Mining Build Changes to Automatically Repair Build Breakage

Christian Macho

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

Shane McIntosh

McGill

Martin Pinzger

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

@Mitschiiii

christian.macho@aau.at

Motivation

Motivation

Motivation

Motivation

→Teams gather immediate feedback on changes

→ Improve productivity

B. Vasilescu et al., "Quality and productivity outcomes relating to continuous integration in github", ESEC/FSE 2015.


“there’s
no such thing
as a free
lunch.”

Motivation

- CI Trade-offs
  - Increase of maintenance
- Includes maintenance of build specifications
- Neglected maintenance

  → Build breakage

M. Hilton et al., "Trade-Offs in Continuous Integration: Assurance, Security, and Flexibility“, ESEC/FSE 2017
B. Adams et al., "The evolution of the linux build system", *EASST* 2007
H. Seo et al., "Programmers’ build errors: a case study (at google)", ICSE 2014

Motivation

- Developers need to fix the breakage
  → blocked
- Large SW company
  – 900 man hours fixing build breakage

N. Kerzazi et al., "Why do automated builds break? an empirical study“, ICSME 2014

Motivation

- Build Breakage
  - Tests
  - Compiling
  - Dependencies
- Reports of dependency-related breakage
  - 39% - 65%

H. Seo et al., "Programmers' build errors: a case study (at google)", ICSE 2014
M. Sulir et al., "A quantitative study of Java software buildability", *PLATEAU 2016*,
M. Tufano et al., "There and back again: Can you compile that snapshot? ", JSS vol 29/4, 2017

How to repair?

Research Questions

Research Questions

(RQ1)
Strategies?

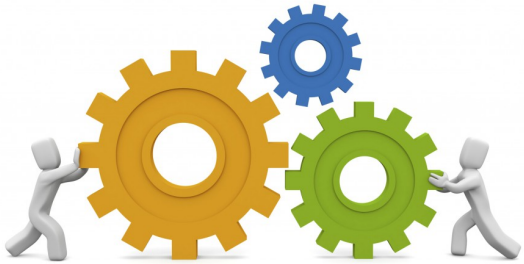
How do developers repair dependency-related build breakage?

Research Questions

(RQ1)
Strategies?

How do developers repair dependency-related build breakage?

(RQ2)
Automation?



To what extent can we automatically repair dependency-related build breakage?

Data Preparation – Projects

- GitHub projects (top-1000 stars)
  - Maven
  - >500 commits
  - actively developed
  - build without manual setup/intervention

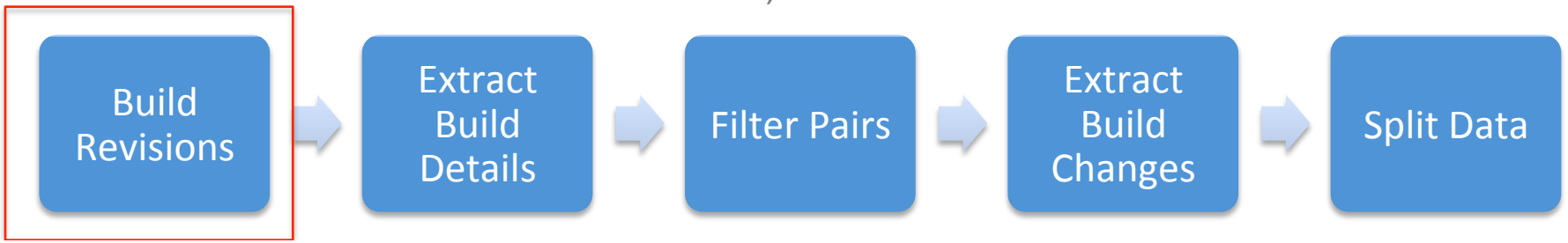
- →23 projects
  - different sizes, vendors, and purposes

Data Preparation – Process

Data Preparation – Build Revisions

- December 2014 > commit date > July 2017
- Mitigate ecosystem-related build failures
- But TravisTorrent?
- Yes – but
  - Build results can be unreliable
  - Depend on the environment
- → Build in our environment

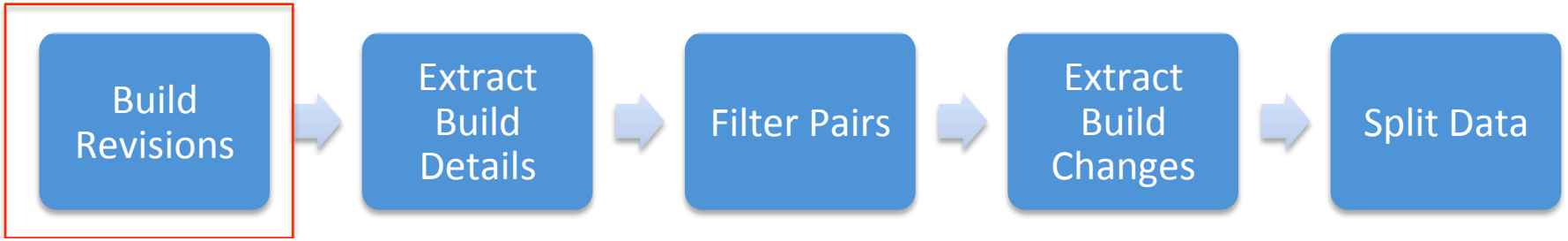
M. Zolfagharinia et al., "Do not trust build results at face value: an empirical study of 30 million CPAN builds", MSR 2017



Data Preparation – Build Revisions

- `mvn -U clean package -DskipTest=true`
  - Force check for update
  - Ignore tests → focus on build errors

```
[INFO] hazelcast-jca-rar ................................ SKIPPED
[INFO] hazelcast-build-utils ............................ SKIPPED
[INFO] ------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------
[INFO] Total time: 11.363 s
[INFO] Finished at: 2017-07-07T03:29:37+02:00
[INFO] Final Memory: 41M/1441M
[INFO] ------------------------------------------------------------
```



Data Preparation – Build Details

- MavenLogAnalyzer (MLA)
- Build result taxonomy

SUCCESS
DEPENDENCY_RESOLUTION_FAILED
TEST_EXECUTION_FAILED
COMPILATION_FAILED

Data Preparation – Build Details

- MavenLogAnalyzer (MLA)
- Build result taxonomy

SUCCESS

DEPENDENCY_RESOLUTION_FAILED

TEST_EXECUTION_FAILED

COMPILATION_FAILED

Data Preparation – Filter Pairs

Old Commit ID	Old Build Result	New Commit ID	New Build Result
a34b2ad	DEP_FAILED	ef8ad8c	SUCCESS
…	TEST_FAILED	…	TEST_FAILED
…	DEP_FAILED	…	DEP_FAILED
…	SUCCESS	…	DEP_FAILED
…	DEP_FAILED	…	SUCCESS

Data Preparation – Filter Pairs

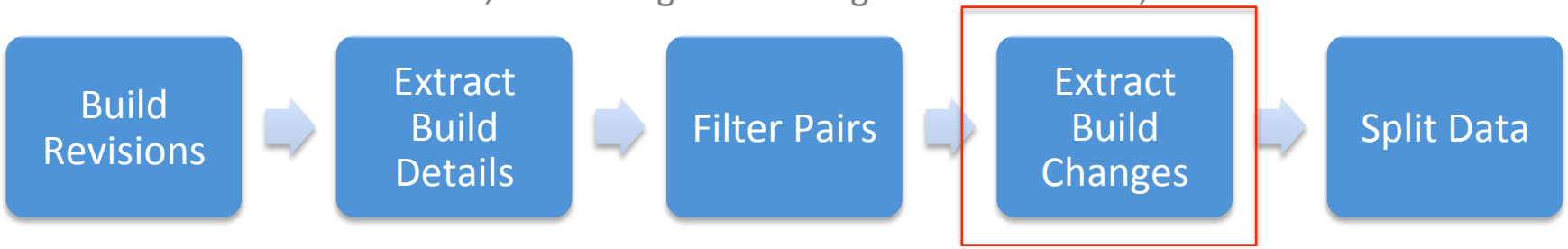
Old Commit ID	Old Build Result	New Commit ID	New Build Result
a34b2ad	**DEP_FAILED**	**ef8ad8c**	**SUCCESS**
…	TEST_FAILED	…	TEST_FAILED
…	DEP_FAILED	…	DEP_FAILED
…	SUCCESS	…	DEP_FAILED
…	**DEP_FAILED**	**…**	**SUCCESS**

→ Filter repairing commits

Data Preparation – Build Changes

- BuildDiff
  - Transform `pom.xml` into two trees
  - Extend GumTree algorithm
    - Only match nodes of same type (e.g. `dependency`)
    - Maven triplet (`groupId`, `artifactId`, `version`)
    - Use `Id` tag
    - Levenshtein similarity > *t* (best: *t = 0.65*)
  - Output: edit operations (add/del/upd/mv)

J.-R. Falleri et al., "Fine-grained and accurate source code differencing", ASE 2014
C. Macho et al., "Extracting Build Changes with BuildDiff", *MSR* 2017



Data Preparation – Build Changes

- Map changes
  - Edit operation → Build Change and Build Change Category (Taxonomy, available online)

Taxonomy - Example

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.2.5.RELEASE</version>
</dependency>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.2.6.RELEASE</version>
</dependency>
```

Taxonomy - Example

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.2.5.RELEASE</version>
</dependency>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.2.6.RELEASE</version>
</dependency>
```

Taxonomy - Example

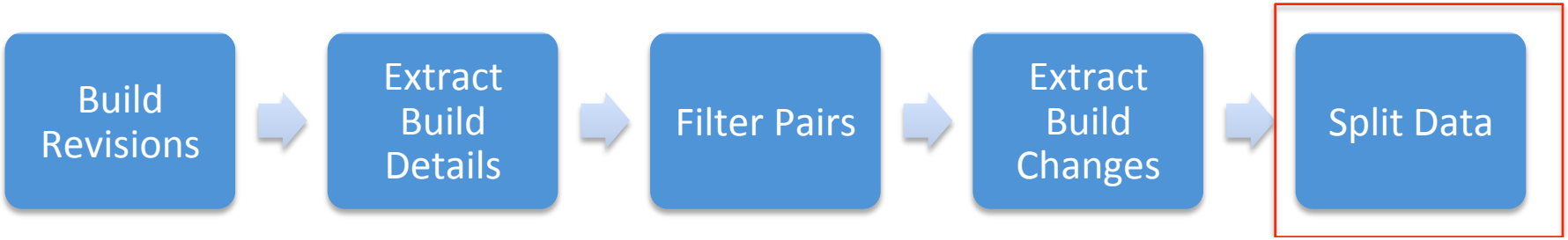
```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.2.5.RELEASE</version>
</dependency>
```

DEPENDENCY_VERSION_UPDATE

```
    <version>4.2.6.RELEASE</version>
</dependency>
```

Data Preparation – Build Changes

- 2 research questions
  - 30/70 data split
  - 37 pairs (RQ1) / 88 pairs (RQ2)



Research Questions

(RQ1)
Strategies?

How do developers repair dependency-related build breakage?

(RQ2)
Automation?

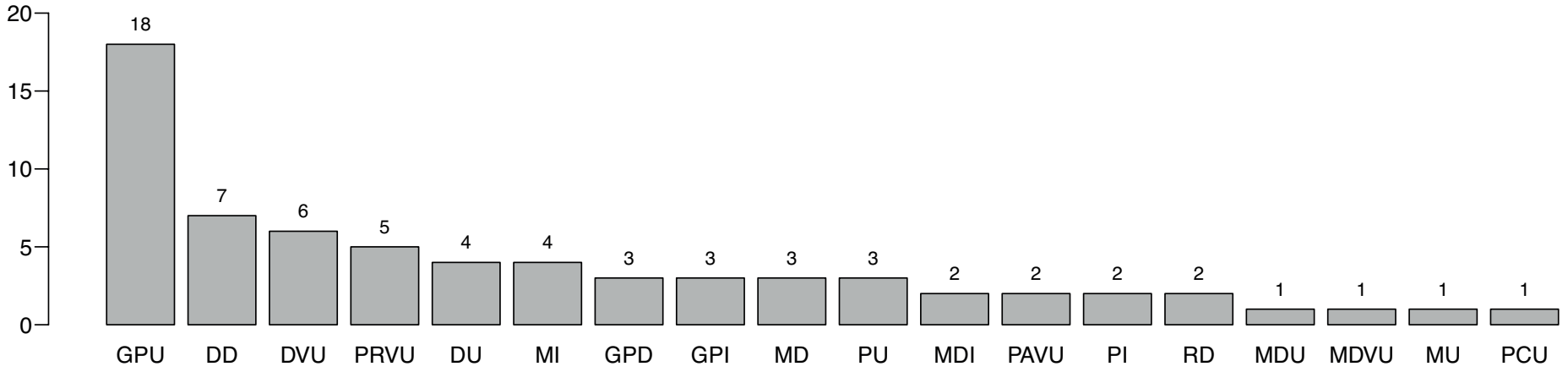
To what extent can we automatically repair dependency-related build breakage?

Approach

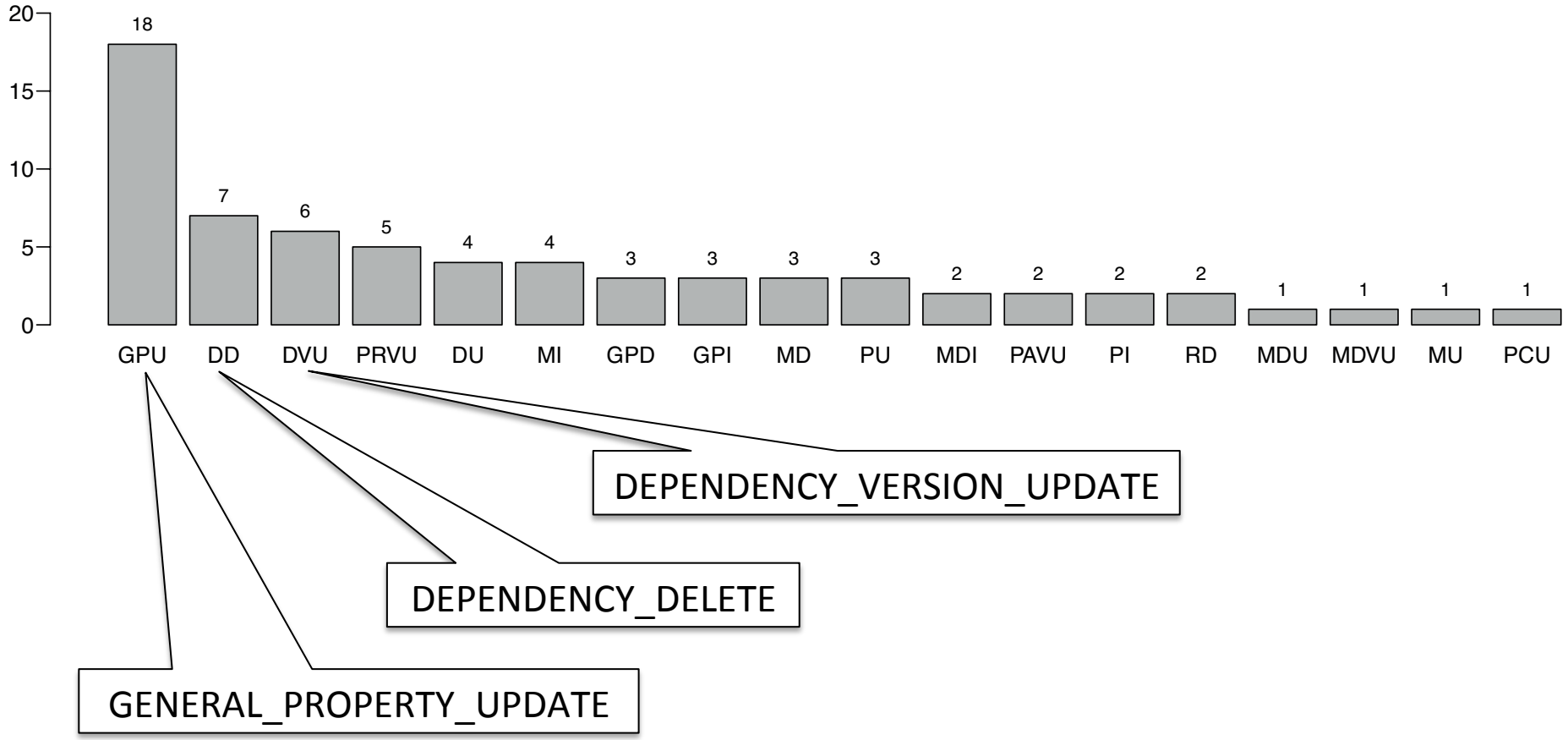
- Quantitative Analysis
  - Frequency of build changes (categories) involved in repairing pairs

    → Number of revisions
  - Categories according to the purpose of the change (e.g., property change)

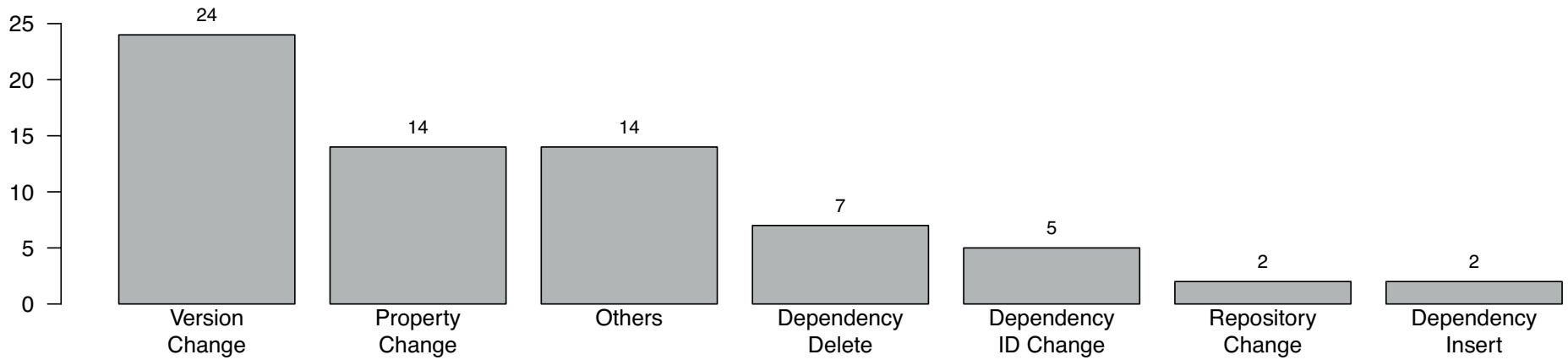
Results


20
15
10
5
0
18
7
6
5
4
4
3
3
3
3
2
2
2
2
1
1
1
1
GPU
DD
DVU
PRVU
DU
MI
GPD
GPI
MD
PU
MDI
PAVU
PI
RD
MDU
MDVU
MU
PCU

Results


20
15
10
5
0
18
7
6
5
4
4
3
3
3
3
2
2
2
2
1
1
1
1
GPU
DD
DVU
PRVU
DU
MI
GPD
GPI
MD
PU
MDI
PAVU
PI
RD
MDU
MDVU
MU
PCU
DEPENDENCY_VERSION_UPDATE
DEPENDENCY_DELETE
GENERAL_PROPERTY_UPDATE

Results

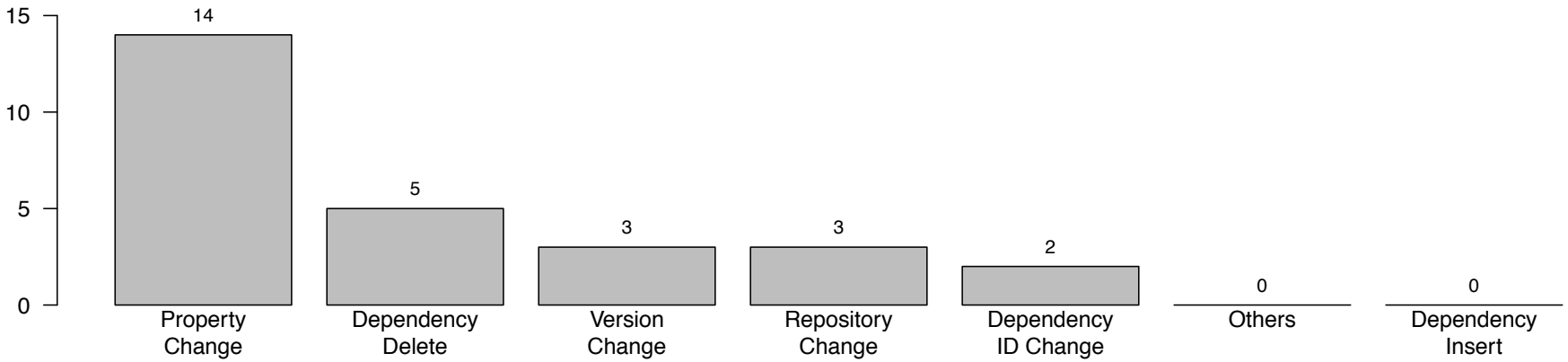

25
20
15
10
5
0
24
14
14
7
5
2
2
Version Change
Property Change
Others
Dependency Delete
Dependency ID Change
Repository Change
Dependency Insert

Approach

- Qualitative Analysis
  - Build changes (categories) actually repaired
  - Manually analyze the change(s) that repaired

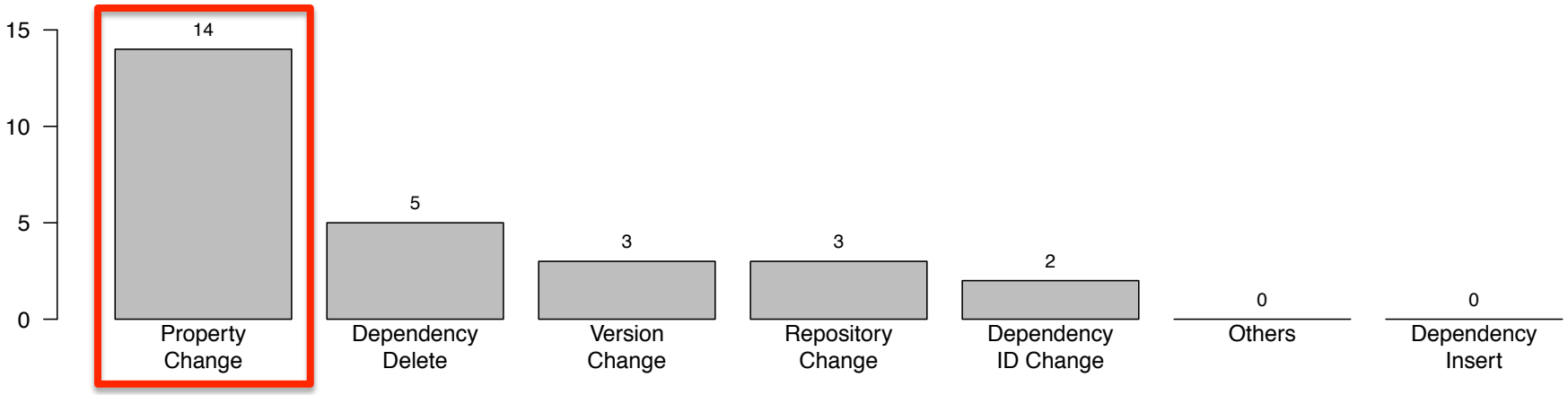
Results

- 27/37 revisions repaired with single change



Results

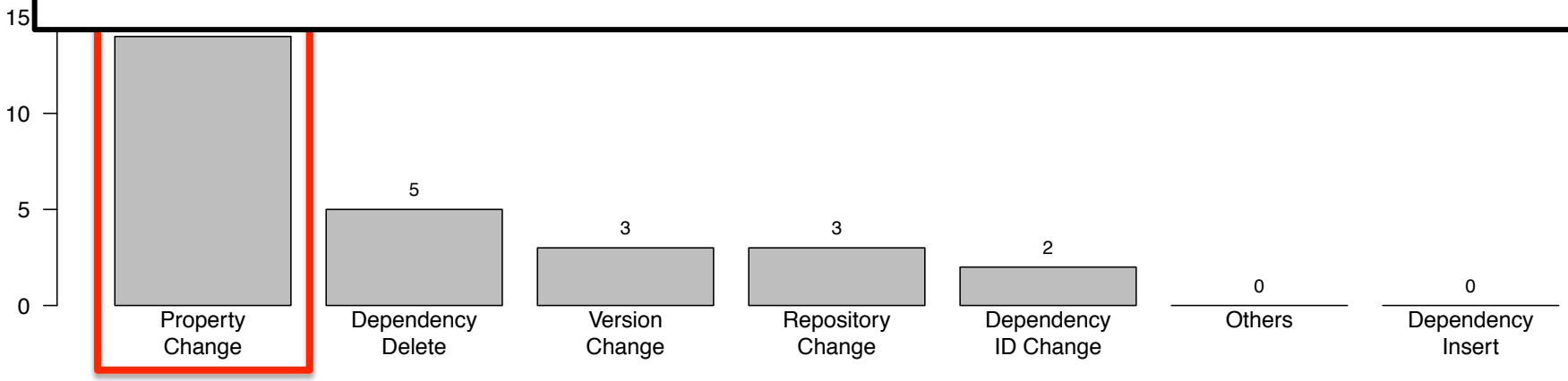
- 27/37 revisions repaired with single change
- Property Change?
  - Refer to version changes



Results

- 27/37 revisions repaired with single change

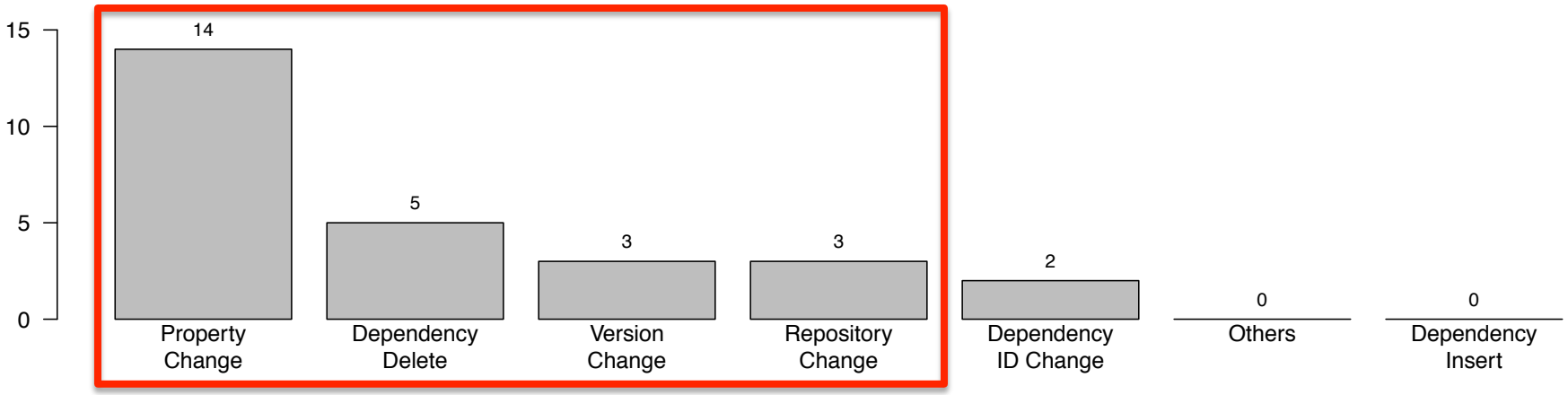
→ Repair strategies!



Results

- 27/37 revisions repaired with single change

- Property Change?
  - Refer to version changes



Strategies

- Version Update
  - Identify failing dependency
  - Remove “-SNAPSHOT“
  - Update version (MAJOR.MINOR.PATCH)

$$distance = abs(10000 * (V1_{maj} - V2_{maj}) + 100 * (V1_{min} - V2_{min}) + (V1_{pat} - V2_{pat}))$$

- Dependency Delete
- Add Repository

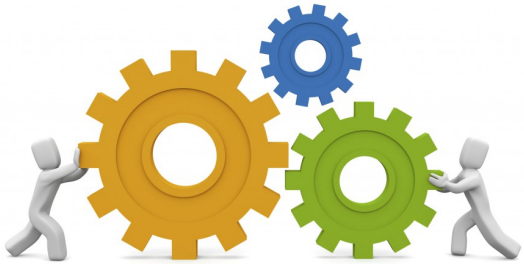
S. Raemaekers et al., “Semantic Versioning versus Breaking Changes : A Study of the Maven Repository” *SCAM* 2014

Research Questions

(RQ1)
Strategies?

How do developers repair dependency-related build breakage?

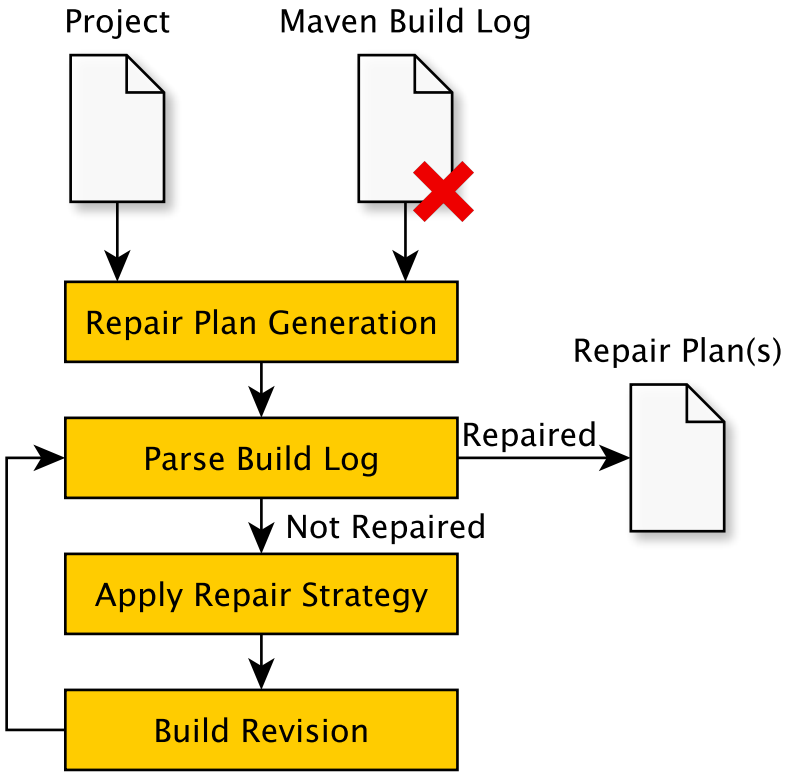
(RQ2)
Automation?



To what extent can we automatically repair dependency-related build breakage?

Approach

BuildMedic



Results

Project	Fixed	Not Fixed	n=1	ID	SIM
async-http-client	1 (100%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
immutables	1 (100%)	0 (0%)	1 (100%)	1 (100%)	0 (0%)
closure-compiler	1 (100%)	0 (0%)	1 (100%)	1 (100%)	0 (0%)
symphony	4 (57%)	3 (43%)	4 (100%)	2 (50%)	2 (50%)
wildfly	0 (0%)	1 (100%)	- (-)	- (-)	- (-)
YCSB	1 (20%)	4 (80%)	1 (100%)	0 (0%)	1 (100%)
alluxio	10 (37%)	17 (63%)	3 (30%)	1 (10%)	7 (70%)
libgdx	5 (100%)	0 (0%)	5 (100%)	1 (20%)	3 (60%)
hazelcast	7 (50%)	7 (50%)	7 (100%)	7 (100%)	0 (0%)
...					
Total	**45 (54%)**	**39 (46%)**	**34 (76%)**	**16 (36%)**	**20 (44%)**

Results

Project	Fixed	Not Fixed	n=1	ID	SIM
async-http-client	1 (100%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
immutables	1 (100%)	0 (0%)	1 (100%)	1 (100%)	0 (0%)
closure-compiler	1 (100%)	0 (0%)	1 (100%)	1 (100%)	0 (0%)
symphony	4 (57%)	3 (43%)	4 (100%)	2 (50%)	2 (50%)
wildfly	0 (0%)	1 (100%)	- (-)	- (-)	- (-)
YCSB	1 (20%)	4 (80%)	1 (100%)	0 (0%)	1 (100%)
alluxio	10 (37%)	17 (63%)	3 (30%)	1 (10%)	7 (70%)
libgdx	5 (100%)	0 (0%)	5 (100%)	1 (20%)	3 (60%)
hazelcast	7 (50%)	7 (50%)	7 (100%)	7 (100%)	0 (0%)
…					
Total	**45 (54%)**	**39 (46%)**	**34 (76%)**	**16 (36%)**	**20 (44%)**

Results

Project	Fixed	Not Fixed	n=1	ID	SIM
async-http-client	1 (100%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
immutables	1 (100%)	0 (0%)	1 (100%)	1 (100%)	0 (0%)
closure-compiler	1 (100%)	0 (0%)	1 (100%)	1 (100%)	0 (0%)
symphony	4 (57%)	3 (43%)	4 (100%)	2 (50%)	2 (50%)
wildfly	0 (0%)	1 (100%)	- (-)	- (-)	- (-)
YCSB	1 (20%)	4 (80%)	1 (100%)	0 (0%)	1 (100%)
alluxio	10 (37%)	17 (63%)	3 (30%)	1 (10%)	7 (70%)
libgdx	5 (100%)	0 (0%)	5 (100%)	1 (20%)	3 (60%)
hazelcast	7 (50%)	7 (50%)	7 (100%)	7 (100%)	0 (0%)
...					
Total	**45 (54%)**	**39 (46%)**	**34 (76%)**	**16 (36%)**	**20 (44%)**

Performance?

Performance

Total time: 4 – 61 minutes (22.8 average)

Overhead: 1.5 – 35 minutes (8.6 average)

Applications/Implications

- Build Breakage can often be repaired with a single change
- Version Update most frequent change
- Build Medic can support developers
  - Post build action
  - Standalone tool

Motivation

Jenkins

Travis CI

→Teams gather immediate feedback on changes
→ Improve productivity

B. Vasilescu et al., "Quality and productivity outcomes relating to continuous integration in github", ESEC/FSE 2015.

Motivation

Jenkins

Travis CI

→Teams gather immediate feedback on changes
→ Improve productivity

B. Vasilescu et al., "Quality and productivity outcomes relating to continuous integration in github", ESEC/FSE 2015.

Research Questions

(RQ1)
Strategies?

How do developers repair dependency-related build breakage?

(RQ2)
Automation?

To what extent can we automatically repair dependency-related build breakage?

Motivation

Jenkins

Travis CI

→Teams gather immediate feedback on changes
→ Improve productivity

B. Vasilescu et al., "Quality and productivity outcomes relating to continuous integration in github", ESEC/FSE 2015.

Research Questions

(RQ1)
Strategies?

How do developers repair dependency-related build breakage?

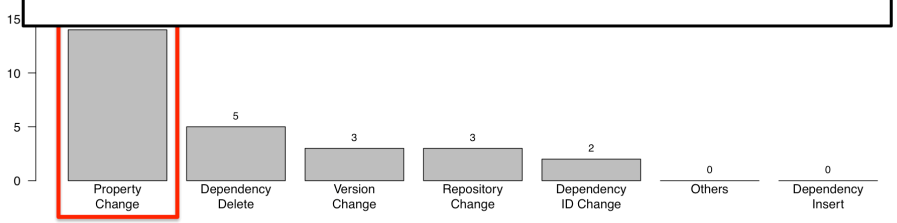
(RQ2)
Automation?

To what extent can we automatically repair dependency-related build breakage?

Results

- 27/37 revisions repaired with single change

→ Repair strategies!



Motivation

GitHub

Jenkins

Bitbucket

Travis CI

→Teams gather immediate feedback on changes
→ Improve productivity

B. Vasilescu et al., "Quality and productivity outcomes relating to continuous integration in github", ESEC/FSE 2015.

Research Questions

(RQ1) Strategies?

How do developers repair dependency-related build breakage?

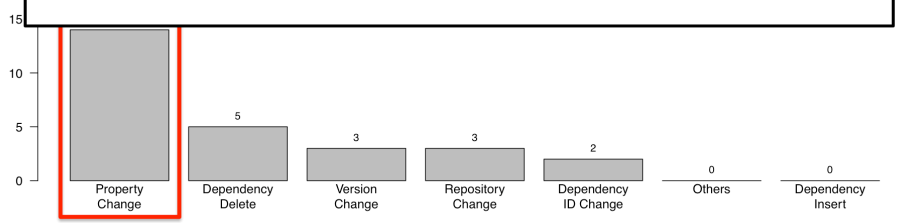
(RQ2) Automation?

To what extent can we automatically repair dependency-related build breakage?

Results

- 27/37 revisions repaired with single change

→ Repair strategies!



Results

Project	Fixed	Not Fixed	n=1	ID	SIM
async-http-client	1 (100%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
immutables	1 (100%)	0 (0%)	1 (100%)	1 (100%)	0 (0%)
closure-compiler	1 (100%)	0 (0%)	1 (100%)	1 (100%)	0 (0%)
symphony	4 (57%)	3 (43%)	4 (100%)	2 (50%)	2 (50%)
wildfly	0 (0%)	1 (100%)	- (-)	- (-)	- (-)
YCSB	1 (20%)	4 (80%)	1 (100%)	0 (0%)	1 (100%)
alluxio	10 (37%)	17 (63%)	3 (30%)	1 (10%)	7 (70%)
libgdx	5 (100%)	0 (0%)	5 (100%)	1 (20%)	3 (60%)
hazelcast	7 (50%)	7 (50%)	7 (100%)	7 (100%)	0 (0%)
...					
Total	**45 (54%)**	**39 (46%)**	**34 (76%)**	**16 (36%)**	**20 (44%)**