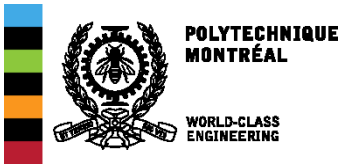
Understanding and Tooling Framework API Evolution

Wei Wu

Supervised by
Yann-Gaël Guéhéneuc and Giuliano Antoniol



Outlines

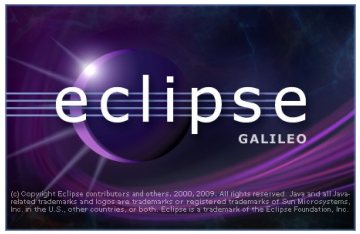
- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives

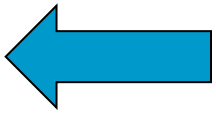
Outlines

- **Framework API evolution, problem?**
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives

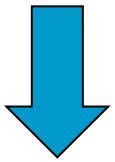
Framework API Evolution?

Framework/Client Program





Client Program



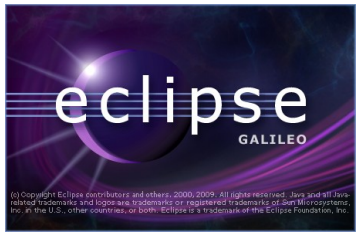
Framework

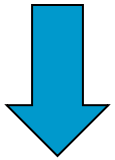
```java
public class SampleHandler extends AbstractHandler {
    /**
     * the command has been executed, so extract extract the needed information
     * from the application context.
     */
    public Object execute(ExecutionEvent event) throws ExecutionException {

        // Implemented by client programs

        return null;
    }
}
```

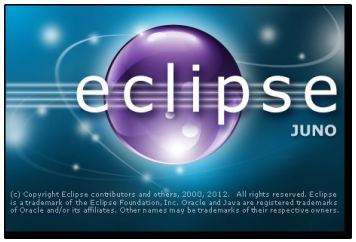
API

Framework API Evolution?

Framework

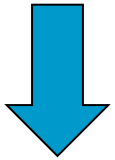






API

```
/**
 * An implementation for the extension registry API.
 */
public class ExtensionRegistry implements IExtensionRegistry {

    public void stop() {
```



```
/**
 * An implementation for the extension registry API.
 */
public class ExtensionRegistry implements IExtensionRegistry {

    public void stop(Object key) {
```

Prob

The New York Times

INVESTMENT BANKING | LEGAL/REGULATORY

JPMorgan Chase Hacking Affects 76 Million Households

By JESSICA SILVER-GREENBERG, MATTHEW GOLDSTEIN and NICOLE PERLROTH OCTOBER 2, 2014 12:50 PM

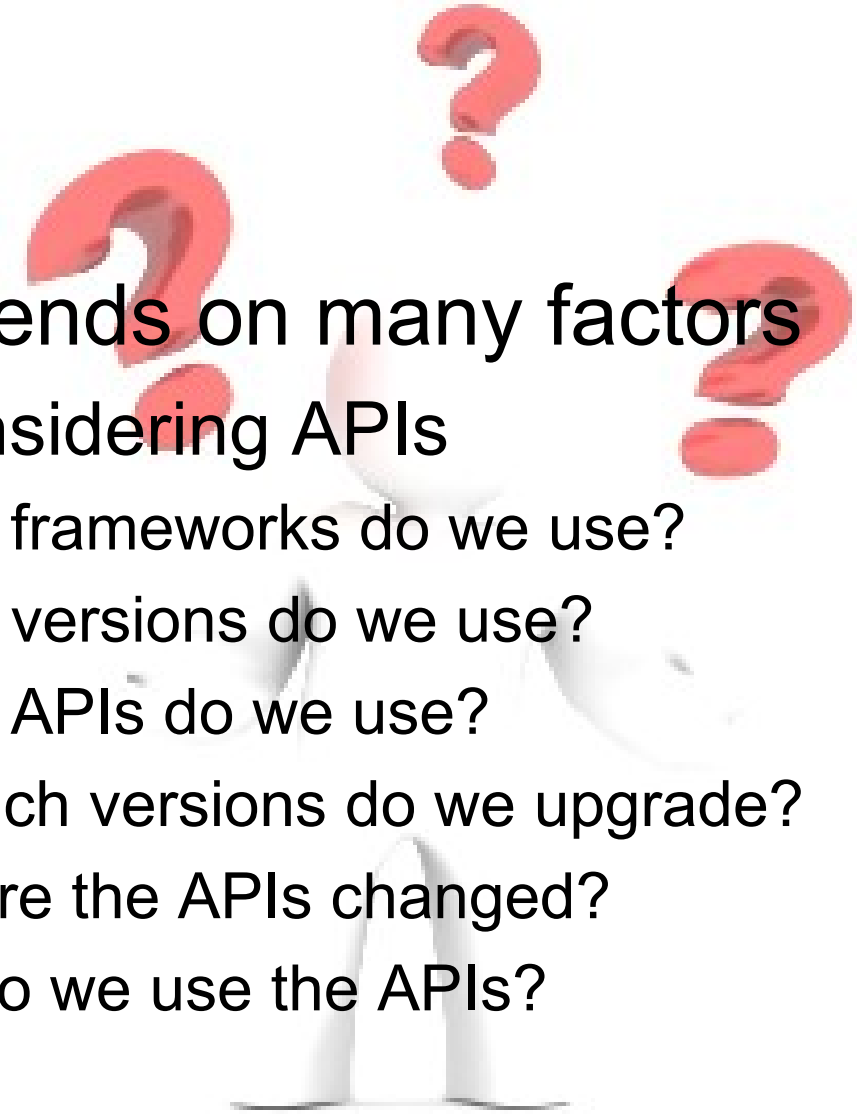
526 Comments

The Manhattan headquarters of JPMorgan Chase, which securities filings revealed was attacked by hackers over the summer. Andrew Burton/Getty Images

Problem!

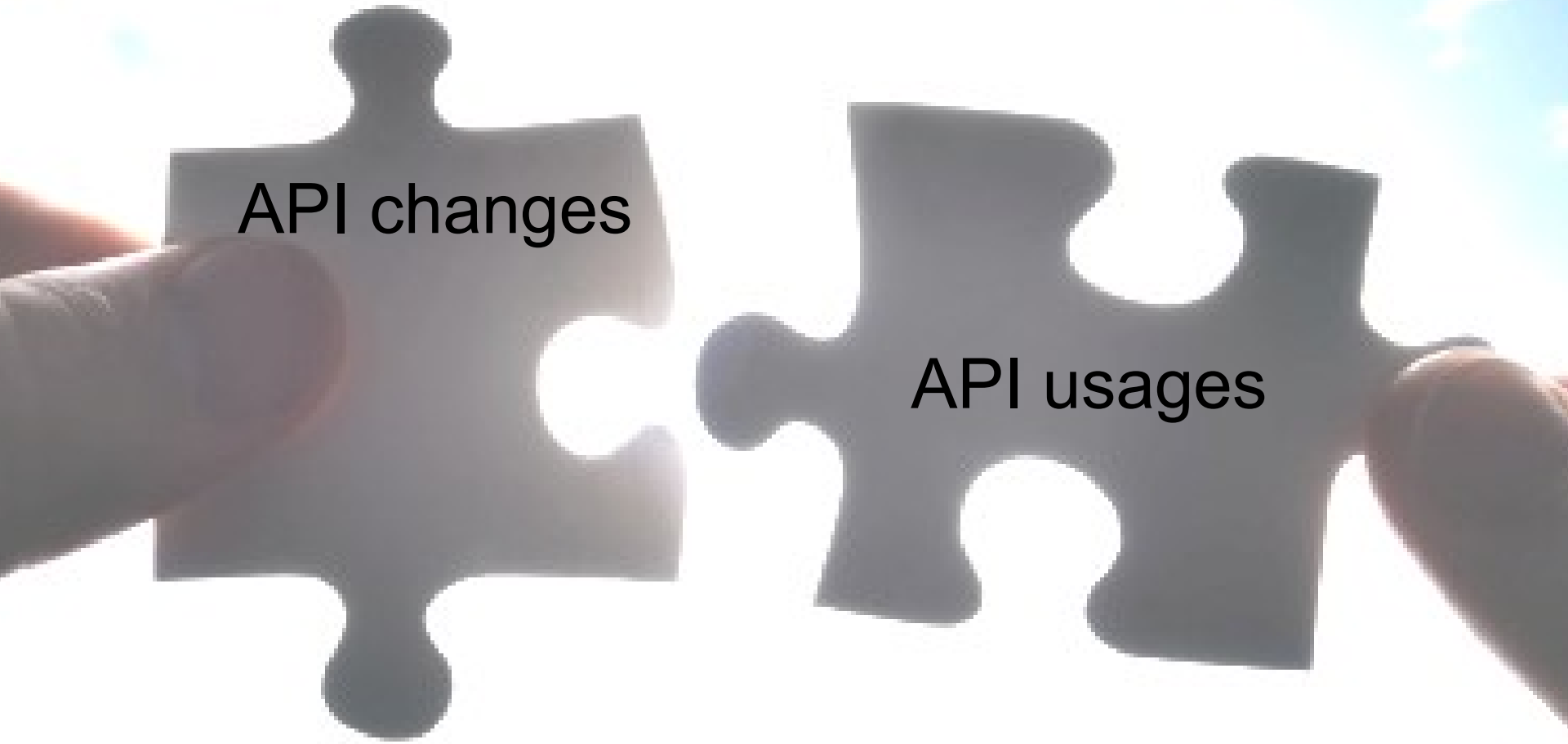
- Evolving with frameworks is costly
  - Raemaekers et al. (ICSM 2012)
    - Upgrading an authentication framework
    - A whole week of work
  - Linux Debian Distribution
    - Upgrading Perl from 5.10 to 5.12 took
    - Seven weeks to complete

Problem!



- Cost depends on many factors
  - Just considering APIs
    - Which frameworks do we use?
    - Which versions do we use?
    - Which APIs do we use?
    - To which versions do we upgrade?
    - How are the APIs changed?
    - How do we use the APIs?

Problem!



Study API change and usage on a large scale

Outlines

- Framework API evolution, problem?
- API change and usage mining
  - **Previous works**
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives

Previous Works

- API changes
  - Cossette and Walker (2012)
    - API changes in Struts, Log4j
  - Des Rivières (2008)
    - API change cl… (about 150 types)
    - No dete…
  - He… (2011)
    - …sons for API changes in AWT and Swing

No large-scale study on API changes

Previous Works

- API usages
  - Businge et al. (2013)
    - Official and internal API usa… plug-ins
  - Lämmel et al. (2011)
    - API usages in… ge on a large scale
  - Roover… 3)
    - …s from various angles, such as intent, …eholders, etc. in enhanced QUALITAS corpus

No studies on API usage **and** changes

Previous Works

- API usages **and** changes
  - Robbes et al. (2012)
    - Deprecated APIs in Smalltalk
  - Dietrich et al. (2014)
    - Binary API in[illegible]s in QUALITAS corpus
    - Only 8[illegible]nt programs

Studies limited to specific API changes

Solution

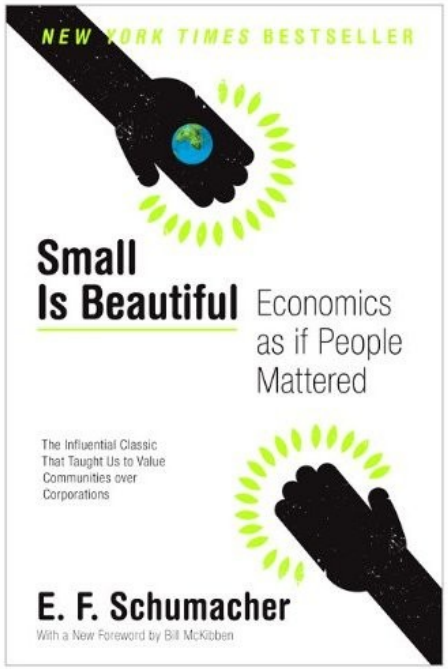
- Study API change and usage together on a large scale to answer
  - RQ1: How do framework APIs change?
  - RQ2: How do framework API changes affect client programs?
- Need a tool to collect relevant data

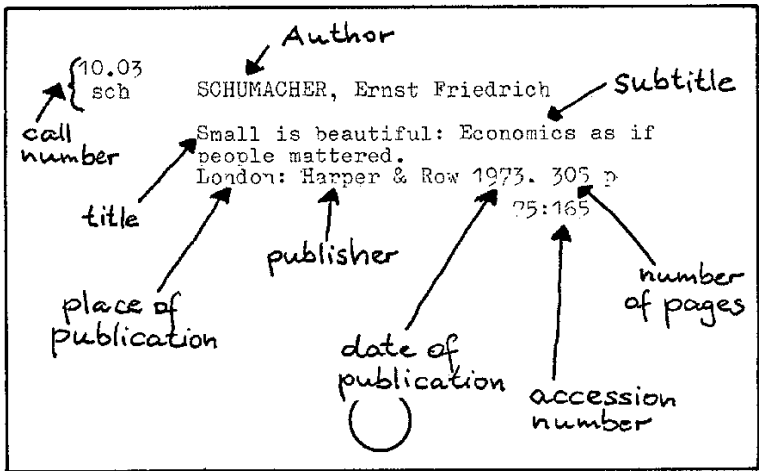
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - **Tooling**
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives

Tooling – ACUA

- API Change and Usage Auditor to collect API-related data





Tooling – ACUA

- Generating API change and usage reports
  - Which frameworks do we use?
  - Which versions do we use?
  - Which APIs do we use?
  - To which versions do we upgrade?
  - How are the APIs changed?
  - How do we use the APIs?

"ACUA: API change and usage auditor",
W. Wu, B. Adams, Y.-G. Guéhéneuc,
and G. Antoniol, SCAM Tool Demo, 2014

Outlines

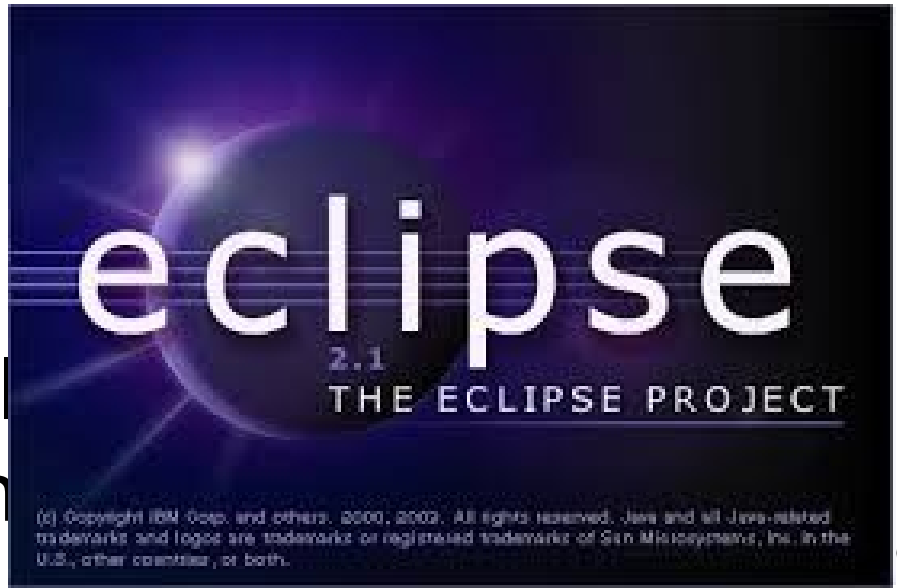
- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - **Dataset**
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives

API Change and Usage Mining

14 ars

el

in

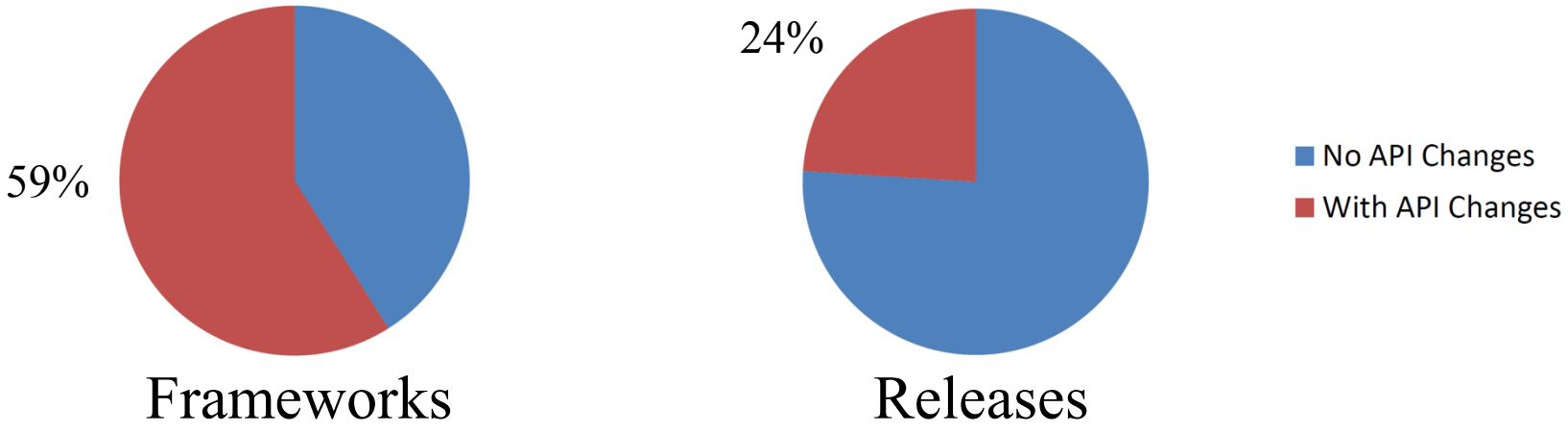


/60

Outlines

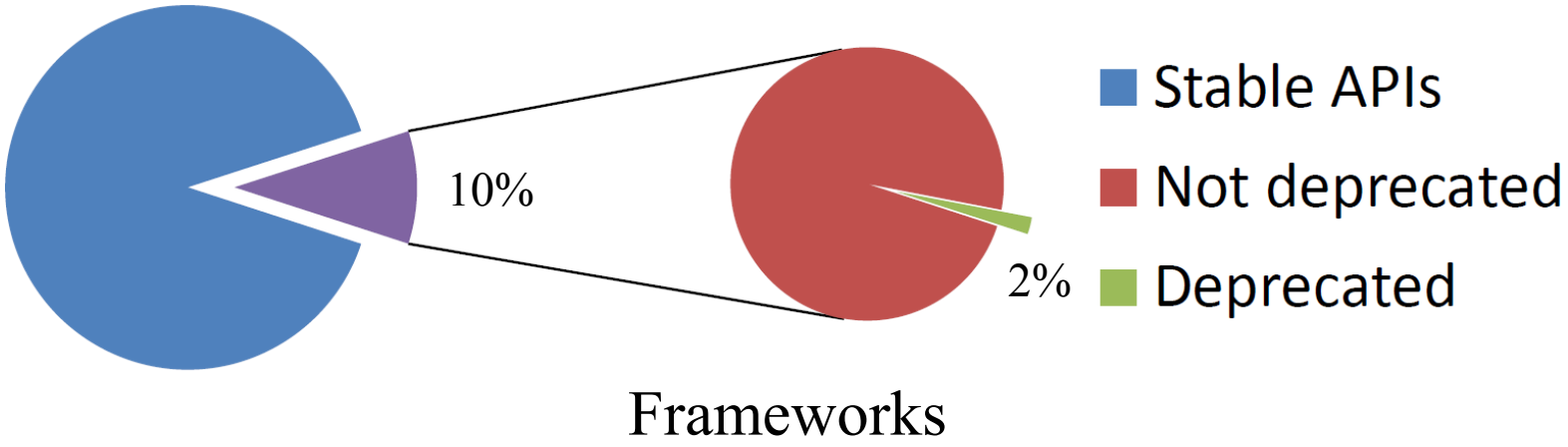
- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - **General study**
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives

RQ1: How do frwk APIs change?



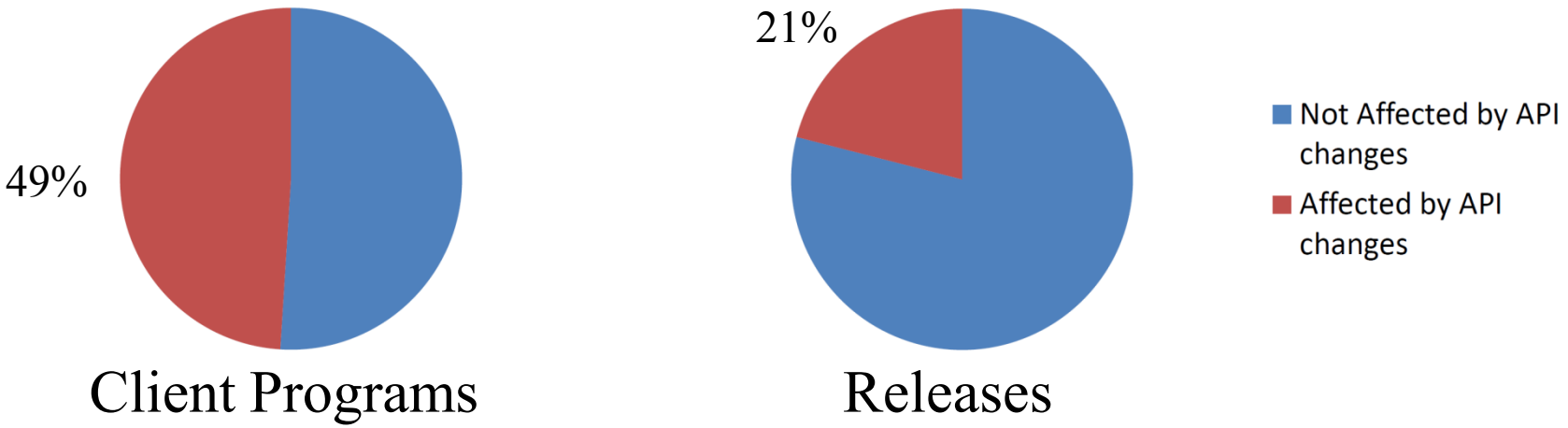
API changes in 59% of frameworks
and in 24% of their releases

RQ1: How do frwk APIs change?



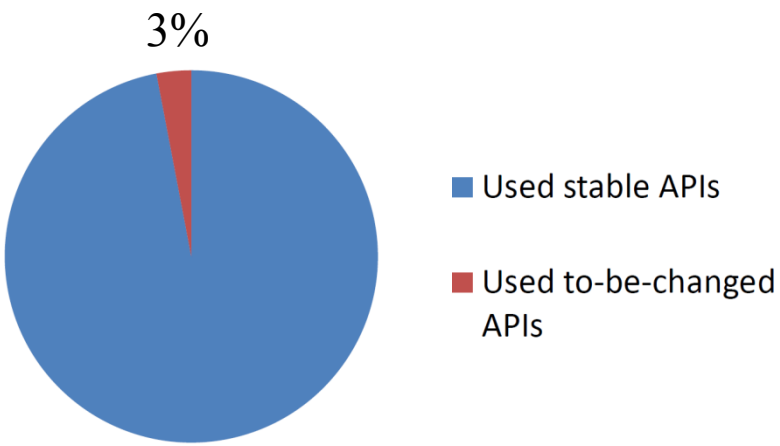
At method level, 10% of APIs are changed, only 2% are deprecated

RQ2: How do frwk API changes affect client programs?



API changes affect 49% of client programs and 21% of their releases

RQ2: How do frwk API changes affect client programs?



API changes affect only 3% of the APIs used by client programs, none deprecated

Summary

- Program-level
  - Framework API changes happen (59%)
  - Client programs are affected (49%)
- Method-level
  - 10% of APIs are changed
  - 3% of used APIs are affected
- Developers do not document API changes
  - 2% of the changed APIs are deprecated
  - None of them are used by client programs

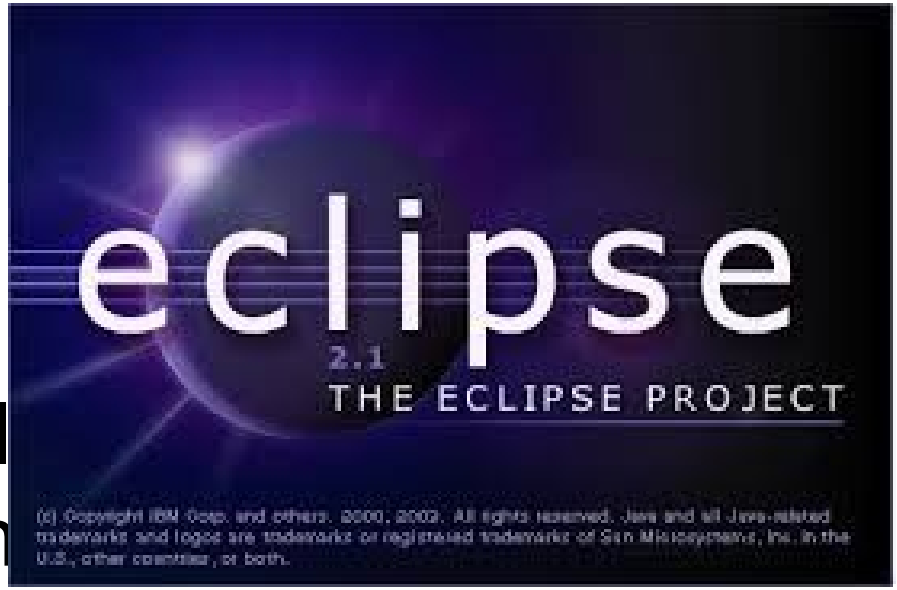
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - **Detailed study**
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives

Dataset

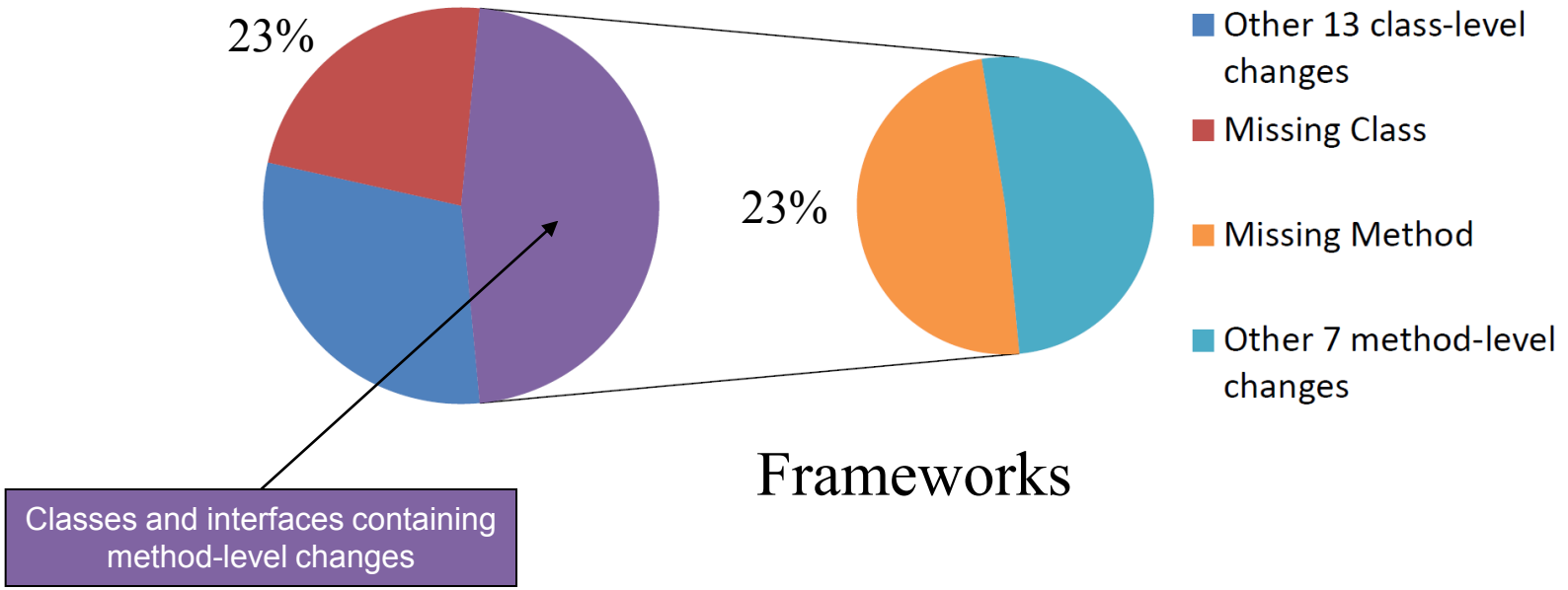
el

in



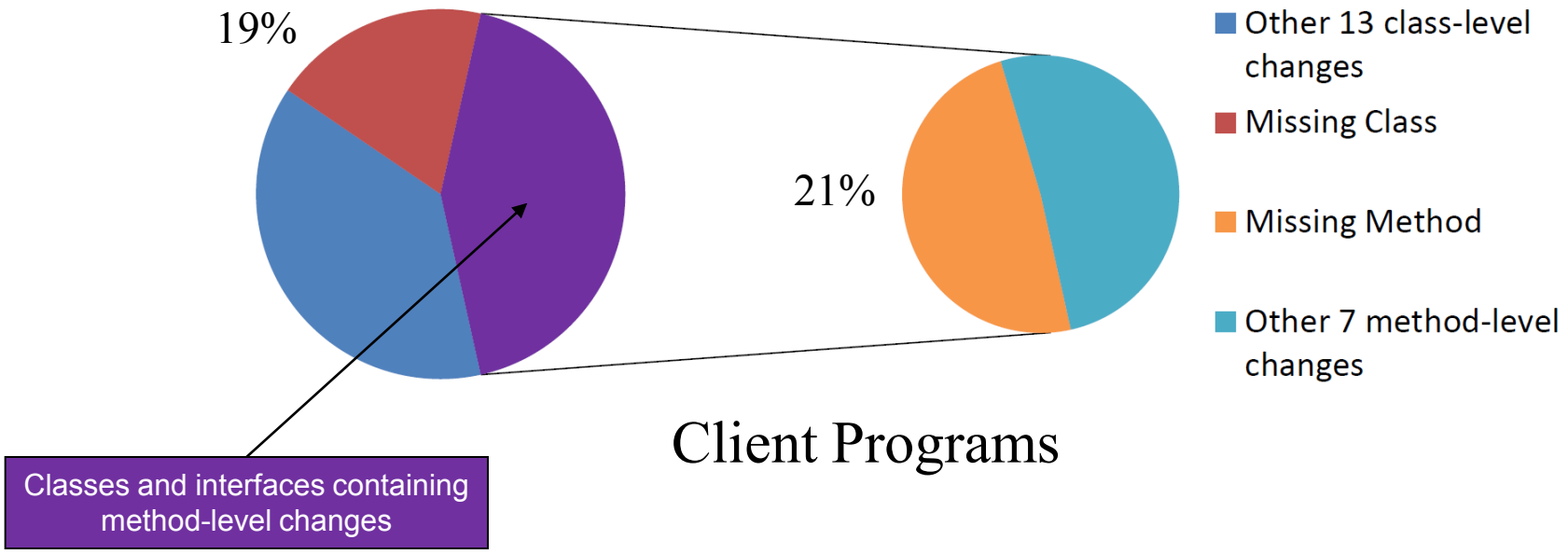
"An Exploratory Study of API Changes and Usages based on Apache and Eclipse Ecosystems", W. Wu, Y.-G. Guéhéneuc, and G. Antoniol, under review in EMSE

RQ1: How do frwk APIs evolve?



Missing classes and methods are the most frequent: 46% of total API changes

RQ2: How do frwk API changes affect client programs?



Missing classes and methods affect client programs most frequently (40%)

Summary

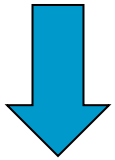
- Missing classes and methods
  - 46% of API changes
  - 40% of API changes affect client programs
- Insufficient documentations

- API change rules help developers find the replacements of these missing APIs

Remedy for Missing Classes and Methods

- API change rule
  - A map between a missing API and its replacement in a new release of a framework
  - Target methods represent missing APIs

```
VCardComposer.shouldAppendCharsetAttribute(List<String>)
```



```
VCardBuilder.shouldAppendCharsetParam(String[])
```

Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - **Previous works**
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives

Previous Approaches

- **AURA (2010)**
- HiMa (2012)
- Kim et al (2007)
- MadMatch (2013)
- Schäfer et al. (2008)
- SemDiff (2011)
- ...

"AURA: a hybrid approach to identify framework evolution", W. Wu, Y.-G. Guéhéneuc, G. Antoniol, and M. Kim, ICSE 2010

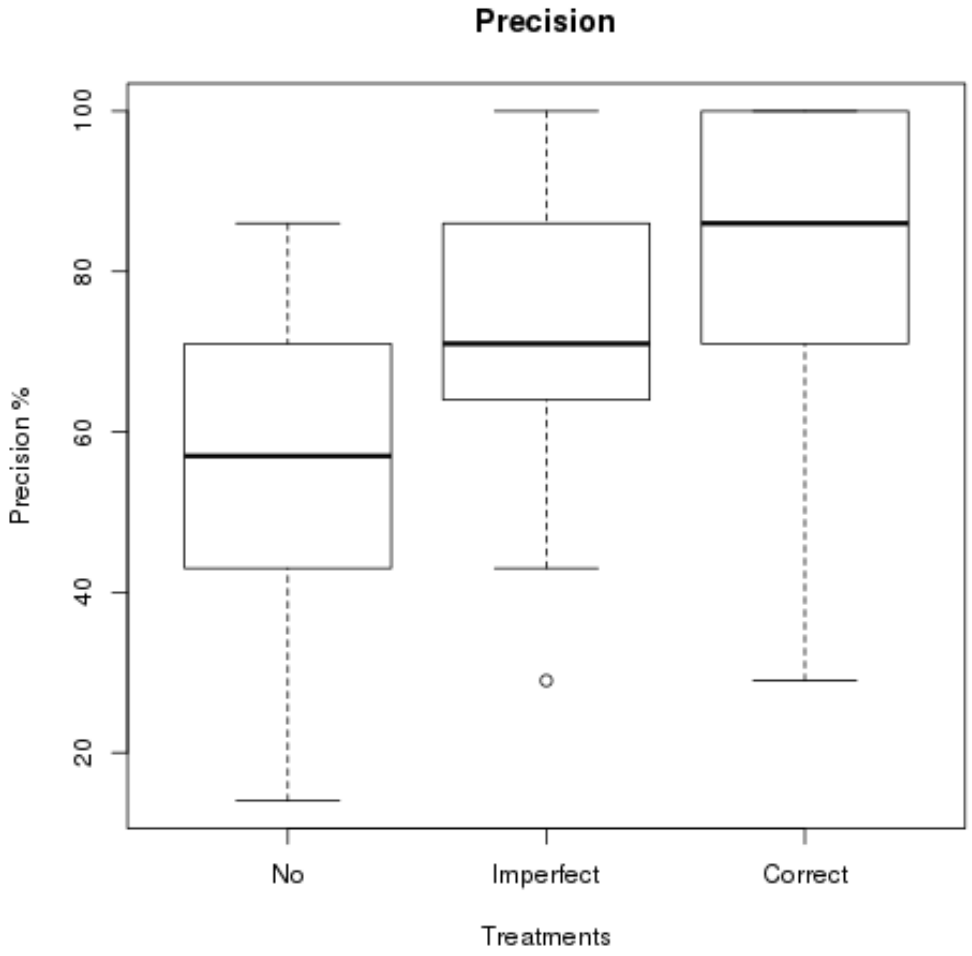
Limitations

- Generated API change rules are imperfect
  - Precisions vary on different frameworks
- No study on the usefulness of imperfect API change rules

Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - **Usefulness study**
  - Feature usage study
- Conclusion and perspectives

API Chang Rule Usefulness



"The impact of imperfect change rules on framework API evolution identification: an empirical study", W. Wu, A. Serveaux, Y.-G. Guéhéneuc, and G. Antoniol, EMSE, 2014

API Chang Rule Usefulness

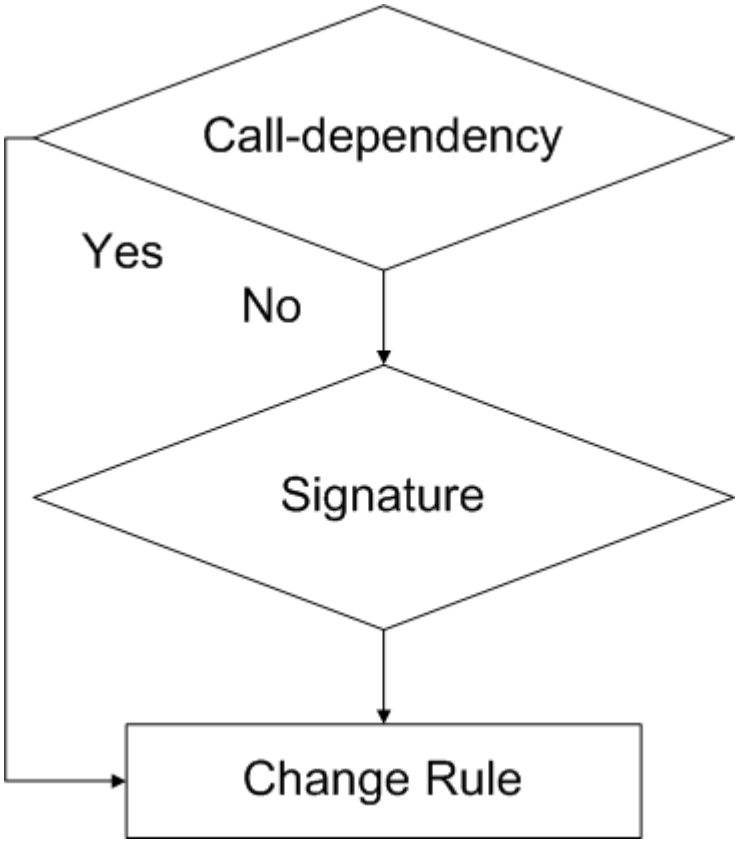
- API change rules are useful
- The more accurate the API change rules, the more helpful

- How can we improve the accuracy of API change rule building?

Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - **Feature usage study**
- Conclusion and perspectives

AURA



"AURA: a hybrid approach to identify framework evolution",
W. Wu, Y.-G. Guéhéneuc, G. Antoniol, and M. Kim, ICSE 2010

Features Used in Previous Works

Approach	Call-Dependency	Signature	Inheritance	Source Comment	Metrics	Structural	SVN Comment
SemDiff	√						
Beagle	√	√			√		
S. Kim et al.	√	√			√		
M. Kim et al.		√					
MADMatch	√	√	√			√	
HiMa	√	√					√
Schäfer et al.	√	√	√				
AURA	√	√					
UMLDiff		√	√	√		√	

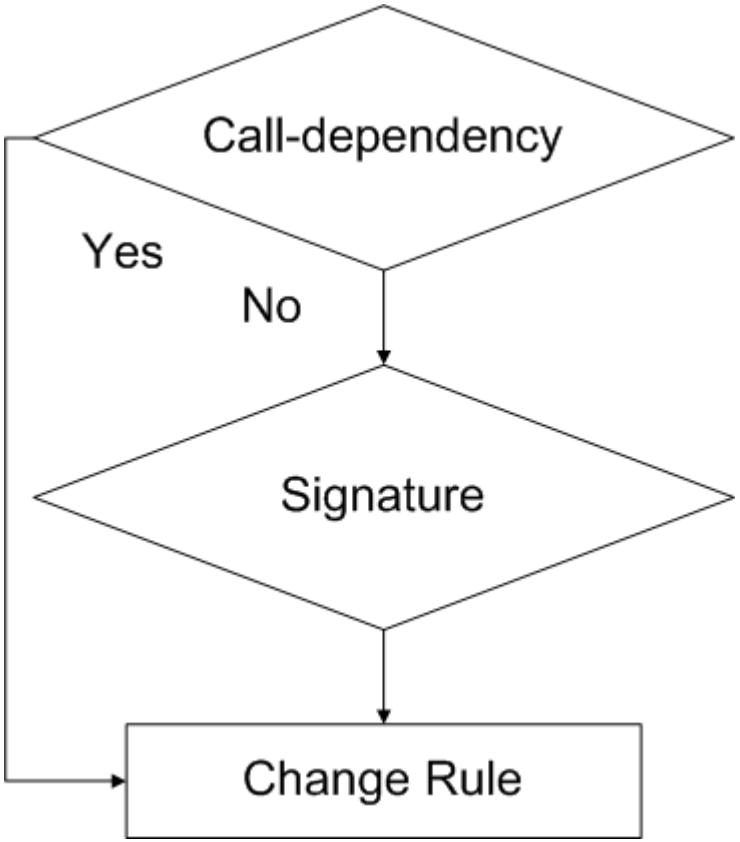
Combination of Features

- Single feature
  - **SemDiff**
  - **M. Kim et al.**
- Prioritised features
  - Explicit: **AURA**, **HiMa**, **Schäfer et al.**
  - Weighting: **Beagle**, **S. Kim et al.**
  - Mixed: **MADMatch**, **UMLDiff**

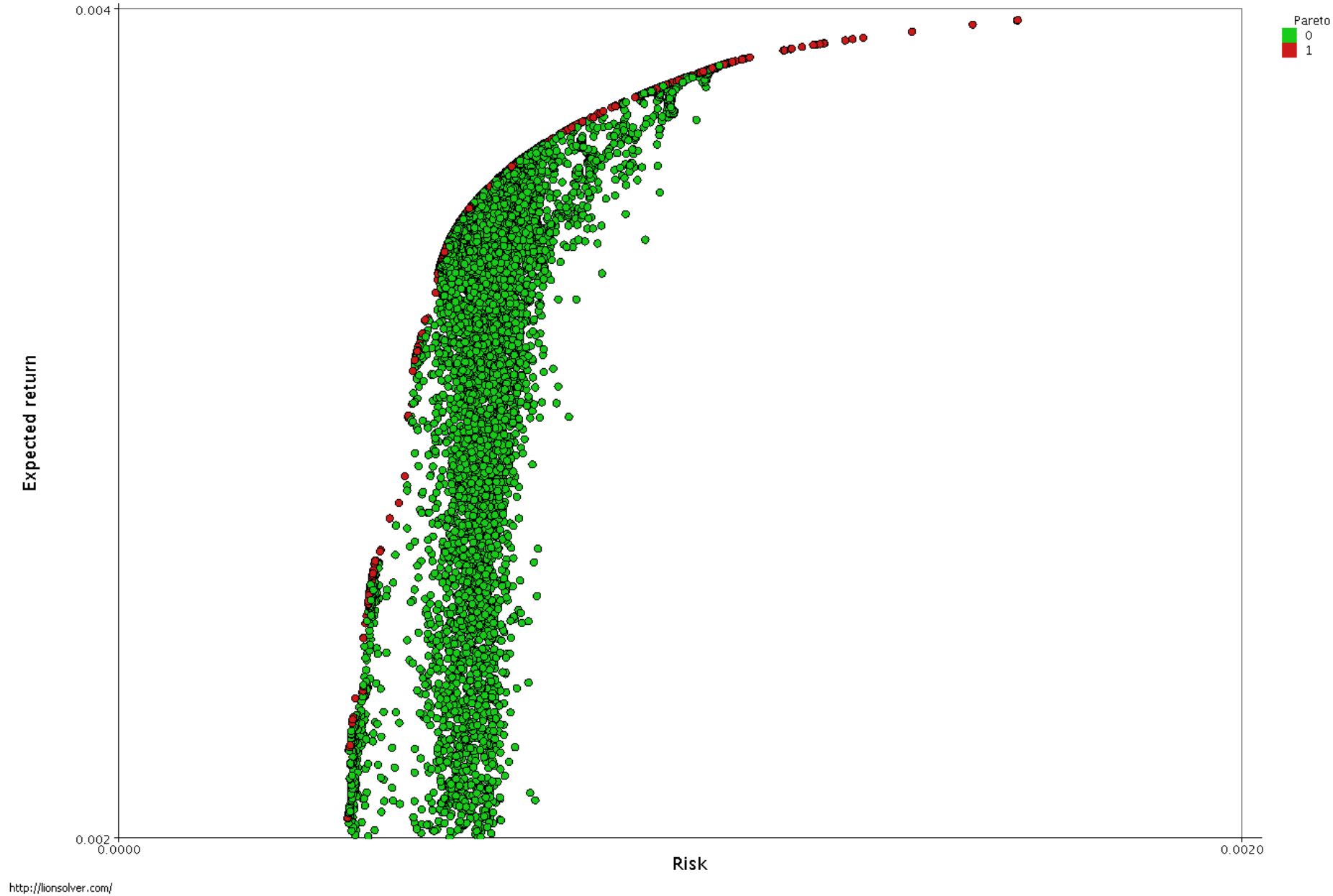
Limitations

- No study on the effectiveness of features
- No study on the effectiveness of the combinations of features
- Prioritised multi-feature approaches
  - Potential contradictions among features
    - High priority features shadow lower priority ones
    - Hard to extend with new/different features

AURA



"AURA: a hybrid approach to identify framework evolution",
W. Wu, Y.-G. Guéhéneuc, G. Antoniol, and M. Kim, ICSE 2010


0.004
0.002
0.0000
0.0020
Expected return
Risk
Pareto
0
1
http://lionsolver.com/

Research Questions

- RQ1: How effective are the features used in the literature to build API change rules?
- RQ2: Can we use MOOP techniques to improve over prioritised multi-feature approaches?

MOFAE

- MOFAE: multi-objective framework for API evolution
  - Reformulates API change rule building
  - as a MOOP
  - Uses features as objectives
  - Uses jMetal MOOP algorithm framework
  - Allows flexible feature configuration

"Feature Usages in Framework API Evolution Identification",
W. Wu, Y.-G. Guéhéneuc, and G. Antoniol, under review in EMSE

Four Experiment Features

Approach	Call-Dependency	Signature	Inheritance	Source Comment	Metrics	Structural	SVN Comment
SemDiff	√						
Beagle	√	√			√		
S. Kim et al.	√	√			√		
M. Kim et al.		√					
MADMatch	√	√	√			√	
HiMa	√	√					√
Schäfer et al.	√	√	√				
AURA	√	√					
UMLDiff		√	√	√		√	

Existing Approaches

- Difficulties to compare
  - Not all available
  - Not always executable
  - One recommendation per target method

Experiment Approaches

- Single-feature approaches
- Multi-feature approaches

Prioritised	MOFAE	Feature Combination
P1	**M1**	Call-dependency + Signature
P2	**M2**	Source code comments + Signature
P3	**M3**	Inheritance + Signature
PA	**MA**	Call-dependency + Inheritance + Source code comments + Signature

Experiment Approaches

- Outputs
  - MOFAE
    - Maximum 6 recommendations
  - Single-feature and prioritized approaches
    - Top 6 recommendations
- More conservative for MOFAE approaches

Experiment Approaches

- Comparison
  - Number of target methods with correct replacements
  - Correct recommendation position

```
MeterPlot.getDialBorderColor()

MeterPlot.getDialBackgroundPaint()
MeterPlot.getDialOutlinePaint()
MeterPlot.getNormalPaint()
MeterPlot.getCriticalPaint()
MeterPlot.getValueFont()
MeterPlot.getNeedlePaint()
```

Target Frameworks

	Releases	# Methods	# Target Methods
Android SDK	2.1_r2.1p2	20,516	106
	2.2.3_r2	21,214	
jEdit	4.1	2,774	87
	4.2	3,547	
jFreeChart	0.9.11	4,751	30
	0.9.12	5,197	
jHotDraw	5.2	1,486	43
	5.3	2,265	
Log4j	1.0.4	906	15
	1.1.3	1,110	
Struts	1.1	5,973	91
	1.2.4	6,111	

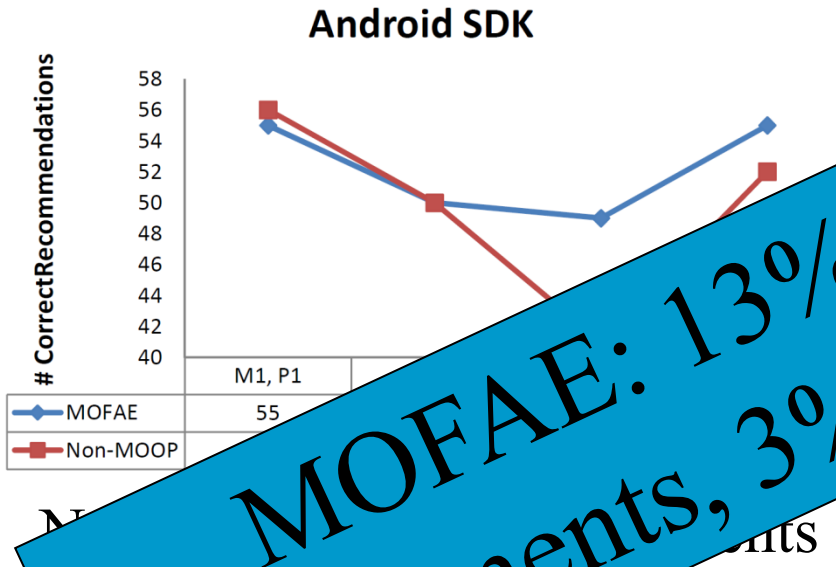
RQ1: How effective are the features to identify change rules?

# Correct	Call-dependency	Inheritance	Comment	Signature
Android SDK	22	9	20	50/106
jEdit	12	5		30/87
jFreeChart	5		15	29/30
jHotDraw	1	14	20	36/43
Log4j		3	8	12/15
Struts	2	17	8	19/91

Signatures are most effective

The smaller, the better

RQ2: Can we use MOOP techniques to improve multi-feature approaches?

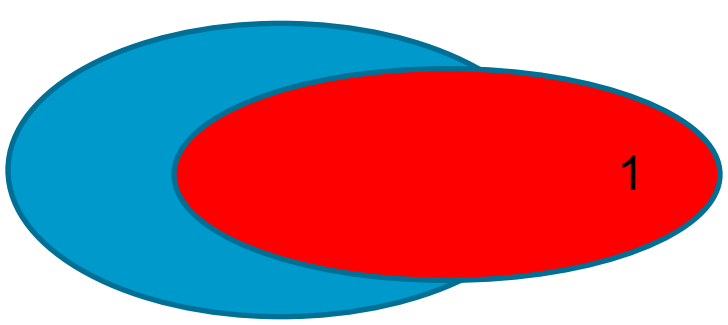


	M1, P1	M2, P2	M3, P3	MA, PA
AE	1.09	1.28	1.39	1.13
Non-MOOP	1.39	1.52	1.56	1.33

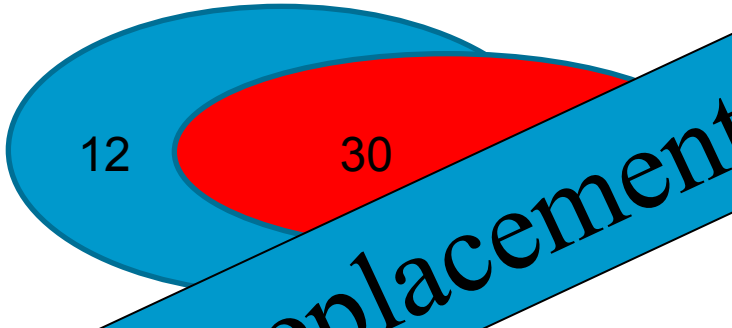
Pos of Correct Replacements

MOFAE: 13% more correct replacements, 3% higher in position

RQ2: Can we use MOOP techniques to improve multi-feature approaches?

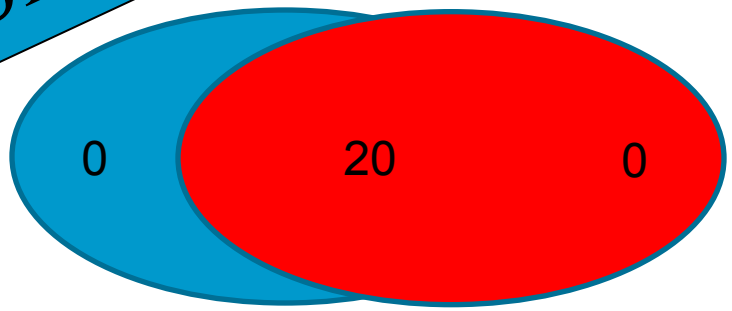


jEdit



Draw

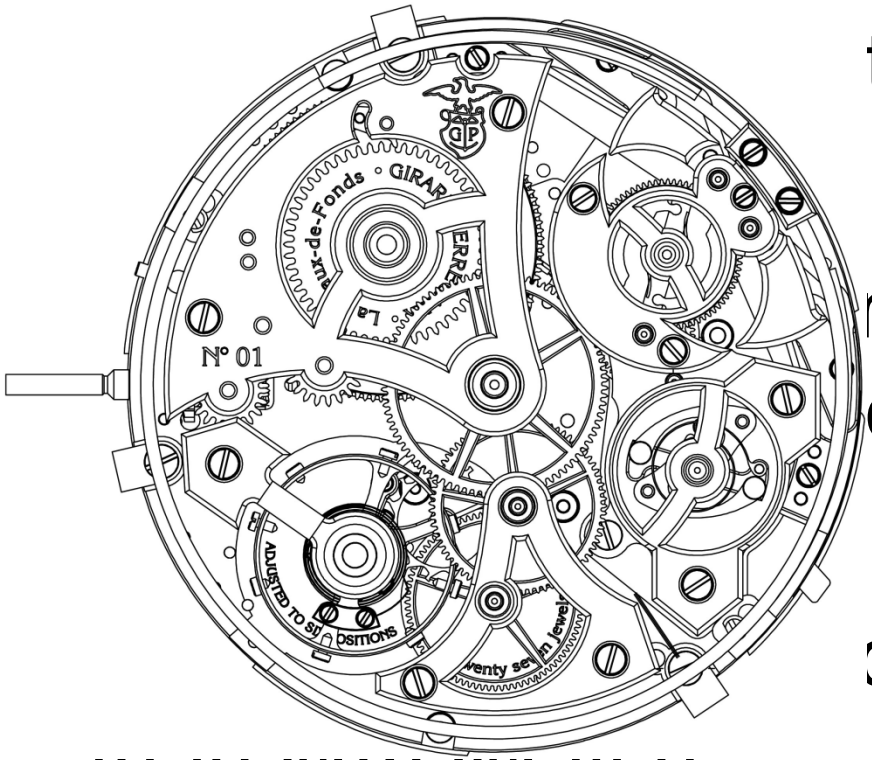
jFreeChart



Struts

MOFAE: 20% more correct replacements

Summary

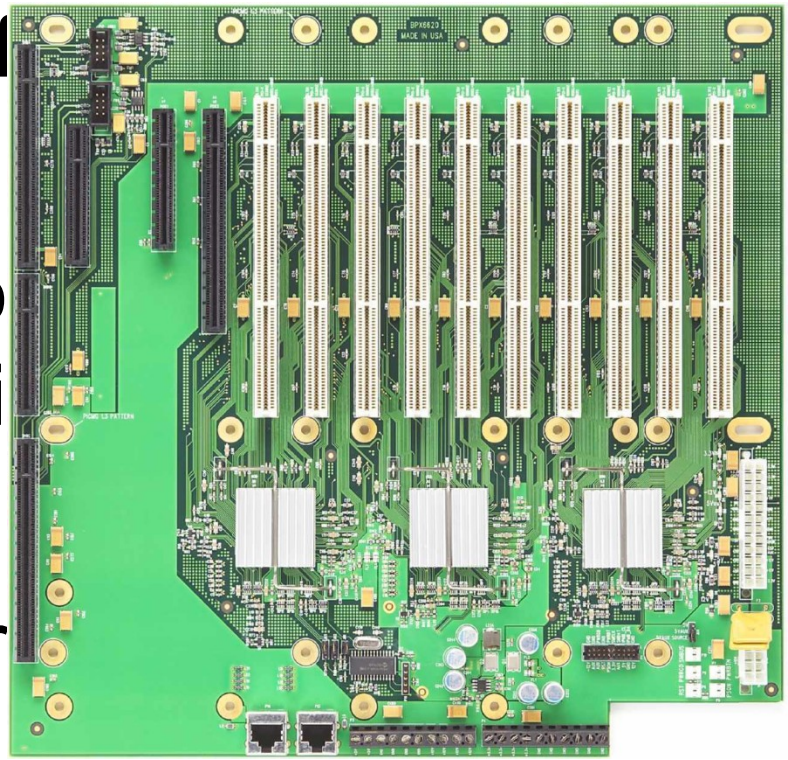


t ef

mo

osi

cor



than MADMatch

Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- **Conclusion and perspectives**

Thesis

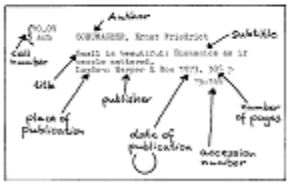
Following analyses of the reality of **API changes and usages**, of the **usefulness** of API change rules, and of the **effectiveness** of the features used to build these rules, we can build **more effective and extensible** API change-rule recommendation tools with **MOFAE**.

Perspectives

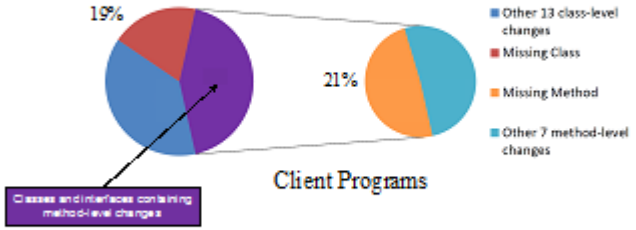
- Near term
  - Extensive qualitative analyses
  - More effective features
  - Tools for other upgrading tasks
  - Developers' interviews
- Long term
  - Language-supported API visibility
  - Framework API standards
  - Independent framework evaluation

Tooling – ACUA

- API Change and Usage Auditor to collect API-related data

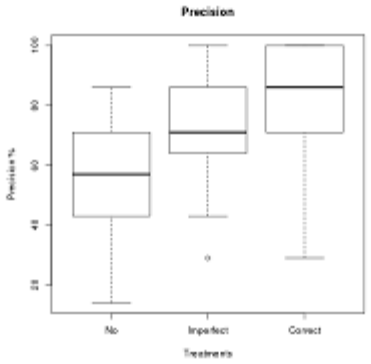


RQ2: How do frwk API changes affect client programs?



Missing classes and methods affect client programs most frequently (40%)

API Chang Rule Usefulness



RQ2: Can we use MOOP techniques to improve multi-feature approaches?