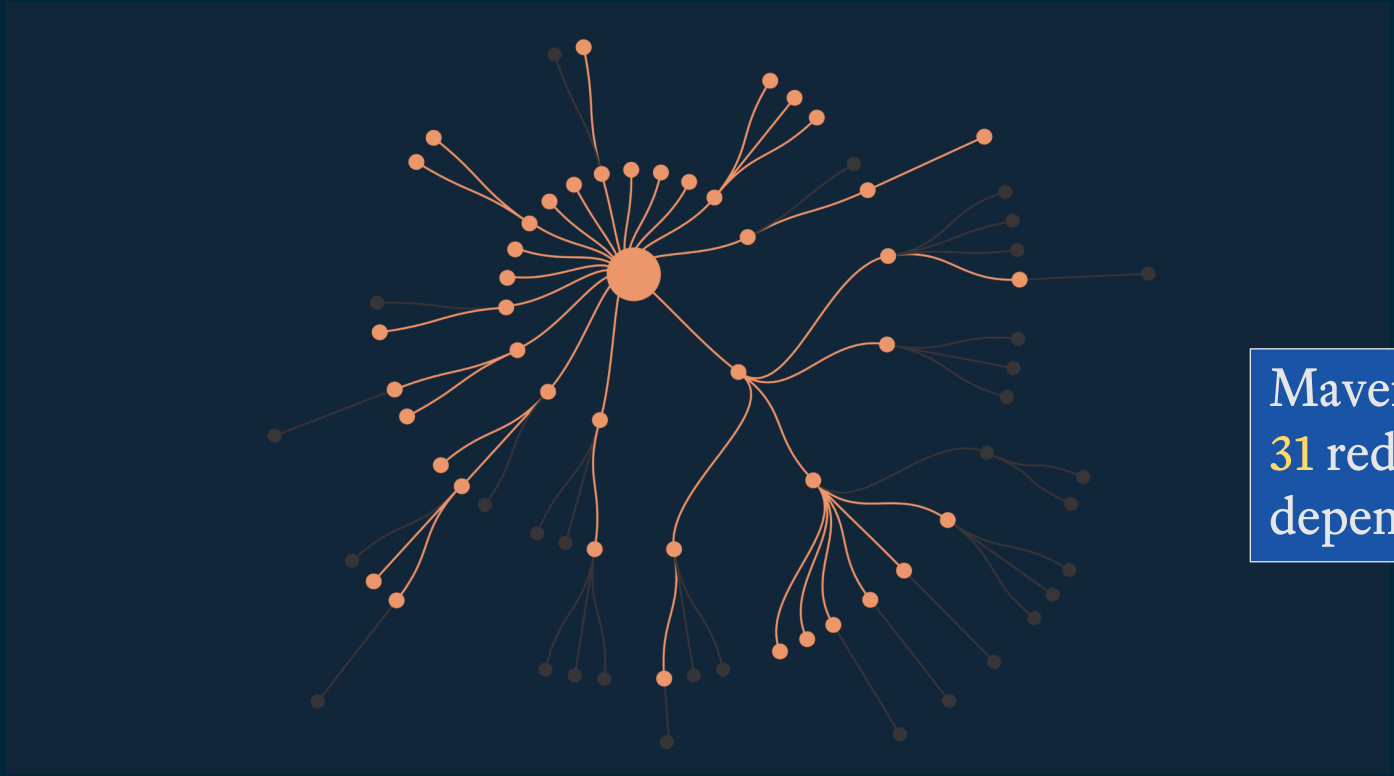
Example: Spoon library



Open source library for code analysis, 75 dependencies.

https://github.com/INRIA/spoon

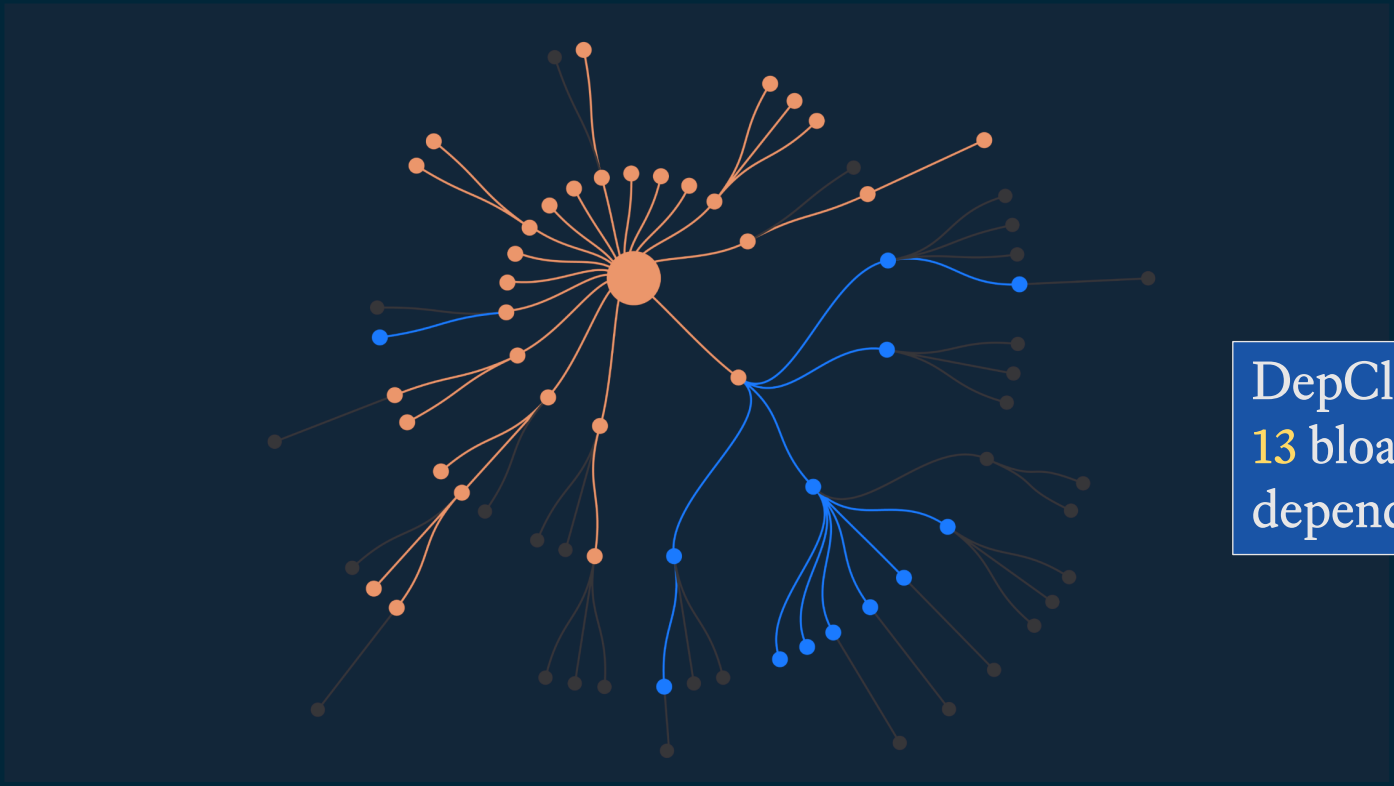
Regular Maven analysis



Maven excludes 31 redundant dependencies.

https://github.com/INRIA/spoon

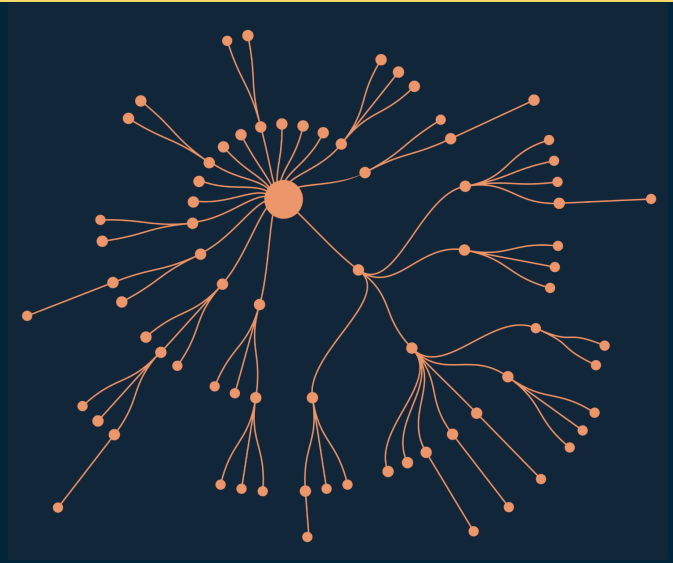
DepClean novel analysis



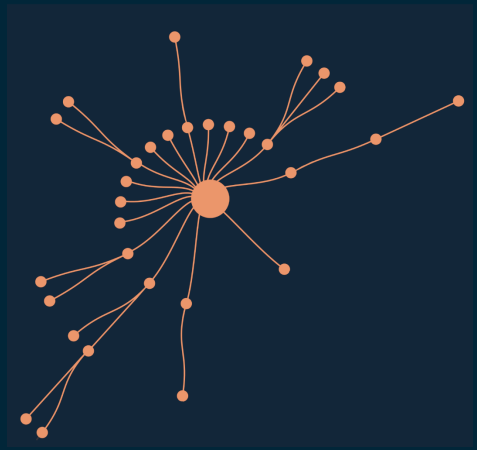
DepClean detects 13 bloated dependencies.

https://github.com/INRIA/spoon

Debloated Spoon library



debloat

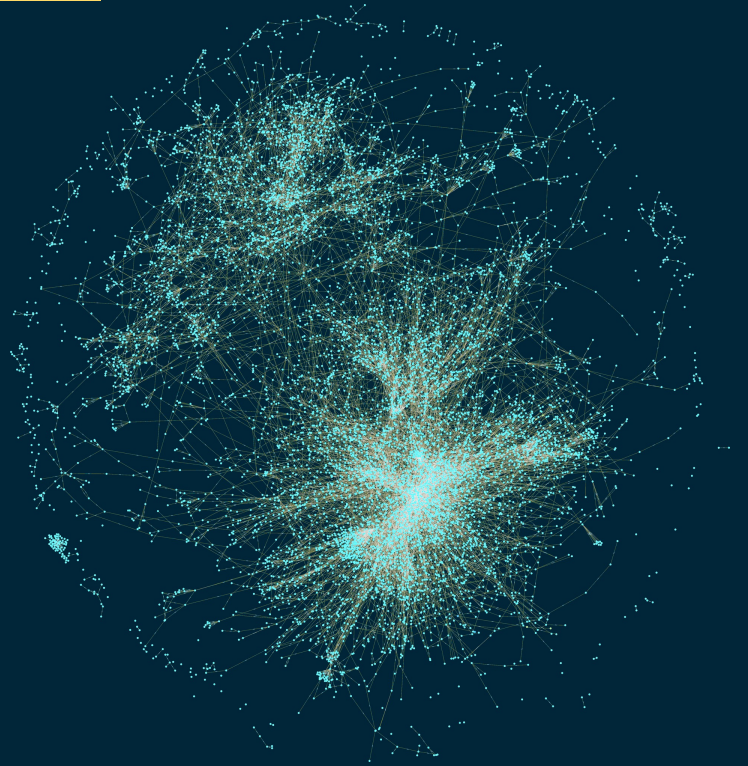


	JAR Size	#Classes
Before	16.2 Mb	7 425
After	12.7 Mb	5 593
Reduction	*28%*	*25%*

https://github.com/INRIA/spoon/pull/3167

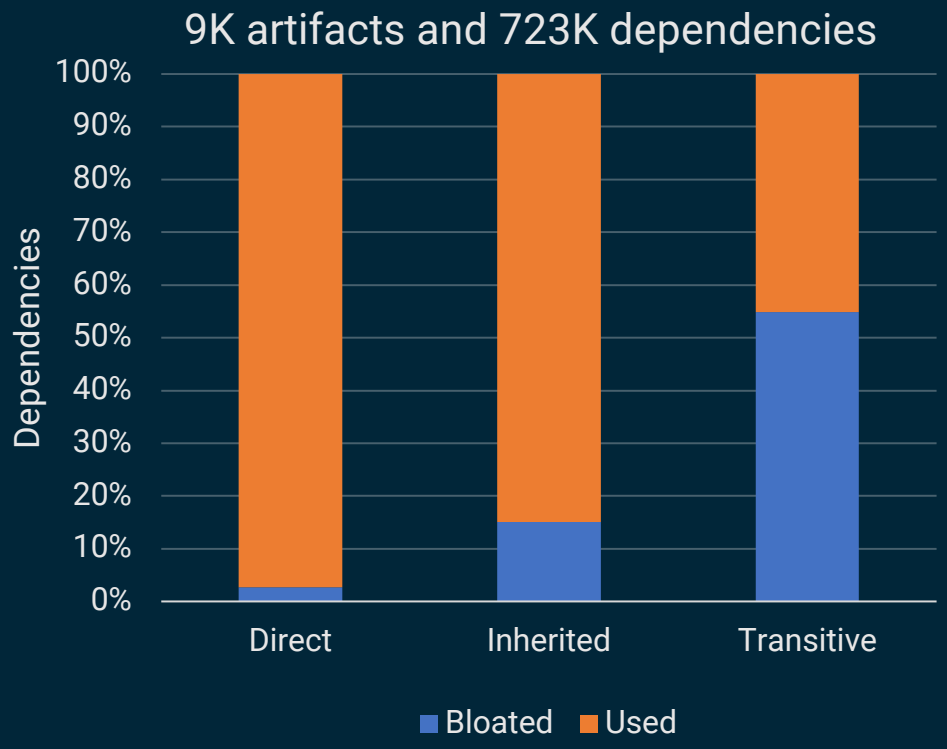
Empirical study (data collection)

- 9K artifacts
  - Diverse
  - Reused
  - Complex
- 723K dependency relationships
  - 45K direct (6%)
  - 180K inherited (25%)
  - 498K transitive (69%)



3.6M artifacts in 2019.

Result

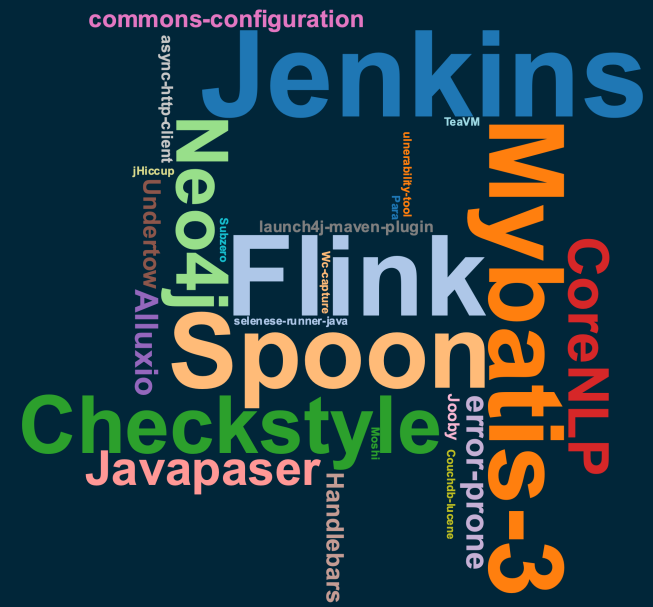


- 2.7% of direct dependencies are bloated
- 15% of inherited dependencies are bloated
- 57% of transitive dependencies are bloated

RQ: Do developers care about bloated dependencies?

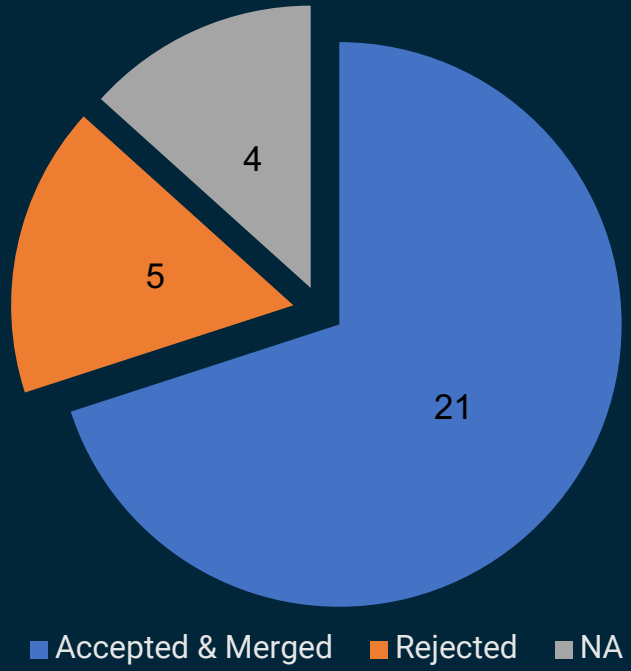
User study

- 30 software projects
  - Open source
  - Active
  - Popular
  - Build succesfuly with Maven
  - Contain dependencies



Result

30 pull requests in 30 notable open-source projects



Removed 140 bloated dependencies in 21 projects thanks to DepClean.

Example: Jenkins

- `jenkins-core`
  - `org.jvnet.hudson:jtidy` (direct)
  - `org.jenkins-ci:constant-pool-scanner` (transitive)
  - `net.i2p.crypto:eddsa` (transitive)
- `jenkins-cli`
  - `commons-codec` (direct)

https://github.com/jenkinsci/jenkins/pull/4378

Developers’ comments

`jenkins-core`

“Past experiences removing unused dependencies have consistently shown that some code will have depended on that inclusion and will be broken by it.”

RQ: Why consider debloating?

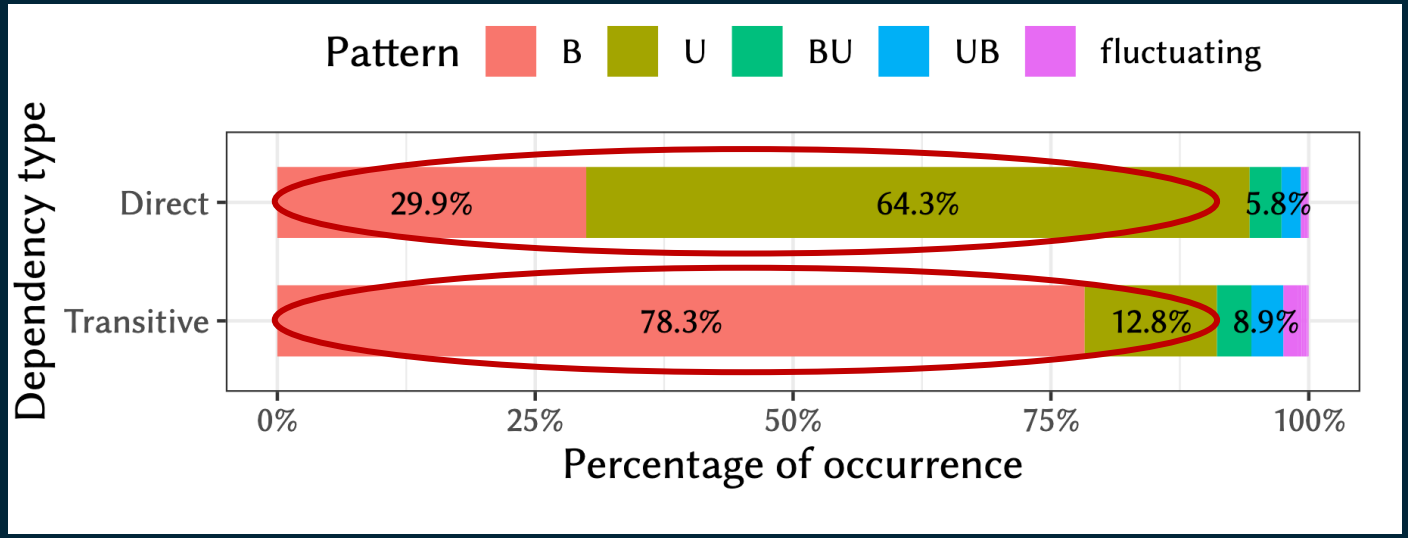
Longitudinal study

- 435 projects
- 31 515 dependency tree versions

- ❑ Do bloated dependencies stay bloated across time?
- ❑ Do developers maintain dependencies that are bloated?

Result

- Do bloated dependencies stay bloated across time?



Result

89% of bloated direct, and 93% of the transitive bloated dependencies remain bloated in all subsequent versions of the studied projects.

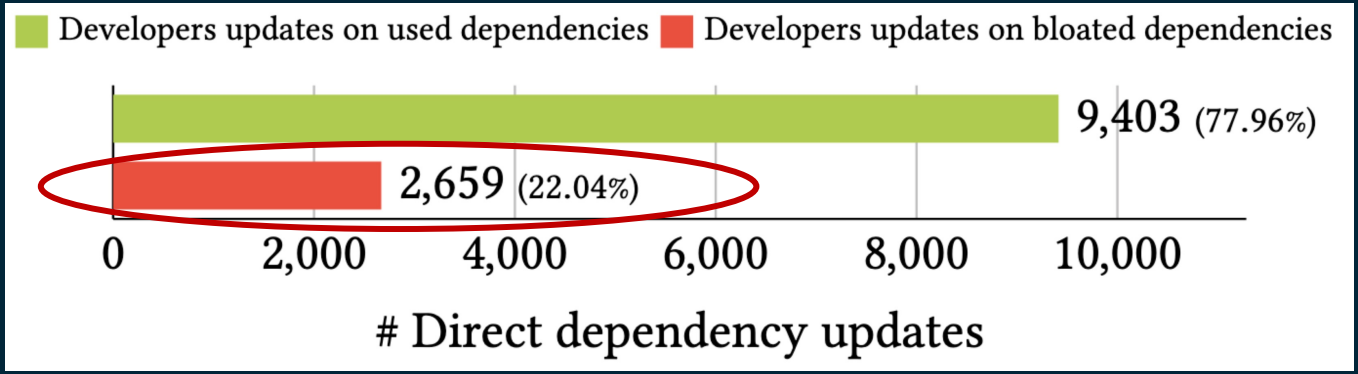
Longitudinal study

- Do developers maintain dependencies that are bloated?

```
118  118        <dependency>
119  119           <groupId>commons-io</groupId>
120  120           <artifactId>commons-io</artifactId>
121       -        <version>2.6</version>
     121  +        <version>2.7</version>
122  122        </dependency>
```

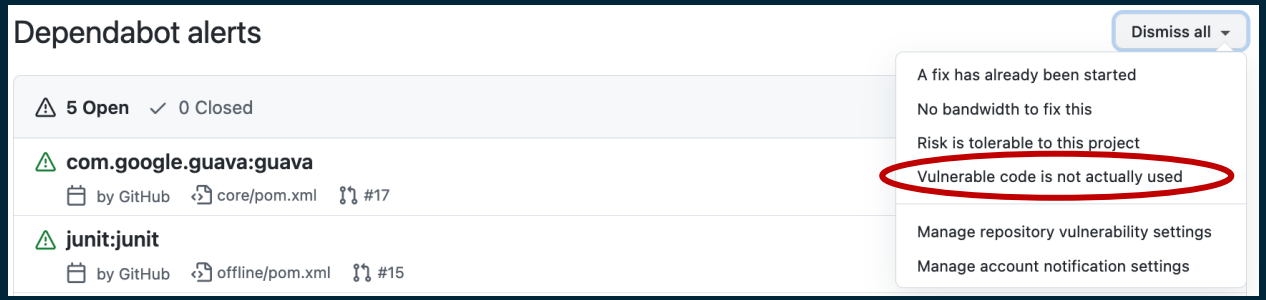
Result

- Do developers maintain dependencies that are bloated?



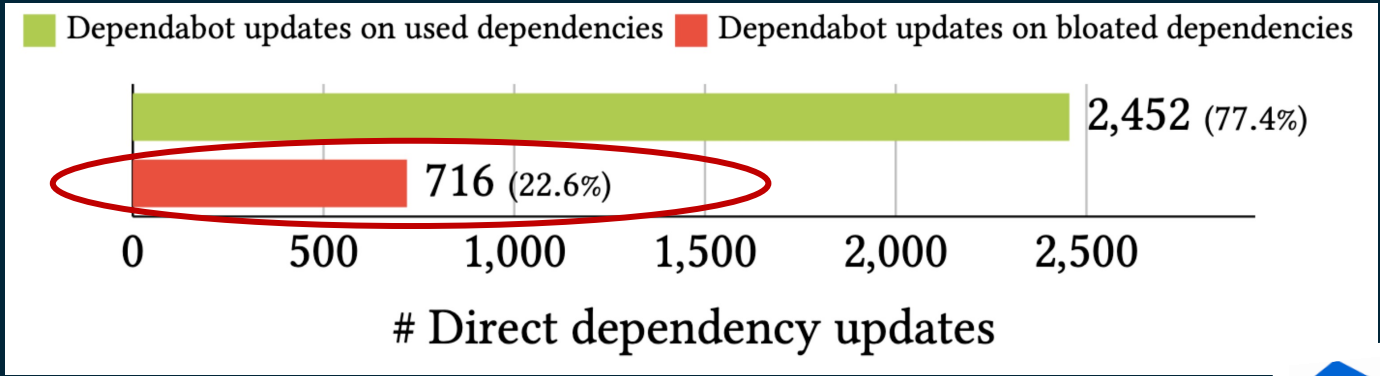
Longitudinal study

- Do developers maintain dependencies that are bloated?



Result

- Do developers maintain dependencies that are bloated?



Dependabot

Result

- Bloated dependencies remain bloated over time.
- Developers maintain dependencies that are bloated.
- Bots suggest maintaining dependencies that are bloated.

Part #3: Debloating Java bytecode

[Dynamic analysis]

Source code


```java
import com.google.common.base.Joiner;
import org.apache.spark.annotation.Private;

@Private
public class EnumUtil {

  public static <E extends Enum<E>> E parseIgnoreCase(Class<E> clz, String str) {
    E[] constants = clz.getEnumConstants();
    if (str == null) {
      return null;
    }
    for (E e : constants) {
      if (e.name().equalsIgnoreCase(str)) {
        return e;
      }
    }
    throw new IllegalArgumentException(
        String.format("Illegal type='%s'. Supported type values: %s",
            str, Joiner.on(", ").join(constants)));
  }
}
```

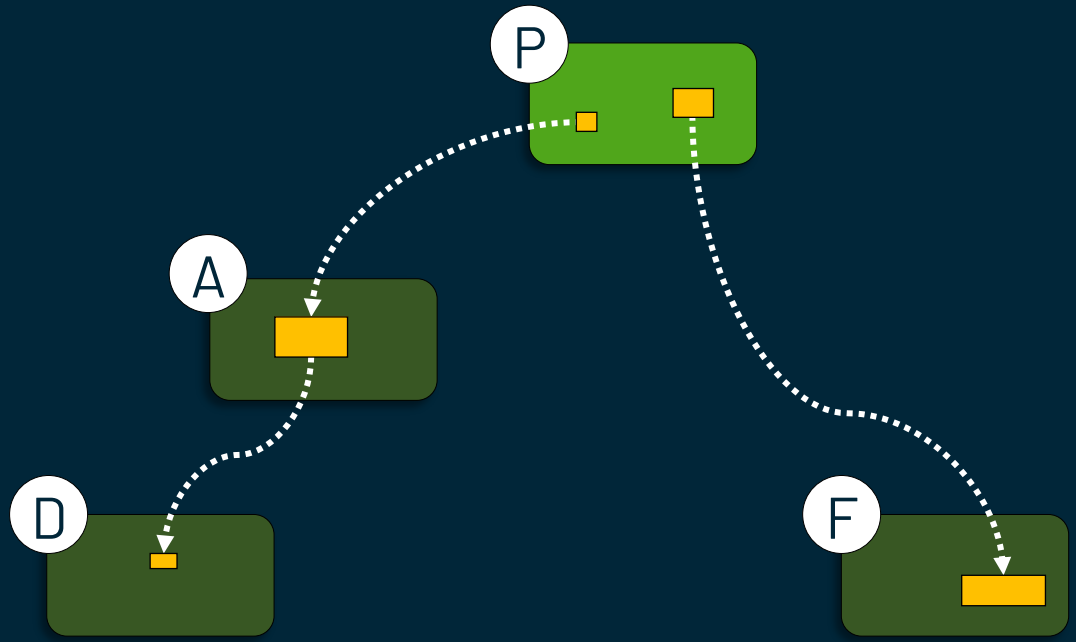
core/src/main/java/org/apache/spark/util/EnumUtil.java

Bytecode

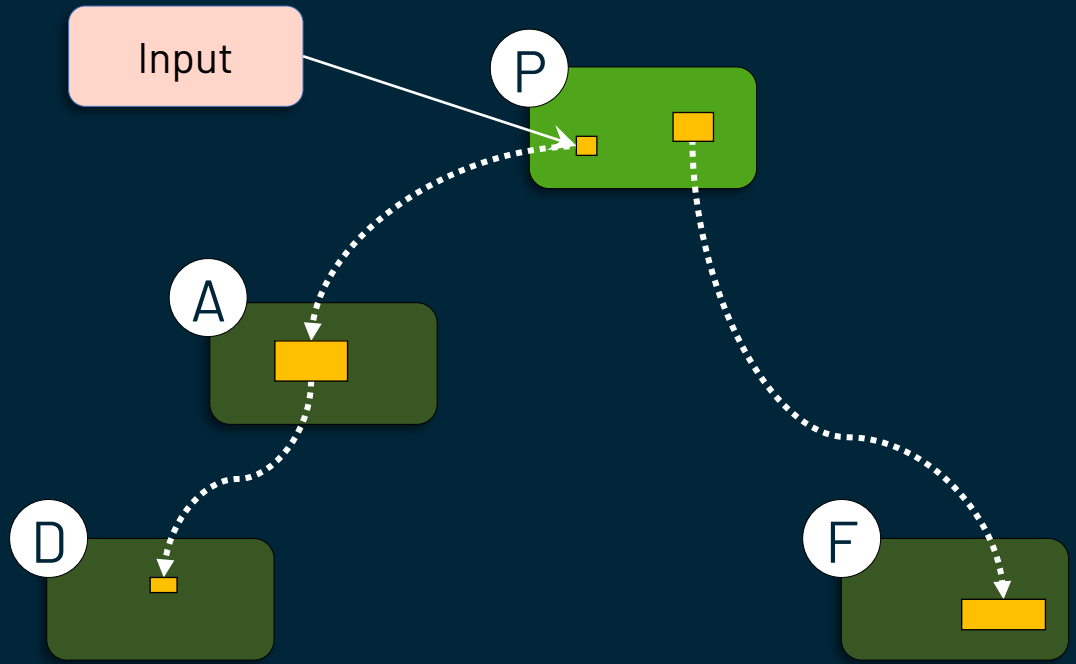

```
// class version 52.0 (52)
// access flags 0x21
public class org/apache/spark/util/EnumUtil {

  // compiled from: EnumUtil.java

  // access flags 0x9
  // signature <E:Ljava/lang/Enum<TE;>;>(Ljava/lang/Class<TE;>;Ljava/lang/String;)TE;
  // declaration: E parseIgnoreCase<E extends java.lang.Enum<E>>(java.lang.Class<E>, java.lang.String)
  public static parseIgnoreCase(Ljava/lang/Class;Ljava/lang/String;)Ljava/lang/Enum;
. . .
L9
    LINENUMBER 35 L9
    INVOKESTATIC com/google/common/base/Joine       ava/lang/String;)Lcom/google/common/base/Joiner;
    ALOAD 2
    INVOKEVIRTUAL com/google/common/base/Joi       ([Ljava/lang/Object;)Ljava/lang/String;
    AASTORE
. . .
```

target/classes/org/apache/spark/util/EnumUtil.class

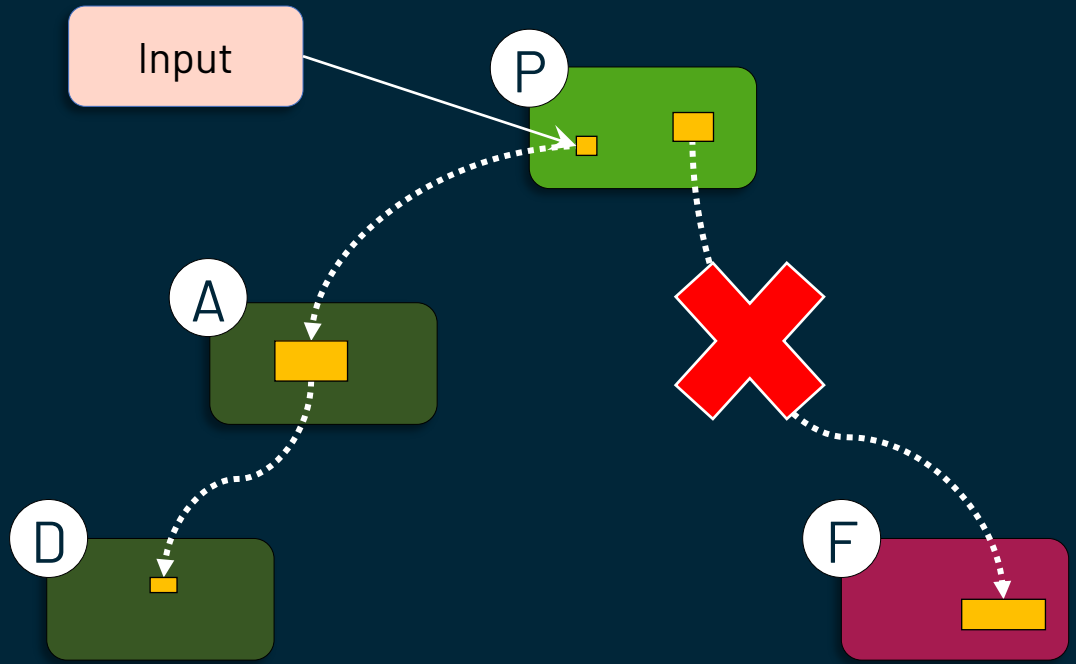
Dynamic analysis



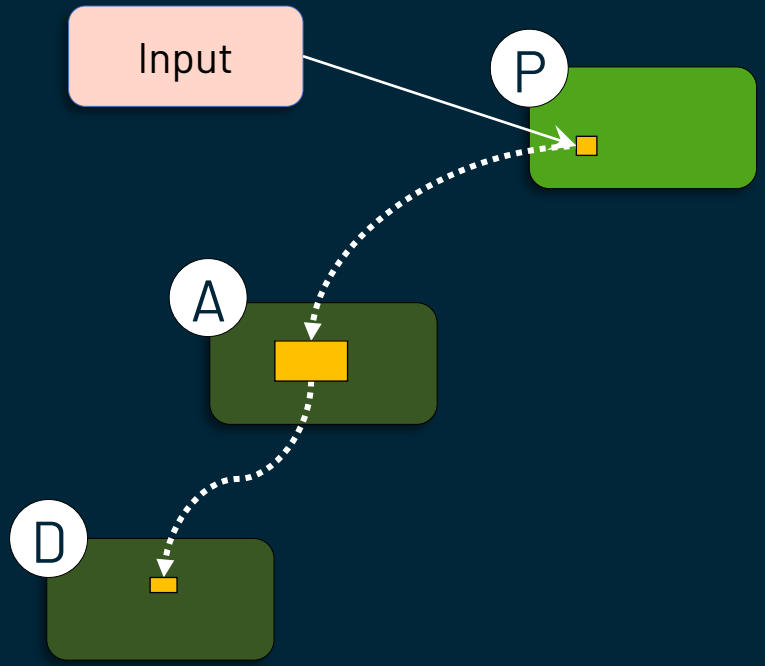
Dynamic analysis



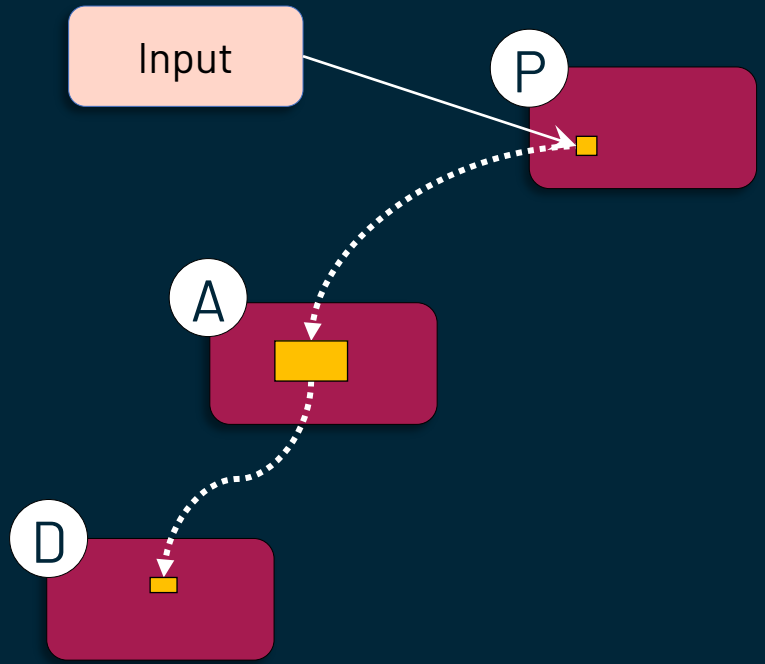
Dynamic analysis



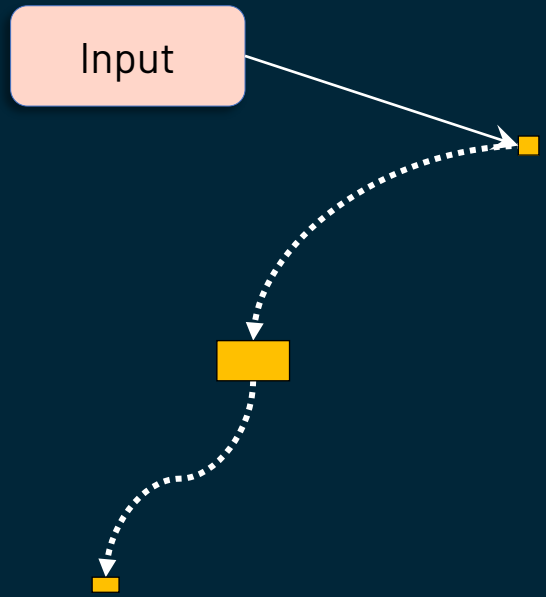
Dynamic analysis



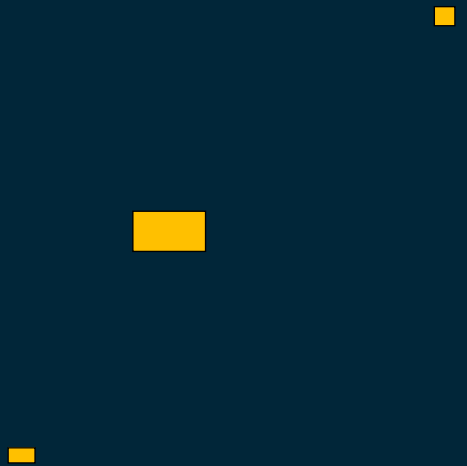
Dynamic analysis



Dynamic analysis

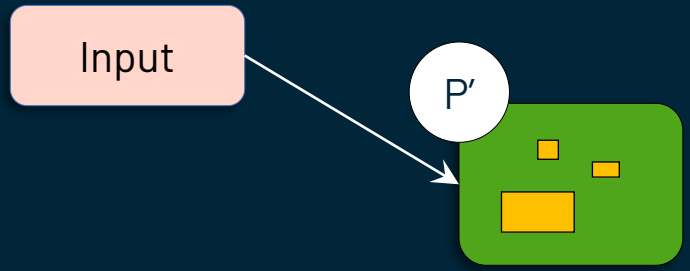


Dynamic analysis



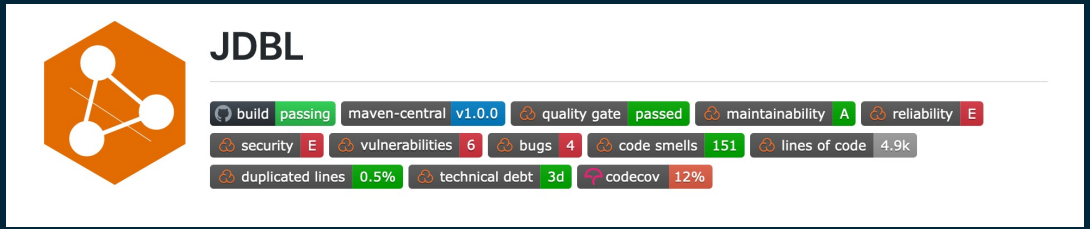
Dynamic analysis

Dynamic analysis

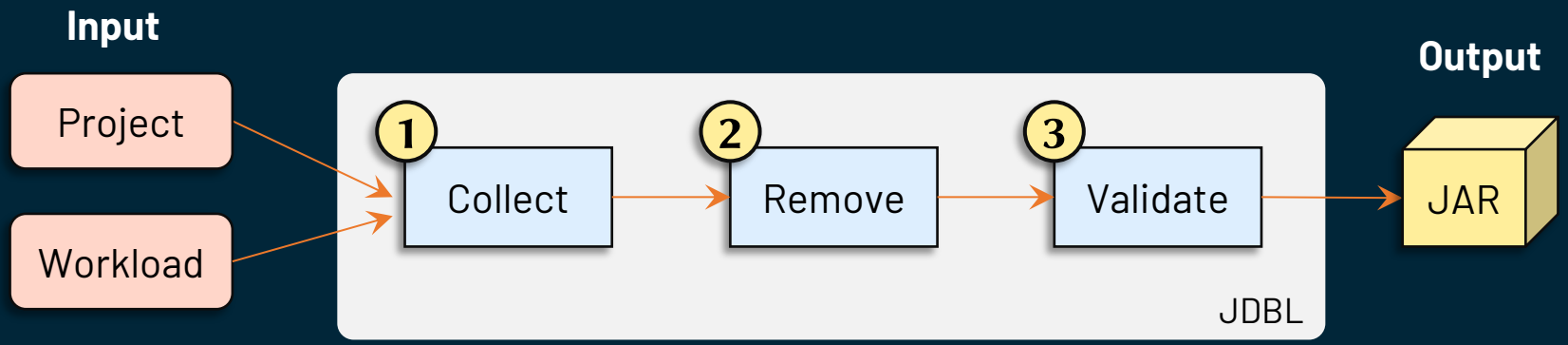


JDBL

- Relies on code-coverage tools to collect bytecode usage information at runtime.
- Automatically removes unused methods, classes, and dependencies in Java projects.
- Open source (https://github.com/castor-software/jdbl)

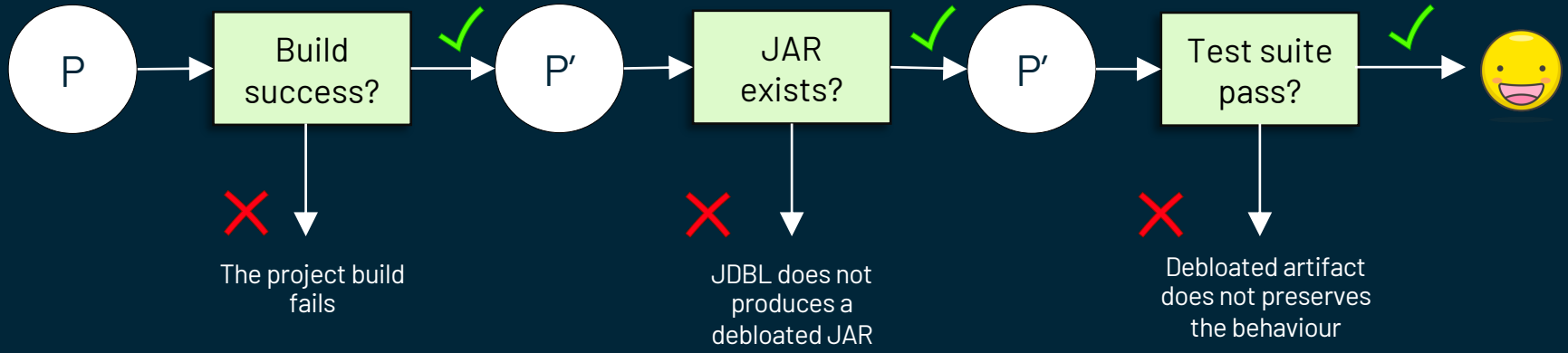


Debloating with JDBL



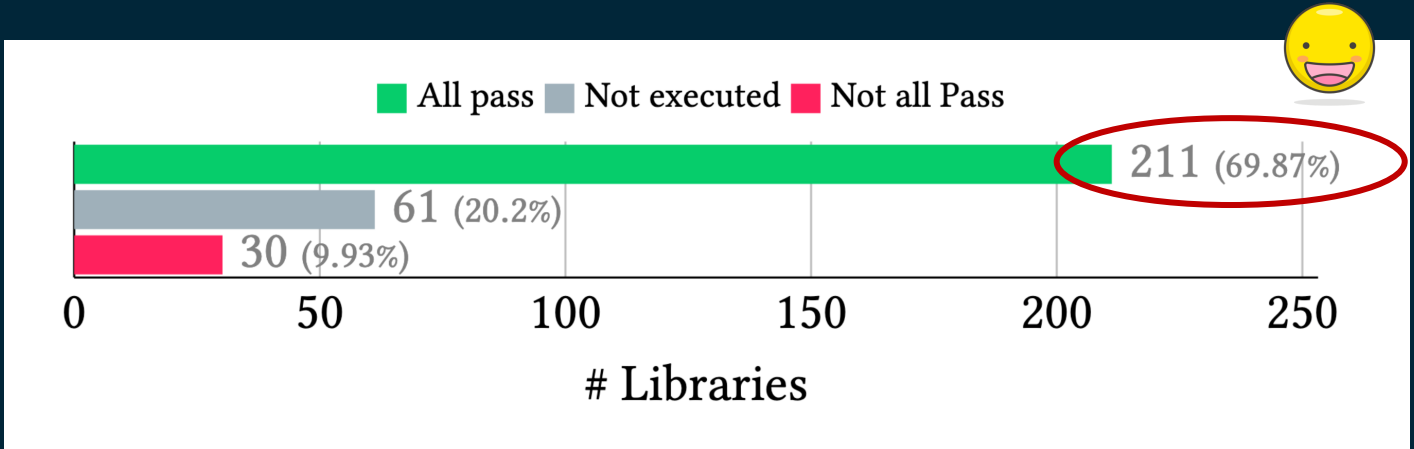
Experiment (debloating libraries)

P = Original library
P′ = Debloated library



https://github.com/castor-software/jdbl-experiments

Result (debloating libraries)



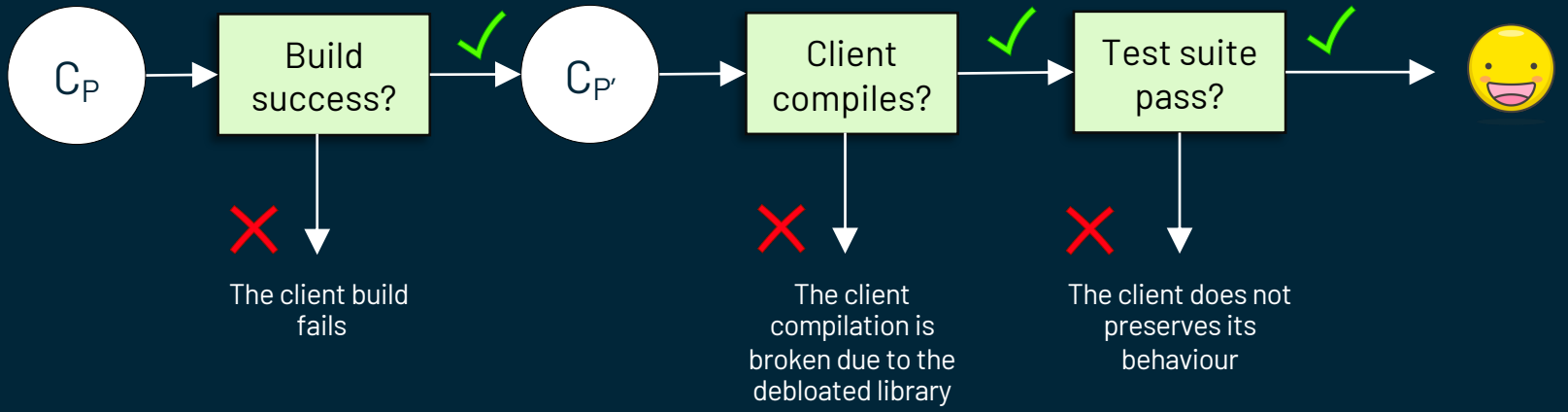
Result (debloating libraries)

	Removed*
Methods	59%
Classes	60%
Dependencies	20%

*211 debloated libraries

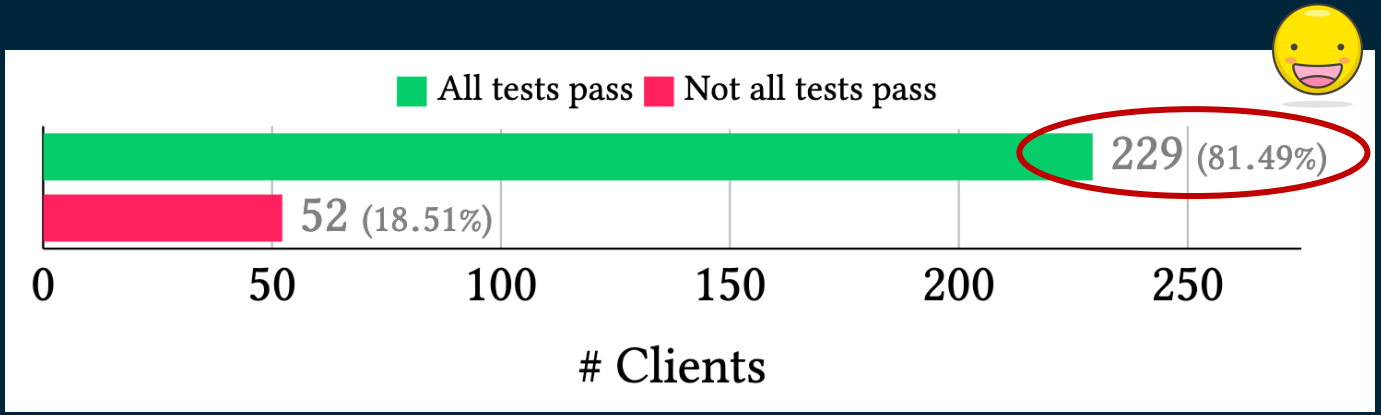
Experiment (impact on clients)

C_P = Client of original library P
$C_{P'}$ = Client of debloated library P'



https://github.com/castor-software/jdbl-experiments

Result (impact on clients)



Part #4: Lessons learned

Lessons learned

- Debloating Java dependencies is a relevant problem.
- Debloating Java bytecode is challenging.
- Debloating real-world applications, automatically, is not the same than debloating hand-picked projects.
- Guaranteeing the safety of the debloating procedure is difficult.

Part #5: Conclusion

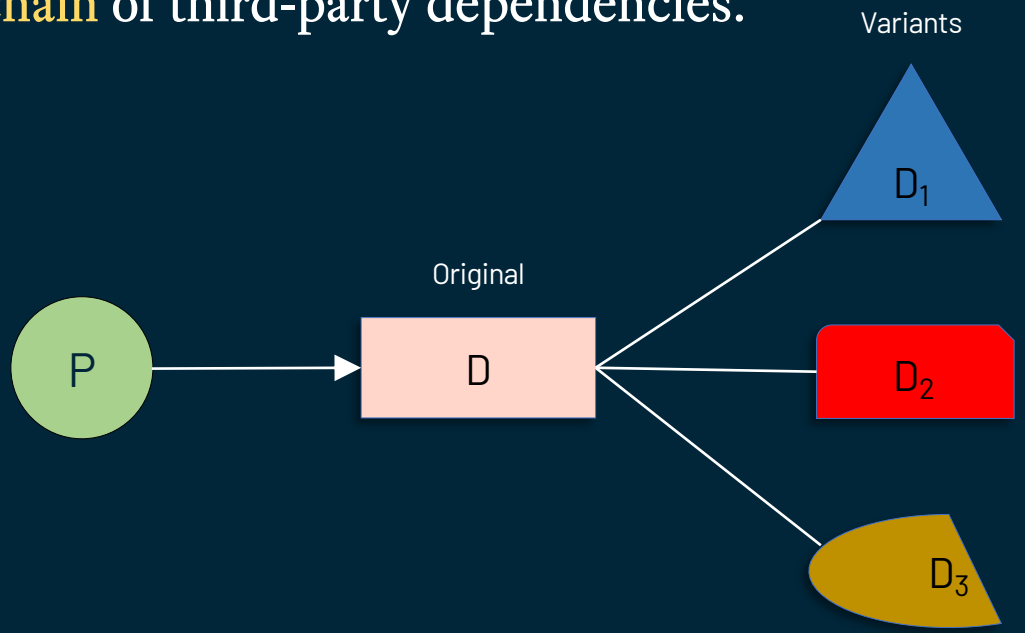
Conclusion

- Maven dependencies are bloated as a consequence of:
  - Transitive dependencies
  - Heritage mechanism of multi-module projects
  - Limited engineering of configuration files (*pom.xml*)
- Software developers care about bloated dependencies.
- Coverage-based debloating is a promising technique that advances the state-of-the-art of Java bytecode debloating.

Part #6: Future work

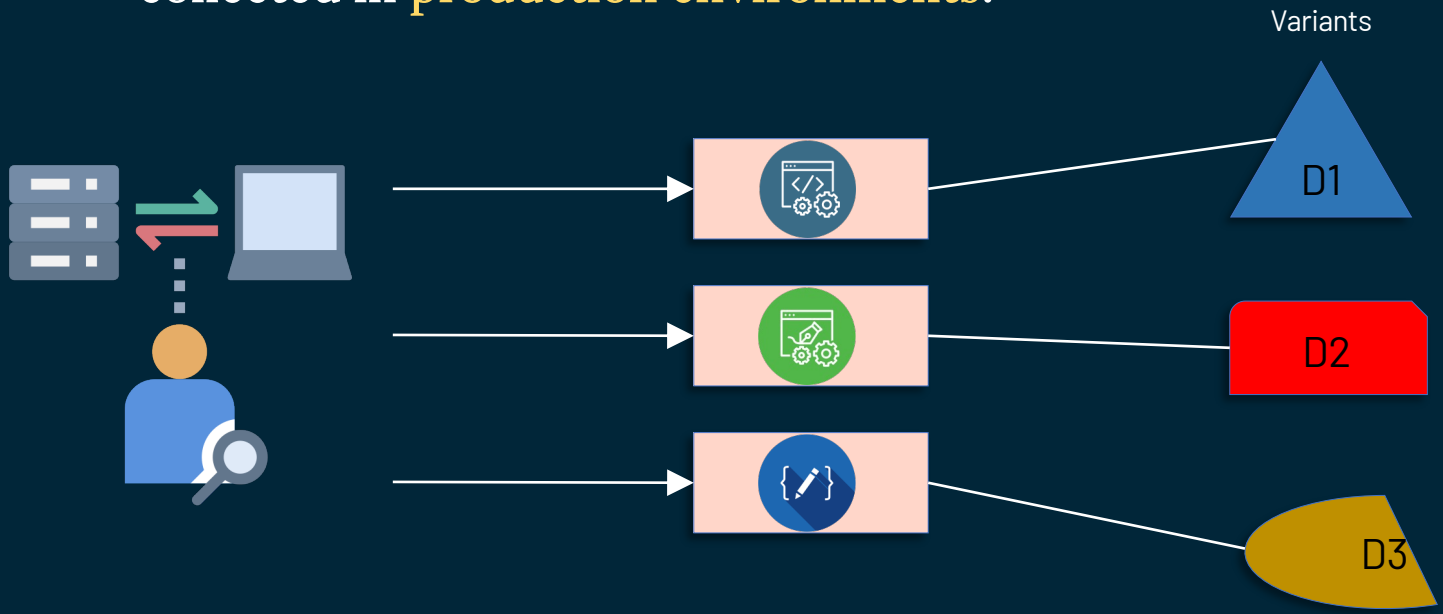
Future work

- Using debloating to specialize and diversify the software supply chain of third-party dependencies.



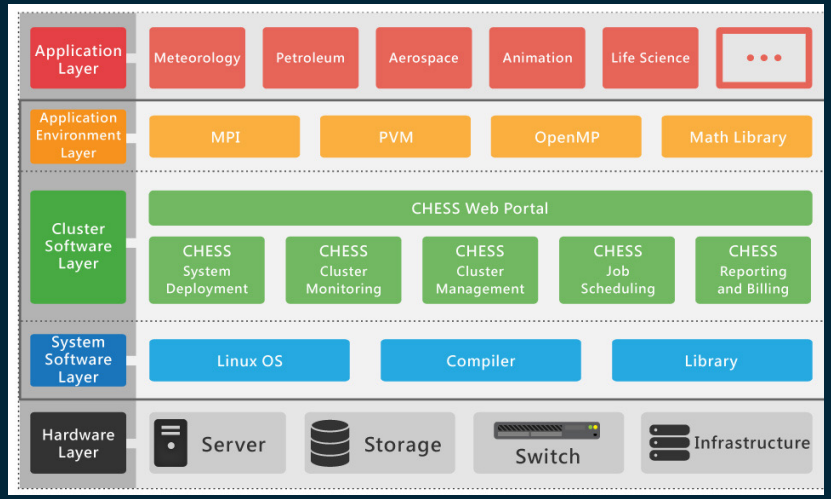
Future work

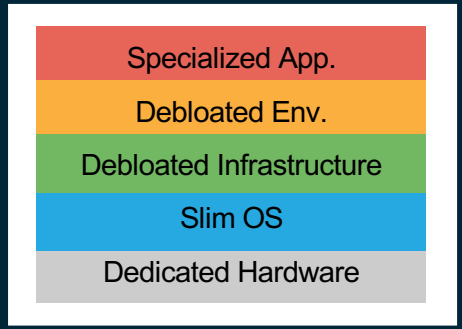
- Debloating applications with respect to usage profiles collected in production environments.



Future work

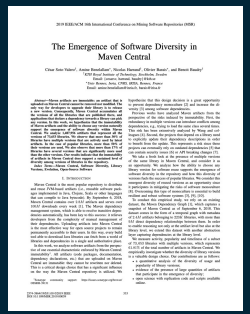
- Extending the debloating techniques to cover other layers of the software stack.

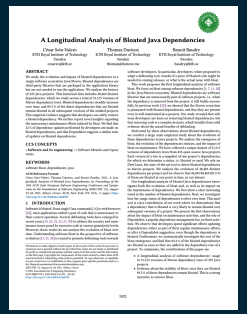


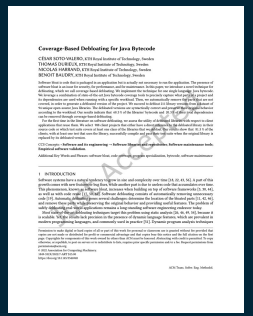


Publications

- The Multibillion Dollar Software Supply Chain of Ethereum. *IEEE Computer*, 2022.
- * Coverage-Based Debloating for Java Bytecode. *TOSEM*, 2022.
- * A Comprehensive Study of Bloated Dependencies in the Maven Ecosystem. *EMSE*, 2021.
- * A Longitudinal Analysis of Bloated Java Dependencies. *ESEC/FSE*, 2021.
- The Emergence of Software Diversity in Maven Central. *MSR*, 2019.







cesarsv@kth.se