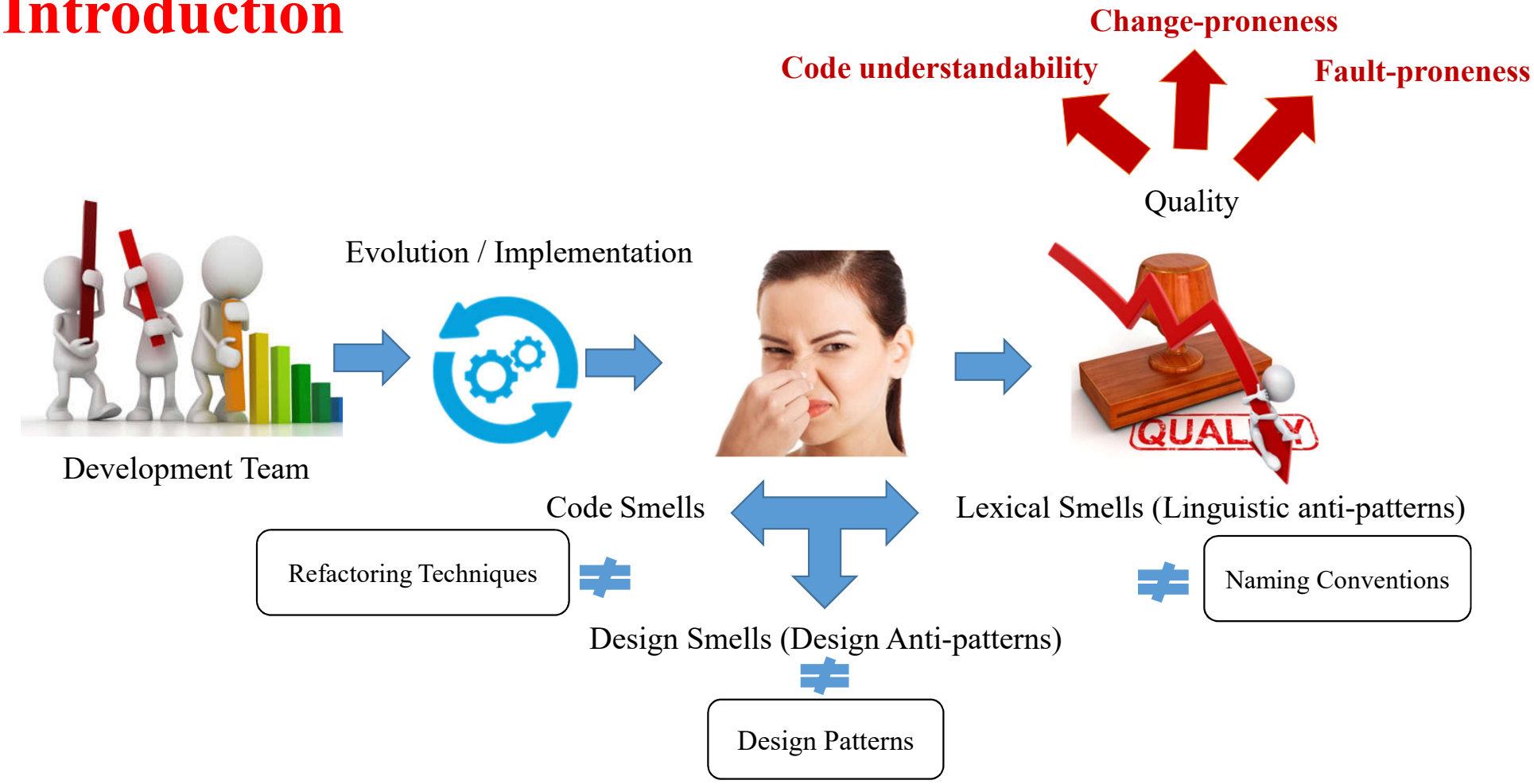
Ph.D. Defense
LINGUISTIC ANTI-PATTERNS: IMPACT ANALYSIS ON CODE QUALITY

Zeinab (Azadeh) Kermansaravi

Dr. Foutse Khomh and Dr. Yann-Gaël Guéhéneuc

Département de génie informatique et génie logiciel
Polytechnique Montréal, Québec, Canada
July 16th, 2019

Introduction



Research Context

A New Family of Software Anti-Patterns: Linguistic Anti-Patterns

Venera Arnaoudova[1,2], Massimiliano Di Penta[3], Giuliano Antoniol[2], Yann-Gaël Guéhéneuc[1]
[1] *Ptidej Team, DGIGL, École Polytechnique de Montréal, Canada*
[2] *Soccer Lab., DGIGL, École Polytechnique de Montréal, Canada*
[3] *Department of Engineering, University of Sannio, Benevento, Italy*
E-mails: venera.arnaoudova@polymtl.ca, dipenta@unisannio.it, antoniol@ieee.org, yann-gael.gueheneuc@polymtl.ca

***Abstract*—Recent and past studies have shown that poor source code lexicon negatively affects software understandability, maintainability, and, overall, quality. Besides a poor usage of lexicon and documentation, sometimes a software artifact description is misleading with respect to its implementation. Consequently, developers will spend more time and effort when understanding these software artifacts, or even make wrong assumptions when they use them.**

This paper introduces the definition of software linguistic antipatterns, and defines a family of them, i.e., those related to inconsistencies (i) between method signatures, documentation, and behavior and (ii) between attribute names, types, and comments. Whereas "design" antipatterns represent recurring, poor design choices, linguistic antipatterns represent recurring,

more) than the method actually does. One such example, occurred in Eclipse 1.0, is a method named *isClassPathCorrect* defined in class *ProblemReporter*. One would expect that such a method returns a Boolean; instead, the method does not return any value and sets an attribute and calls another method to perform the task.

This paper represents the starting point for the definition of a new family of software antipatterns, named linguistic antipatterns. Software antipatterns—as they are known so far—are opposite to design patterns [7], i.e., they identify "poor" solutions to recurring design problems, for example, Brown's 40 antipatterns describe the most common pitfalls

A.3. “Set” method returns (ArgoUML-0.10.1)

```
public Dimension setBreadth
        (Dimension target, int source) {
  if (orientation == VERTICAL)
    return new Dimension(source,
        (int)target.getHeight());
  else
    return new Dimension(
                (int)target.getWidth(), source);
}
```

Problem Statement & Research Goal

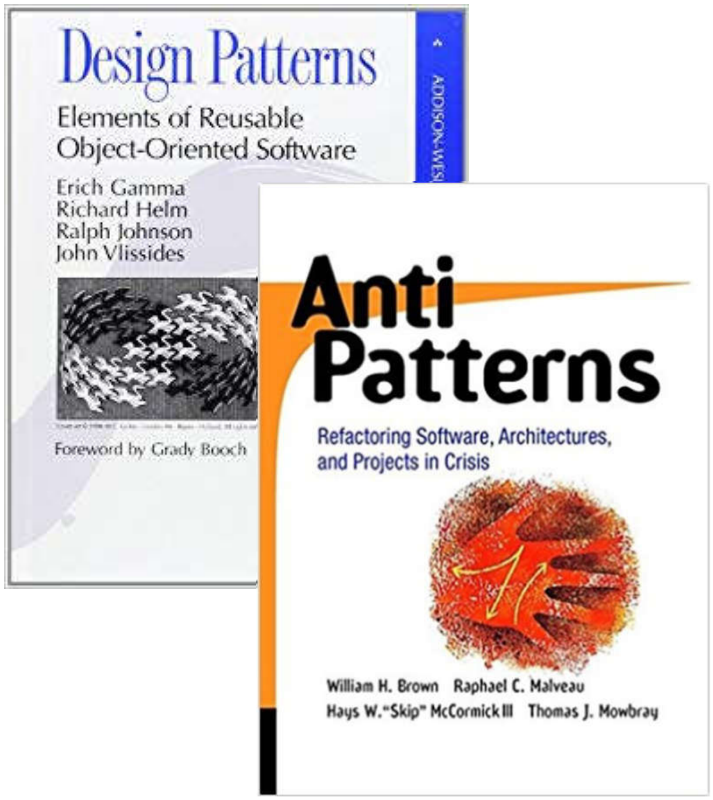
Venera Arnaoudova (2016)

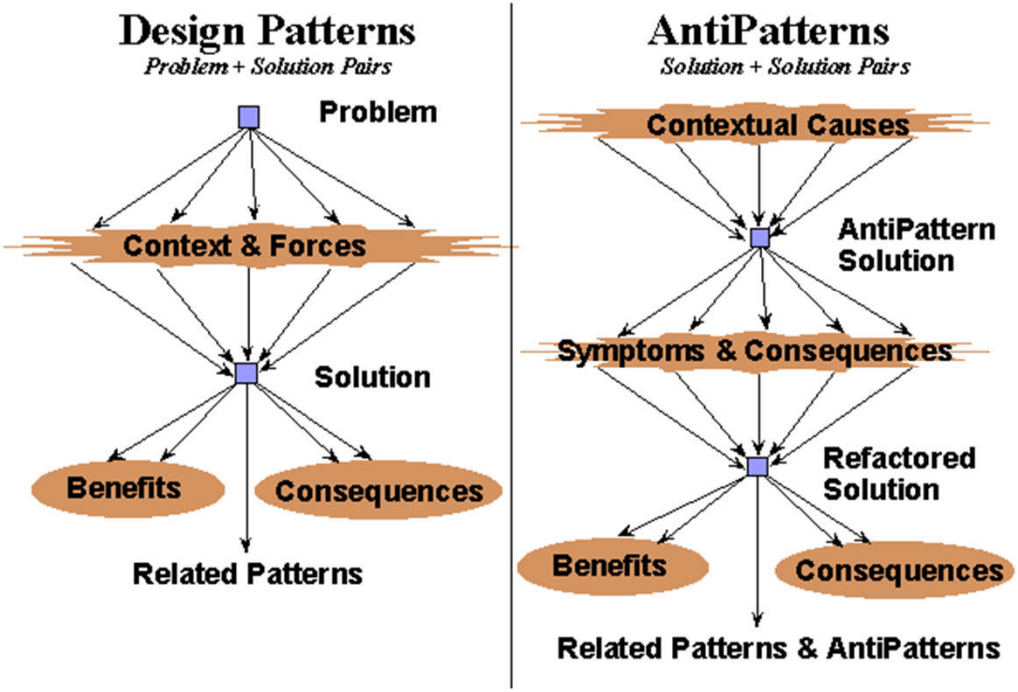
Johannes Hofmeister (2017)

Sarah Fakhoury (2018)

- Do **different LAs** affect understandability **equally**.
- Whether **knowing LAs** improves understandability.

(Contribution 1)
LAs and Program Comprehension


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
Anti Patterns
Refactoring Software, Architectures, and Projects in Crisis
William H. Brown Raphael C. Malveau
Hays W. "Skip" McCormick III Thomas J. Mowbray


Design Patterns
Problem + Solution Pairs
Problem
Context & Forces
Solution
Benefits
Consequences
Related Patterns
AntiPatterns
Solution + Solution Pairs
Contextual Causes
AntiPattern Solution
Symptoms & Consequences
Refactored Solution
Benefits
Consequences
Related Patterns & AntiPatterns

The combination of two anti-patterns impacted negatively and significantly the system understandability

Marwen Abbes (2011)

Classes participating in anti-patterns are more change- and fault-prone than others.

Foutse Khomh (2012)

Classes included in DAPs with relationships with DPs are more change-prone than others but less fault-prone.

Fehmi Jaafar (2013)

- What is the relation between **LAs** and **DAPs** and their **co-occurrence** impacts **change- and fault-proneness**?

(Contribution 2)
LAs, DAPs and change- and fault-proneness

Removal (or lack of use) of DPs and introduction of DAPs degrade software quality.

Foutse Khomh and Yann-Gaël Guéhéneuc (2008)

DAPs mutate over the time. Mutations are more fault-prone than others.

Fehmi Jaafar (2014)

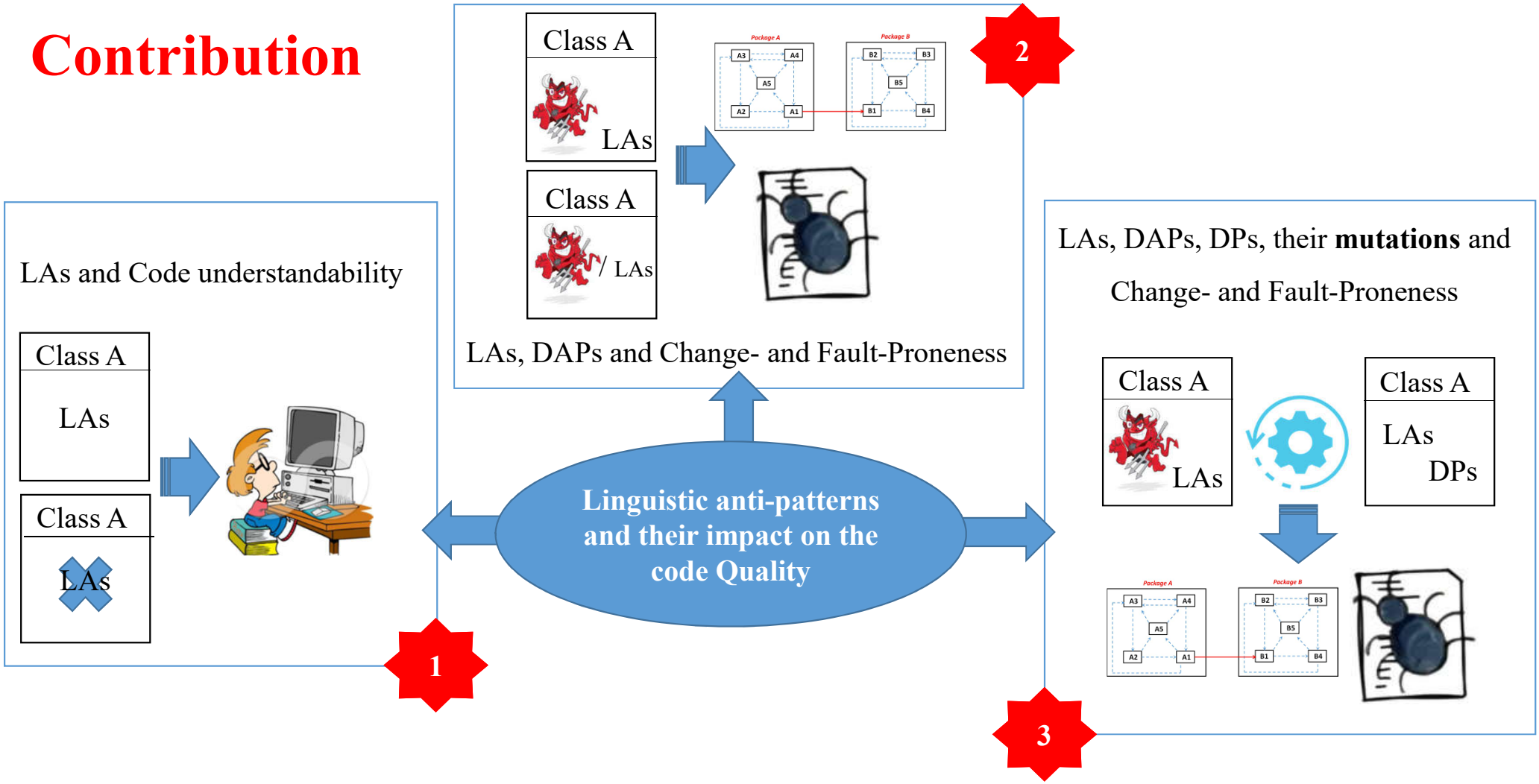
- Understanding the dynamics behind the **evolution** of **DPs and DAPs**, in particular their **mutations**, and their impact on **change- and Fault-proneness**.
- The impacts of the co-occurrence of **LAs and DPs or DAPs** on the **quality of software systems,** particularly on the **change- and fault-proneness**.

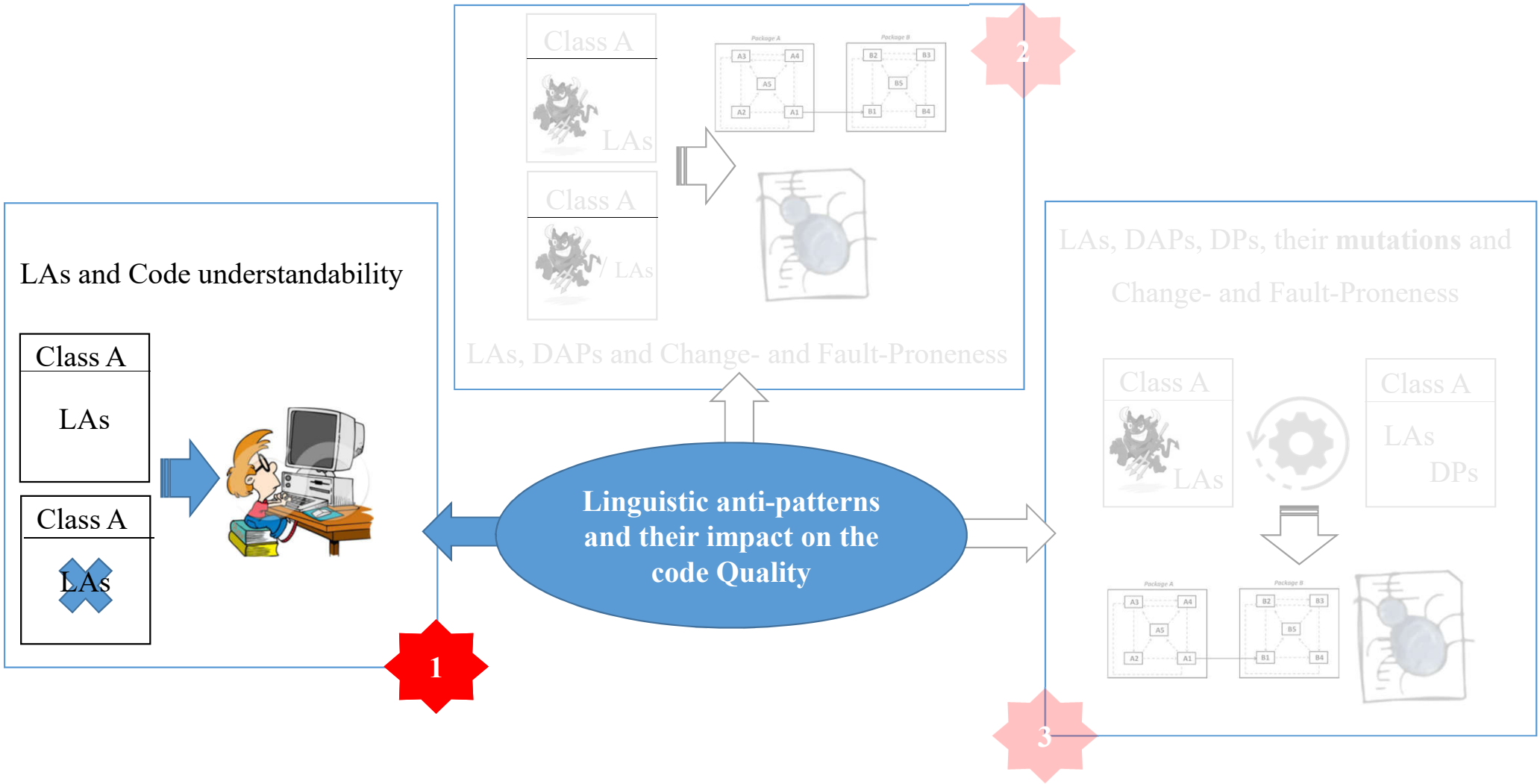
(Contribution 3)
LAs, DAPs, DPs, mutations and change- and fault-proneness

Thesis statement

LAs have a noticeable impact on the code quality.

Contribution




LAs and Code understandability
Class A
LAs
Class A
LAs
1
Class A
LAs
Class A
LAs
LAs, DAPs and Change- and Fault-Proneness
2
Linguistic anti-patterns and their impact on the code Quality
LAs, DAPs, DPs, their mutations and Change- and Fault-Proneness
Class A
LAs
Class A
LAs
DPs
3

Experiments' Definition and Planning

- Two experiments, 7 LAs, 10 studied systems, 142 participants;
- Study the impact of Different types of LAs on understandability;
- Study the impact of prior knowledge about LAs;
- Study the level of English.

Submitted to SQJ, 2019

Studied systems

	Systems	Release Date
System 1	ArgoUML 0.34 ArgoUML 0.14	2011-12-15 2003-12-05
System 2	Cocoon2.2.0	2013-03-14
System 3	JFreeChart1.0.19	2014-07-31
System 4	JHotDraw7.0.6	2011-09-06
System 5	Rhino1.7.7.2	2017-09-27
System 6	Xerces2-j2-11-0	2010-11-26
System 7	Apache Ant 1.10.1	2017-02-06
System 8	Hibernate5.2.12.Final	2017-10-19
System 9	Apache commons-lang-3.7	2017-11-08
System 10	Apache Hadoop3.0.0	2017-12-13

Studied LAs

- A2: “Is” returns more than a Boolean;
- A3: “Set” method returns;
- B4: Not answered question;
- F1: Attribute name and type are opposite;
- F2: Attribute signature and comment are opposite;
- D1: Says one but contains many;
- E1: Says many but contains one.

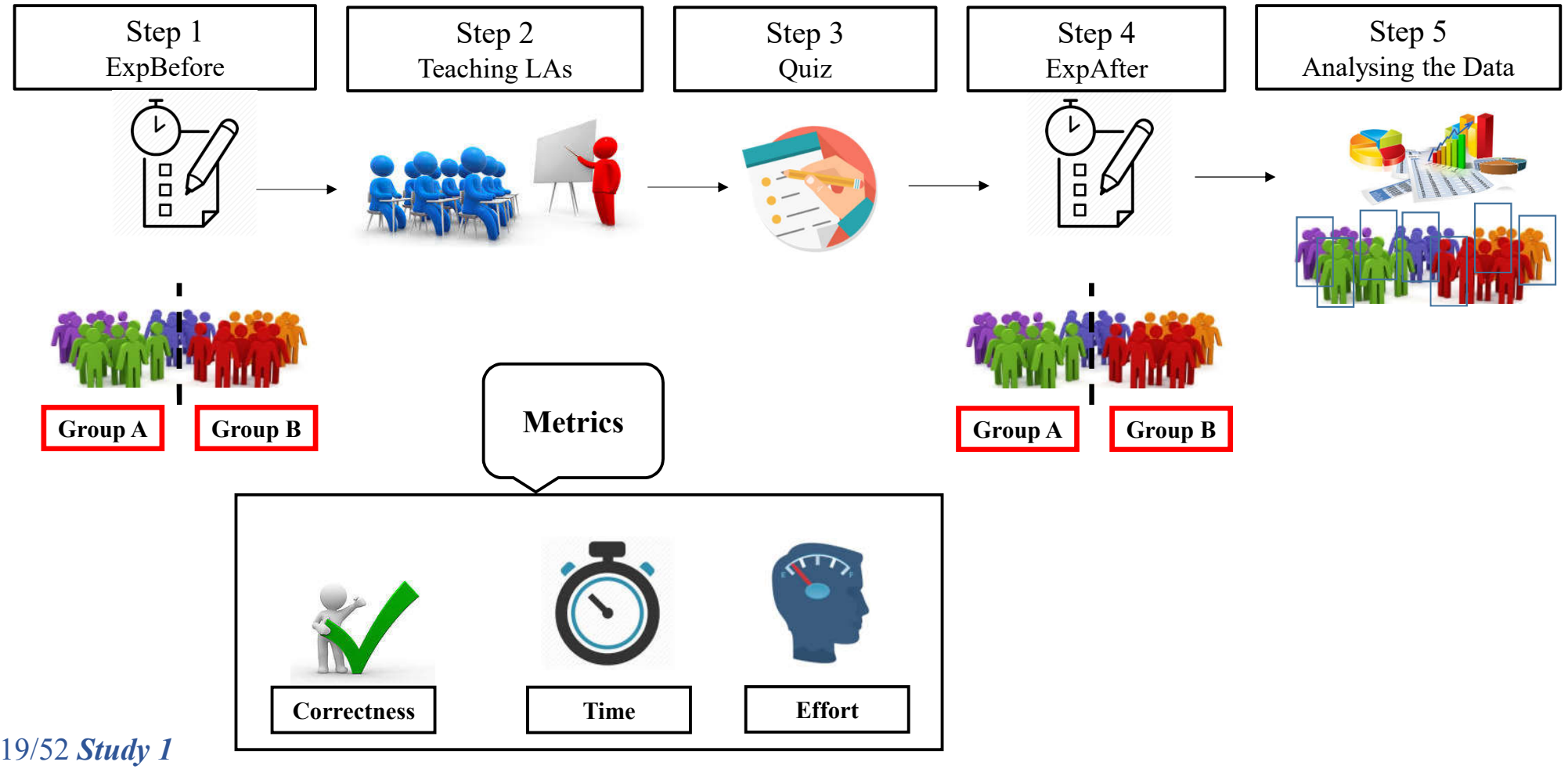
Participants

230 Participants → 142 Participants

SOEN6461 (Software Quality Engineering)

LOG8430 (Software Architecture and Advanced Design)
LOG8371 (Software Quality Engineering)

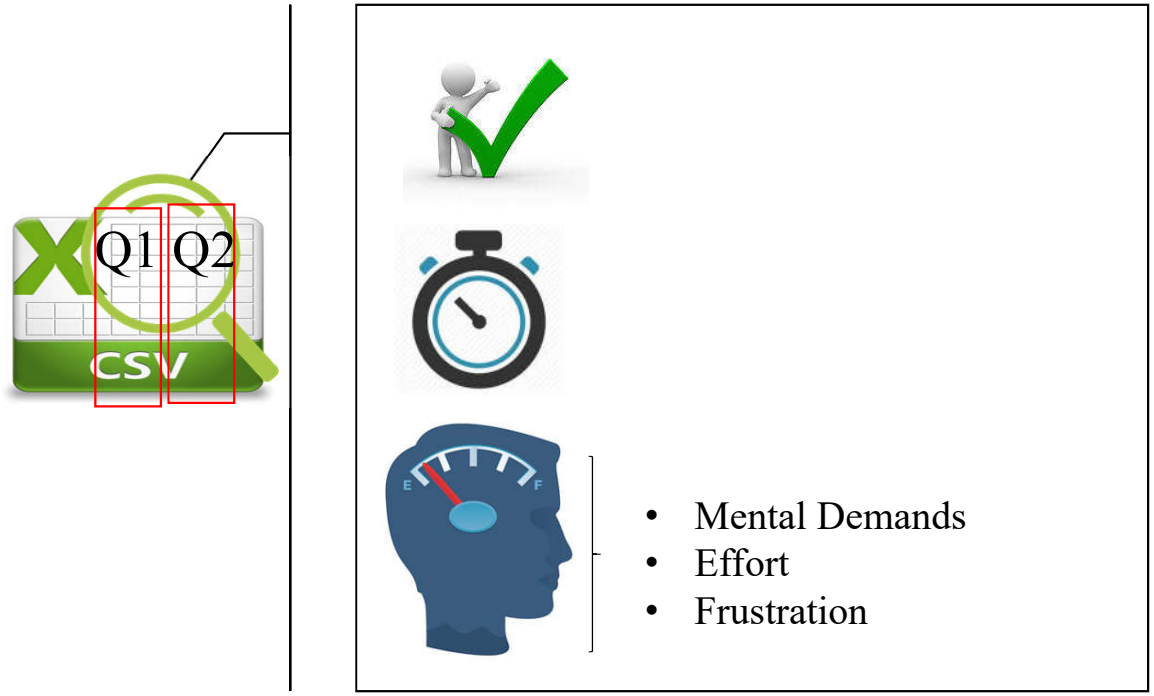
Study Design



Research Questions

LAs	Impact	Understanding
LAs types	Impact	Unknowledgeable developers
		Knowledgeable developers
Knowledge	Impact	Understanding
English level	Impact	Understanding

RQ1. Do LAs affect developers' **understanding**?



Yes. LAs affect developers' understanding **negatively**

RQ2. Do **different types** of LAs affect **unknowledgeable** developers' understandability?

ExpBefore

- A3: "Set" method returns;
- B4: Not answered question;
- D1: Says one but contains many.

Type	ExpBefore	
	Correctness(%)	Effort(mean)
A2	42.4%	57.8%
A3	4.9%	77.2%
B4	19.1%	63.3%
D1	5.7%	85%
E1	52.4%	56.5%
F1	26%	52.1%
F2	29.5%	52.9%

A3, **D1**, and **B4** have respectively the **most negative** impact on the **understandability** when participants **do not** have **knowledge** about the **LAs**.

RQ3. Do **different types** of LAs affect **knowledgeable** developers' understandability **equally**?

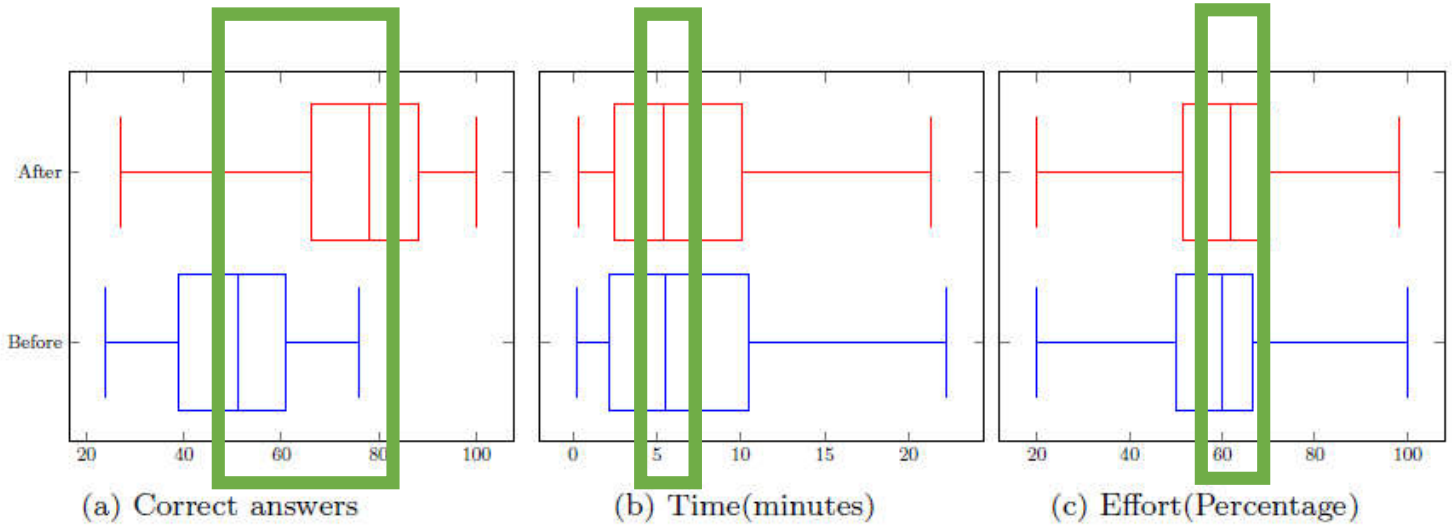
ExpAfter

- D1: Says one but contains many;
- E1: Says many but contains one;
- F1: Attribute name and type are opposite.

Type	ExpAfter	
	Correctness(%)	Effort(mean)
A2	91.5%	61.4%
A3	95.8%	52.4%
B4	97.1%	62.8%
D1	47.8%	66%
E1	52.8%	62.7%
F1	48.5%	60.6%
F2	92.9%	62.6%

D1, **F1**, and **E1** have respectively the **most negative** impact on the **understandability** when participants **have knowledge** about the **LAs**.

RQ4. Can **knowledge** about LAs **mitigate** the impact of LAs on **understandability**?



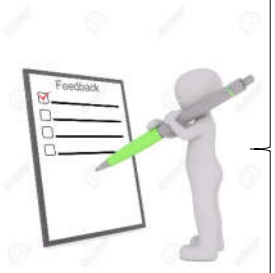
Having knowledge about LAs helps **improve** the **understandability** of code that contain LAs.

RQ5. Can **knowledge** of the language in which comments and identifiers are written **mitigate** the effect of LAs on developers' **understandability** of the code?

Activity	University	Correctness (Percentage)	Time (minute)	Effort (Percentage)
ExpBefore	Concordia	48.94%	4.27	60%
	Polytechnique	48.96%	4.12	52.33%
ExpAfter	Concordia	82.81%	4.34	61.11%
	Polytechnique	73.78%	5.22	56.66%
Quiz	Concordia	95.33%	-	-
	Polytechnique	86.05%	-	-

The **proficiency** in the **Language** in which identifiers and comments are written can have a **slightly positive impact** on understandability.

Other Factors impact on the dependent variables

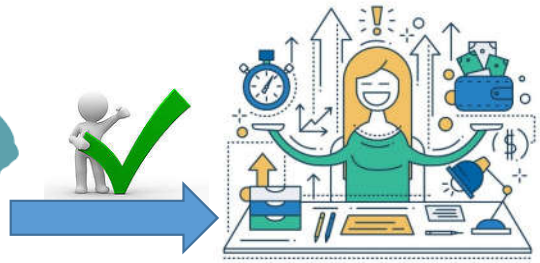


Independent Variables	Correctness (p-value)	Time (p-value)	Effort (p-value)
Age	0.13	0.08	0.52
Gender	0.21	0.52	0.92
Degree	**0.03**	0.41	**0.01**
Programming Knowledge	**0.06**	0.36	**< 0.01**
Working Experience	0.71	0.72	0.62

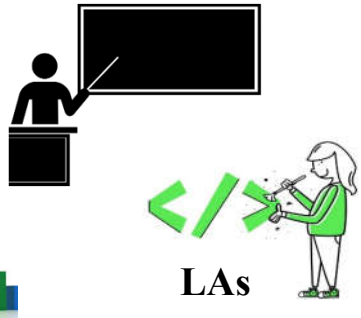
Programming knowledge and **education** has a statistically significant impact on **correctness** and **effort**. We suggest companies to take into account **such profiles** for specific job positions.

LAs ↓ code understandability

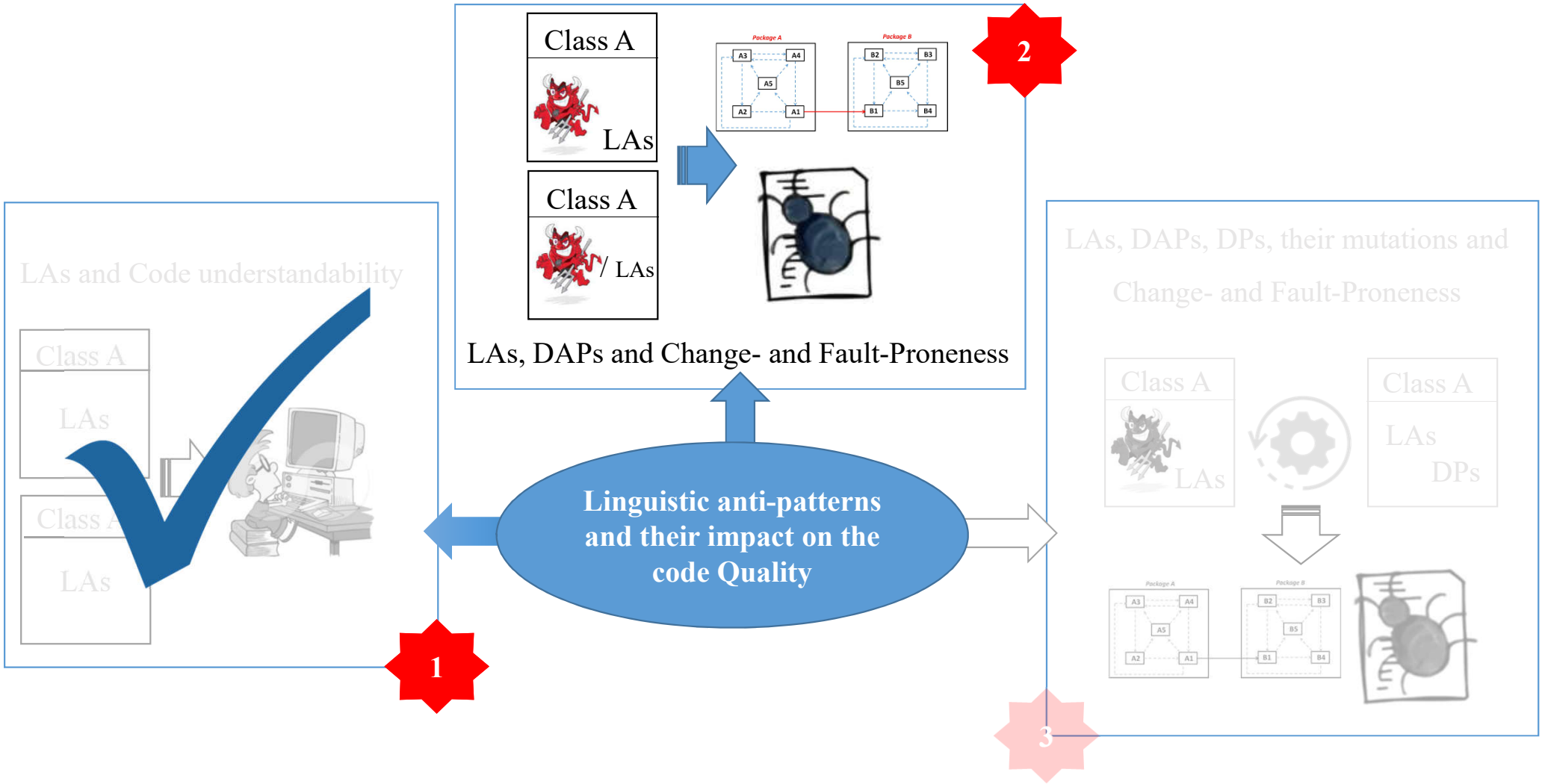
↑ code understandability



Development Team



LAs


Class A
LAs
Class A
/ LAs
LAs, DAPs and Change- and Fault-Proneness
2
LAs and Code understandability
Class A
LAs
Class A
LAs
1
Linguistic anti-patterns and their impact on the code Quality
LAs, DAPs, DPs, their mutations and Change- and Fault-Proneness
Class A
LAs
Class A
LAs
DPs
3

Experiments' Definition and Planning

- 12 DAPs, 17 LAs, 3 studied systems;
- Study the impact of classes containing LAs on change-proneness;
- Study the impact of classes containing LAs on Fault-proneness.

Published in SQJ, 2016

Studied DAPs and LAs

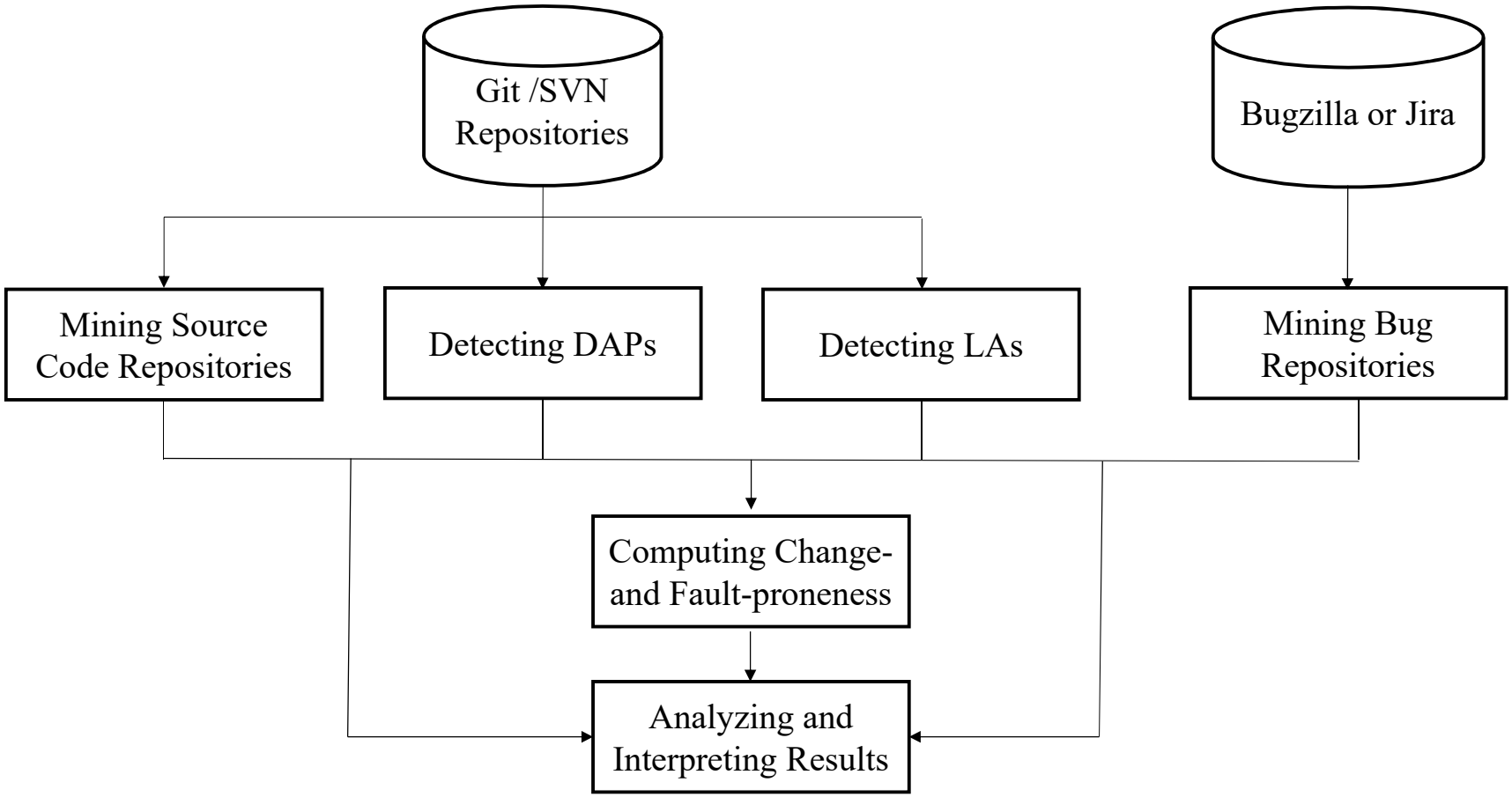
- ❑ AntiSingleton;
- ❑ GodClass (Blob);
- ❑ ClassDataShouldBePrivate;
- ❑ ComplexClass;
- ❑ LargeClass;
- ❑ LazyClass;
- ❑ LongMethod;
- ❑ LongParameterList;
- ❑ MessageChain;
- ❑ RefusedParentBequest;
- ❑ SpeculativeGenerality;
- ❑ SwissArmyKnife.

& All Types of LAs

Studied Systems

System	# Releases	Sizes (LOCs)	# Classes
ANT	7	1,600,256	14,052
ArgoUML	13	644,829	27,822
Hibernate	10	7,239,075	21,876

Study Design



Research Questions

LAs and DAPs Impact Change-proneness

LAs and DAPs Impact Fault-proneness

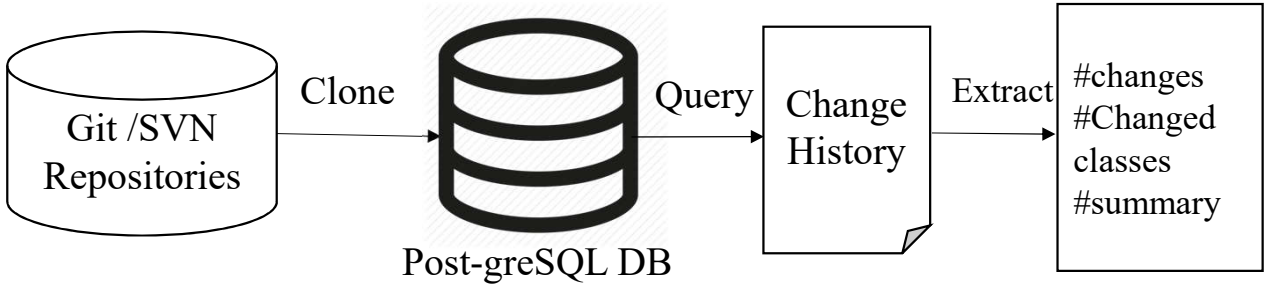
RQ1. Are classes with a particular family of smells (**DAPs**, **LAs**, or **both**) more **change-prone** than others?

1. Classes containing both DAPs and LAs versus classes with DAPs
   p q

2. Classes having DAPs and LAs versus classes containing LAs
   p q

3. Classes containing DAPs versus classes with LAs
   p q

$$OR = \frac{p/(1-p)}{q/(1-q)} > 1$$

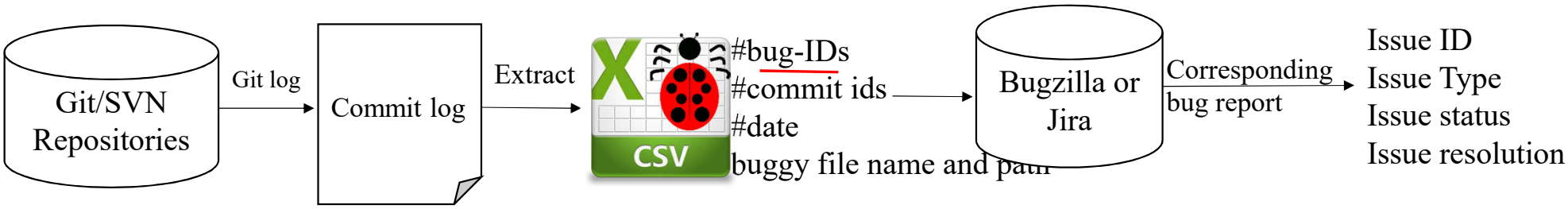


DAPs contribute more to the **change-proneness** of **LAs** classes than **LAs** do to the change-proneness of **DAP** classes.

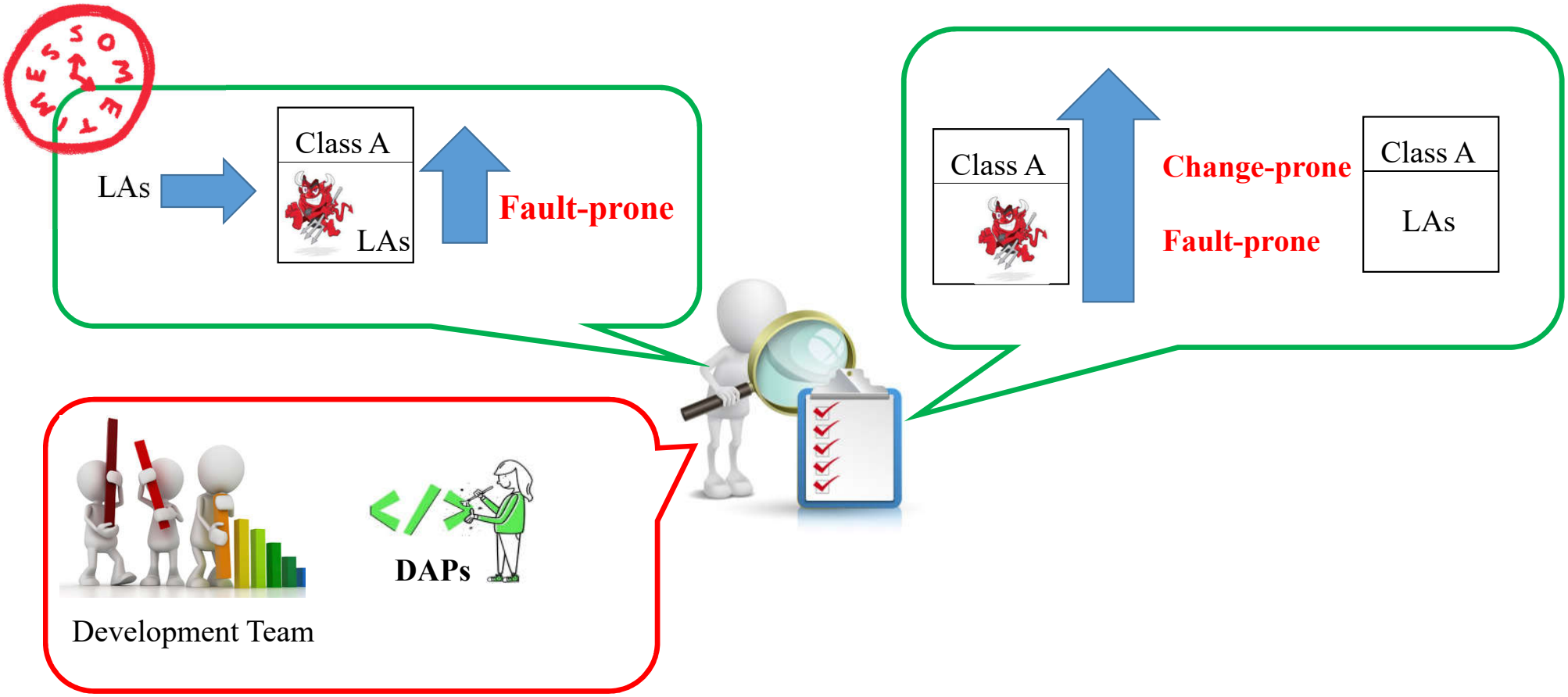
RQ2. Are classes with a particular family of smells (**DAPs**, **LAs**, or **both**) more **fault-prone** than others?

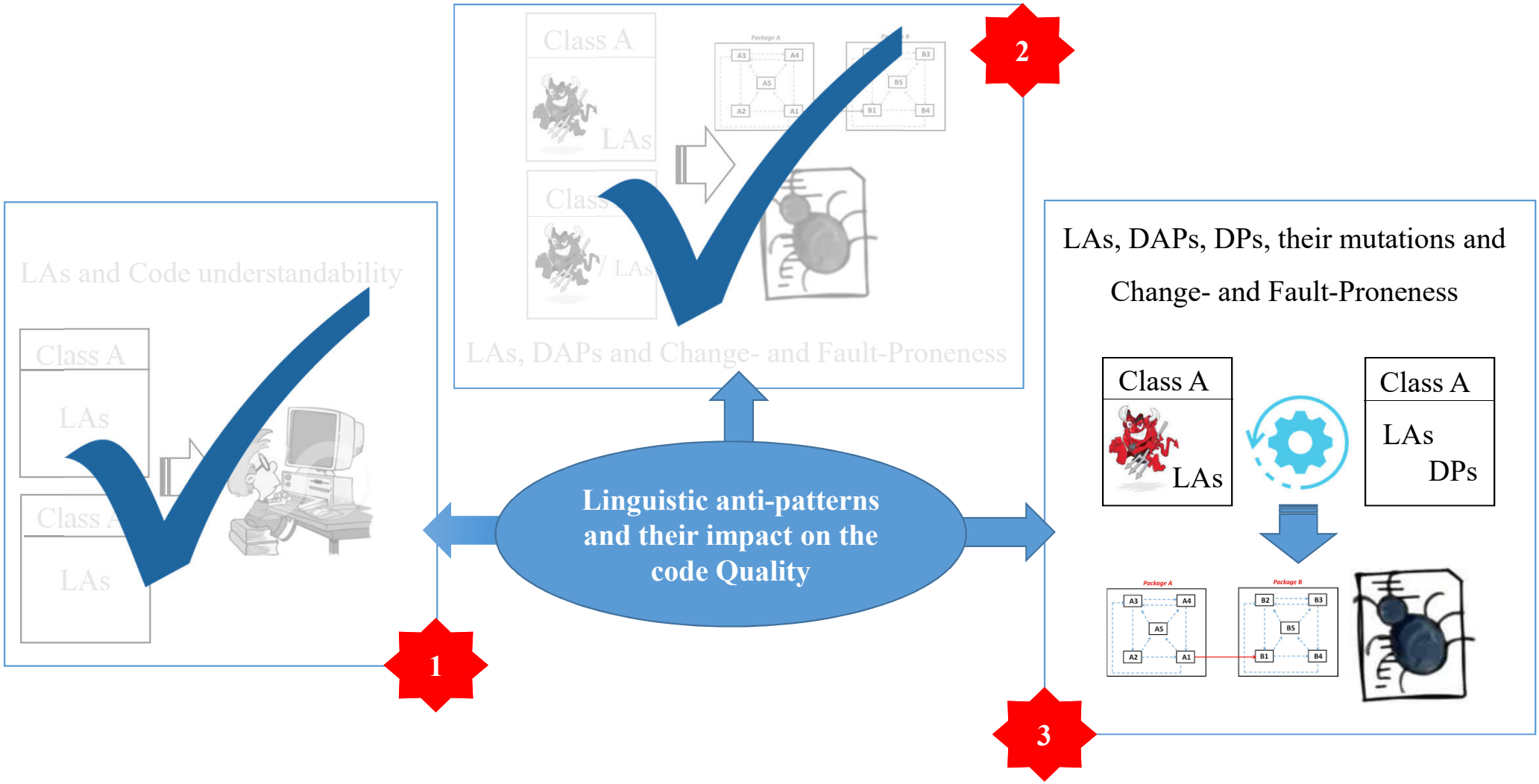
1. Classes containing both DAPs and LAs (p) versus classes with DAPs (q)
2. Classes having DAPs and LAs (p) versus classes containing LAs (q)
3. Classes containing DAPs (p) versus classes with LAs (q)

$$OR = \frac{p/(1-p)}{q/(1-q)} > 1$$



The occurrence of **DAPs** in a class that experienced a **LAs** has a strong relationship with **fault-proneness** than the occurrence of **LAs** in a class that experienced a **DAPs**.


LAs
Class A
LAs
Fault-prone
Class A
Change-prone
Fault-prone
Class A
LAs
DAPs
Development Team

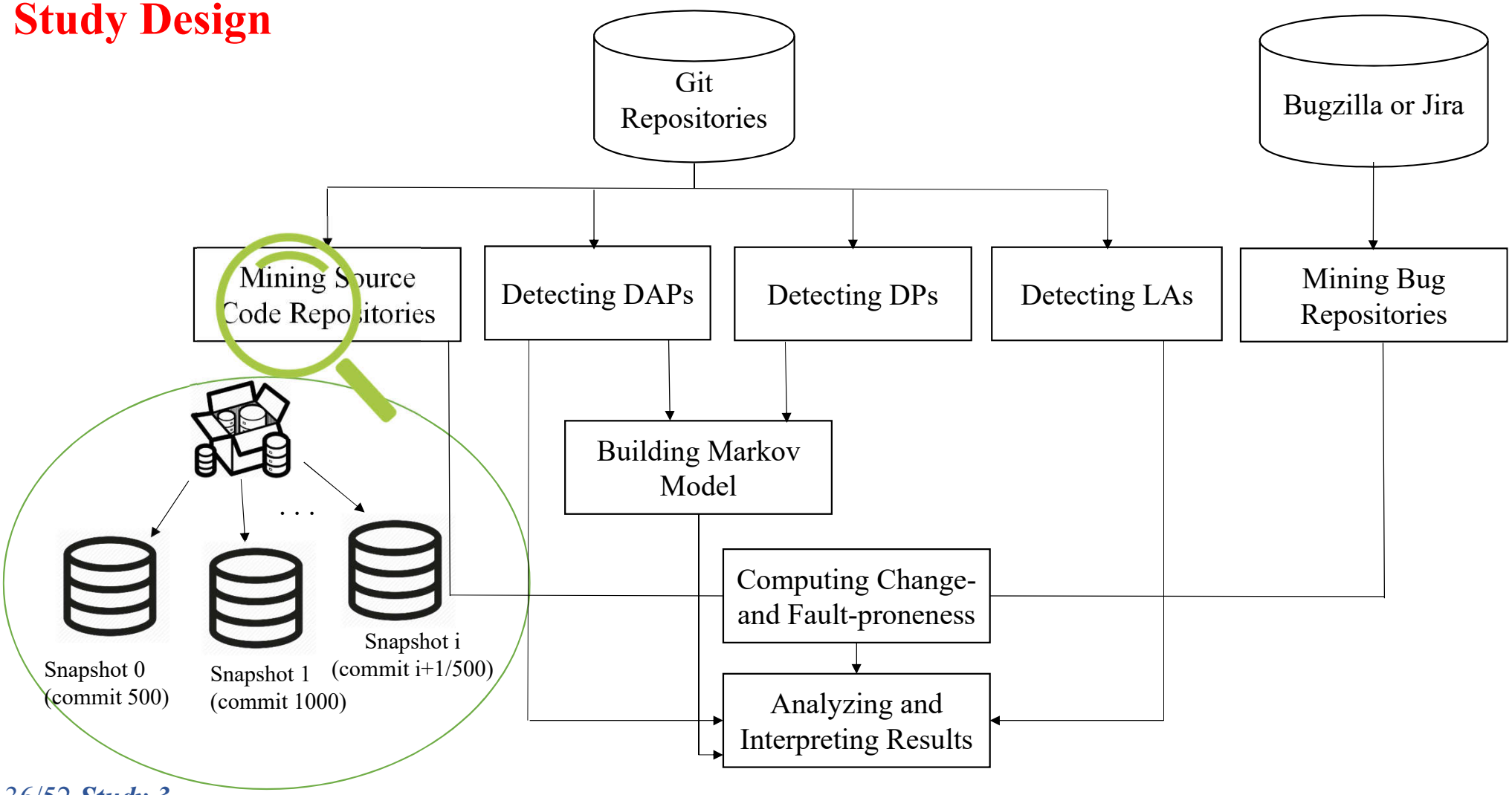

LAs and Code understandability
Class A
LAs
Class A
LAs
1
Class A
LAs
Class
LAs
LAs, DAPs and Change- and Fault-Proneness
2
Linguistic anti-patterns and their impact on the code Quality
LAs, DAPs, DPs, their mutations and Change- and Fault-Proneness
Class A
LAs
Class A
LAs
DPs
Package A
A3
A4
A5
A2
A1
Package B
B2
B3
B5
B1
B4
3

Experiments' Definition and Planning

- 13 DAPs, 8 DPs and 17 LAs, 7 studied systems;
- DPs and-or DAPs mutations;
- Impact of these mutations on change- and fault-proneness;
- Impact of LAs and DPs/DAPs on change- and fault- proneness.

Submitted to EMSE, 2019

Study Design



Studied LAs, DAPs and DPs

- ❑ AntiSingleton
- ❑ GodClass (Blob)
- ❑ ClassDataShouldBePrivate
- ❑ ComplexClass
- ❑ LargeClass
- ❑ LazyClass

- ❑ LongMethod
- ❑ LongParameterList
- ❑ MessageChain
- ❑ RefusedParentBequest
- ❑ SpaghettiCode
- ❑ SpeculativeGenerality
- ❑ SwissArmyKnife

& All Types of LAs

- ❑ Builder
- ❑ Command
- ❑ Composite
- ❑ Decorator
- ❑ Factory Method
- ❑ Observer
- ❑ Prototype
- ❑ Singleton

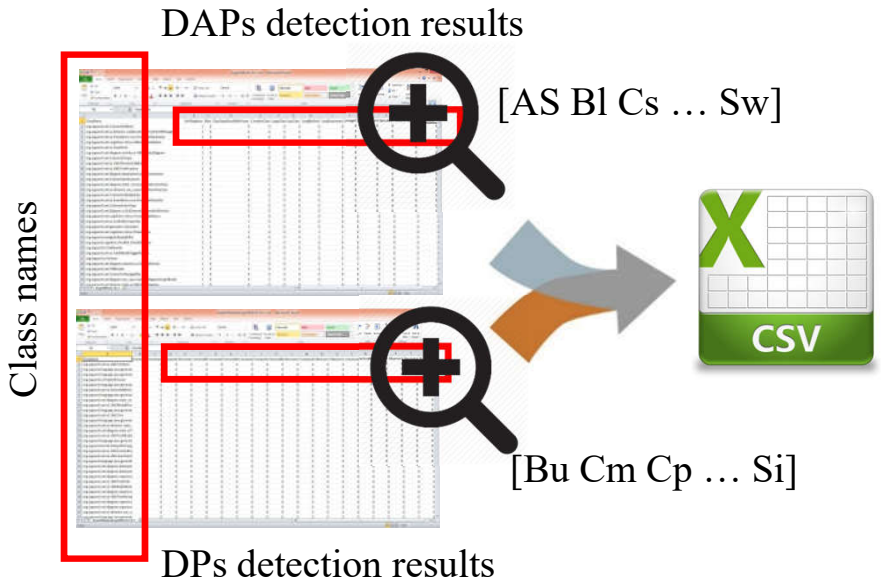
Studied Systems

System	# Commits	Sizes (LOCs)	Issue Tracker
Eclipse for Java	281,396	9,064,794	Bugzilla
Nuxeo Platform	265,380	5,741,131	Jira
oVirt	149,128	2,764,655	Bugzilla
Matsim	44,200	1,602,877	Atlassian
Apache Solr	30,995	652,711	Jira
Apache Ignite	24,104	1,471,036	Jira
Mule Community Edition	22,891	309,616	Jira

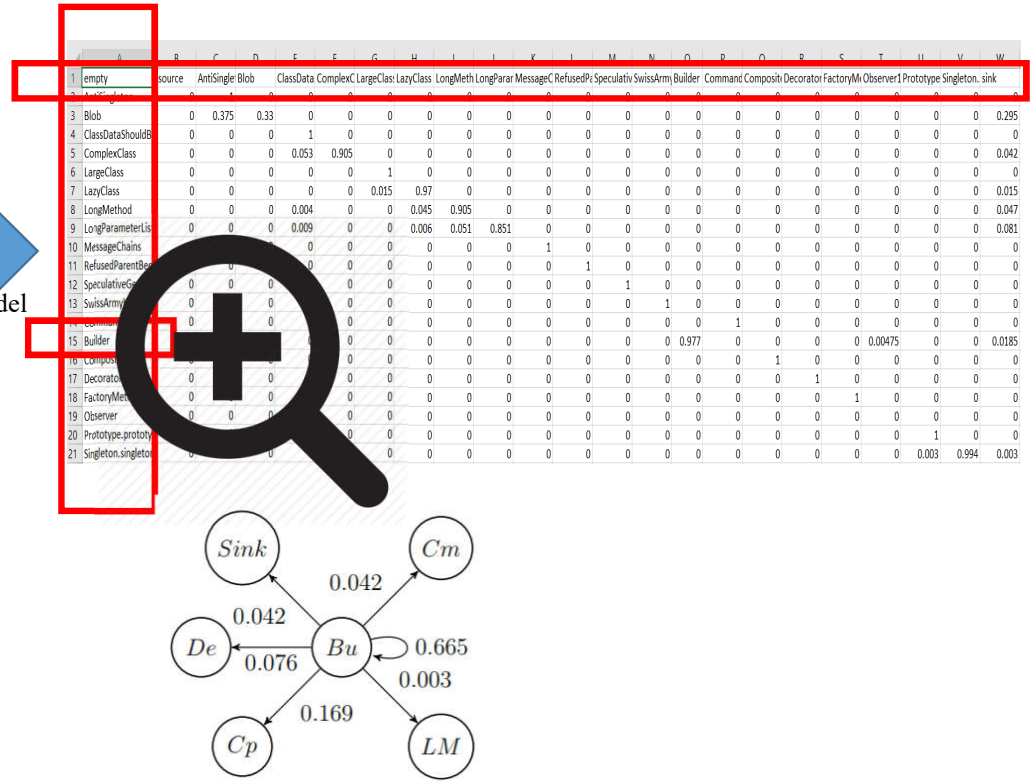
Research Questions

DAPs, DPs	Probability	Mutation
Change types	DAPs, DPs	Mutation
DAPs, DPs mutation	Impact	Change- and Fault-proneness
Change types	Mutation	Change- and Fault-proneness
LAs	Impact	Change- and Fault-proneness

RQ1. Do **DPs** and–or **DAPs mutate** during the evolution of software systems? What is the **probability** of occurrence of different types of **mutations**?



Markov Model



Builder Mutation among the different revisions of Matsim system

RQ1. Do **DPs** and–or **DAPs mutate** during the evolution of software systems? What is the **probability** of occurrence of different types of **mutations**?

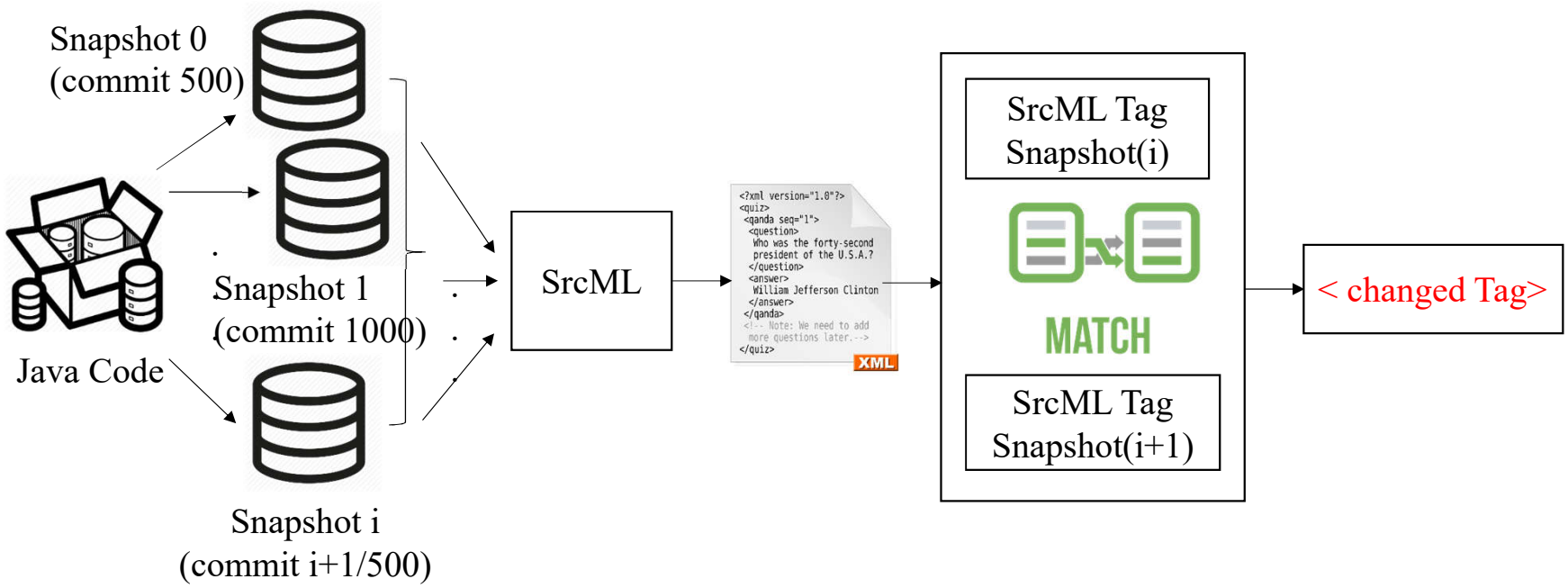
	Source	AS	Bl	CS	CC	LC	LZC	LM	LP	MCh	RP	SC	SG	SA	Bu	Cm	Cp	FM	De	Ob	PT	Si	Sink
AS	0.032	**0.937**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.030	0.000	0.000	0.000	0.000	0.000
Bl	0.000	**0.313**	**0.313**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.374**	0.000	0.000	0.000	0.000	0.000

Yes!

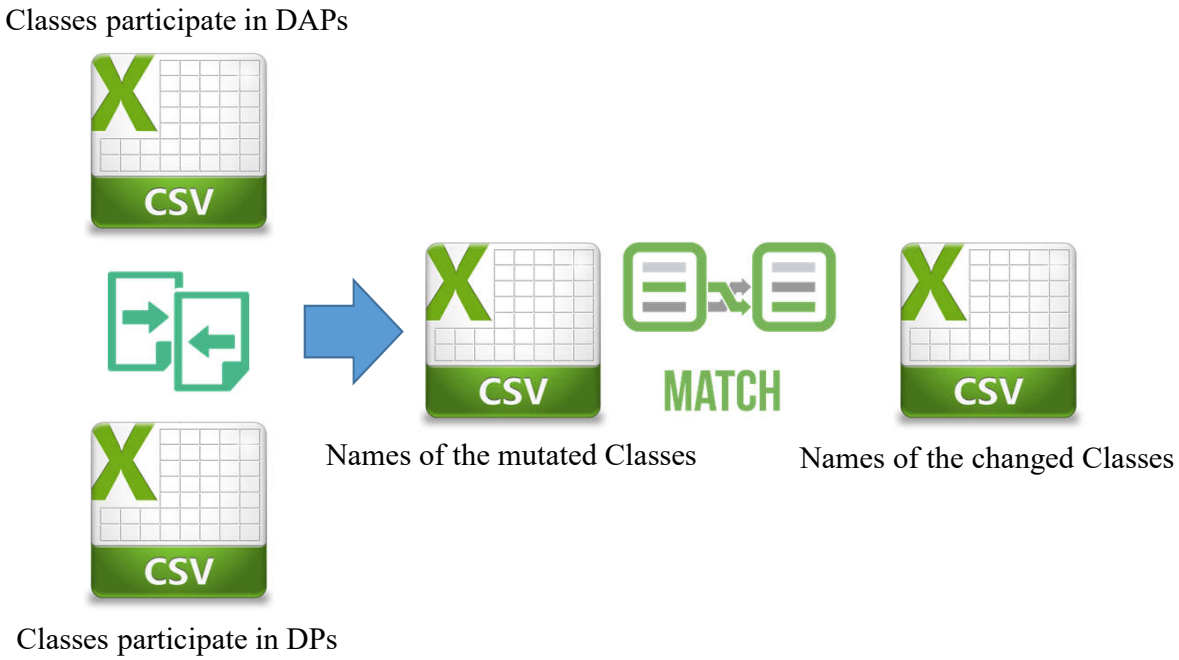
- **DPs** and **DAPs** mutate during the evolution of software systems.
- In most of the studied systems, more than half of the **DAP** occurrences mutated during the evolution.
- In most of the systems, almost all the **DPs** occurrences remained stable during the evolution process.
- **Blob** and **Command** are the **DAPs** and **DPs**, which have higher mutation probabilities.

Ob	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.024	0.024	**0.951**	0.000	0.000	0.000
PT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**1**	0.000	0.000
Si	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.029	0.000	0.000	0.006	**0.964**	0.000

RQ2. What types of **changes** lead to a **mutation** between **DPs** and-or **DAPs**?

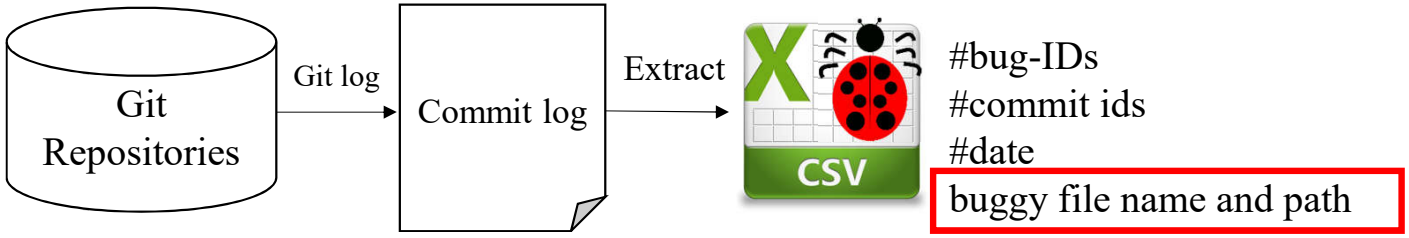


RQ2. What types of **changes** lead to a **mutation** between **DPs** and-or **DAPs**?



In general, some of the change types affect the mutation from **DPs** and–or **DAPs**. The most representative change types leading to mutations in all the studied systems are "**Renaming**", "**Comment**", "**Declaration**", and "**Operator**".

RQ3. What is the **fault-proneness** of mutated **DPs** and **DAPs**? What transitions lead to more **fault-prone mutations**?



DAPs are more **fault-prone** than **DPs**. Mutations from **DAPs** to **DPs** are more **faulty** than other types of mutations.

RQ4. Do specific types of **changes** lead to increase **fault-proneness** during **DPs** and-or **DAPs mutations**?

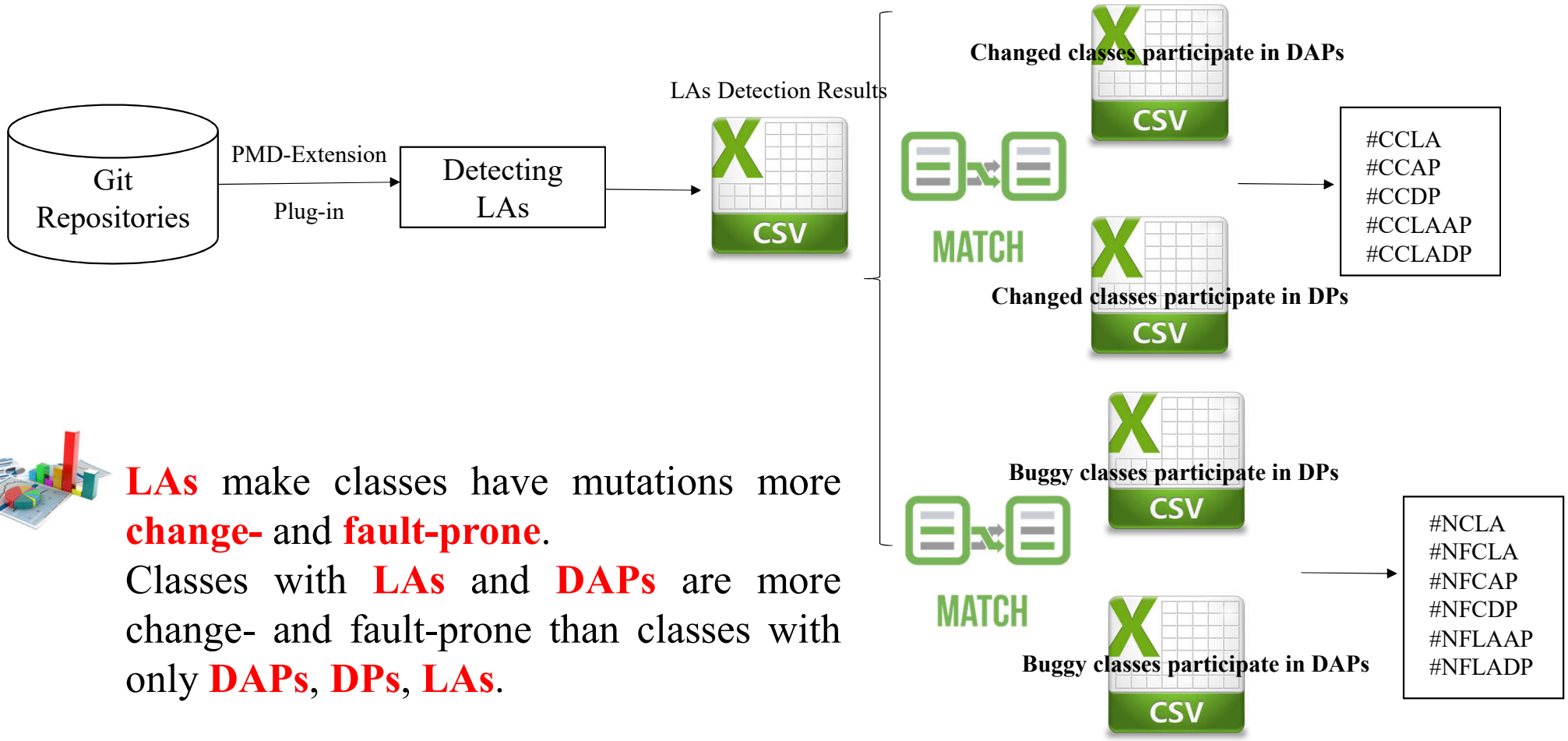
buggy file name and path

changed class names

Systems →	Apache Ignite	Apache Solr	Eclipse	Matsim	Mule	Nuxeo	oVirt
Change Types ↓	# of changes	# of changes	# of changes	# of changes	# of changes	# of changes	# of changes
Access	18	18	37	22	11	62	25
Class	689	431	306	306	422	208	763
Code block	2,972	2,102	4,919	1,284	1,306	854	3,833
Comment	12,169	5,498	38,150	2,813	6,270	6,161	4,653
Control flow	2,903	1,935	11,678	660	1067	819	2,406
Declaration	5,912	5,386	11651	3,129	3,191	2,628	7,484
Exception	1,696	1221	1255	140	526	786	210
Import	2,831	2,400	2,958	1,425	2,443	2,064	4,268
Invocation	1,550	1,196	1,986	1,061	840	476	1,882
Method	2,509	1,851	4,093	637	1,697	1,229	3,619
Operator	5,120	4,364	16,094	6,215	4,228	2,675	8,134
Parameter	6,418	3,504	5,108	2,655	2,607	1,815	5,337
Renaming	47,811	24,640	67,040	32,445	22,968	12,736	65,245
Total changed classes	5,505	4,163	11,934	3,073	4,324	3,514	7,150

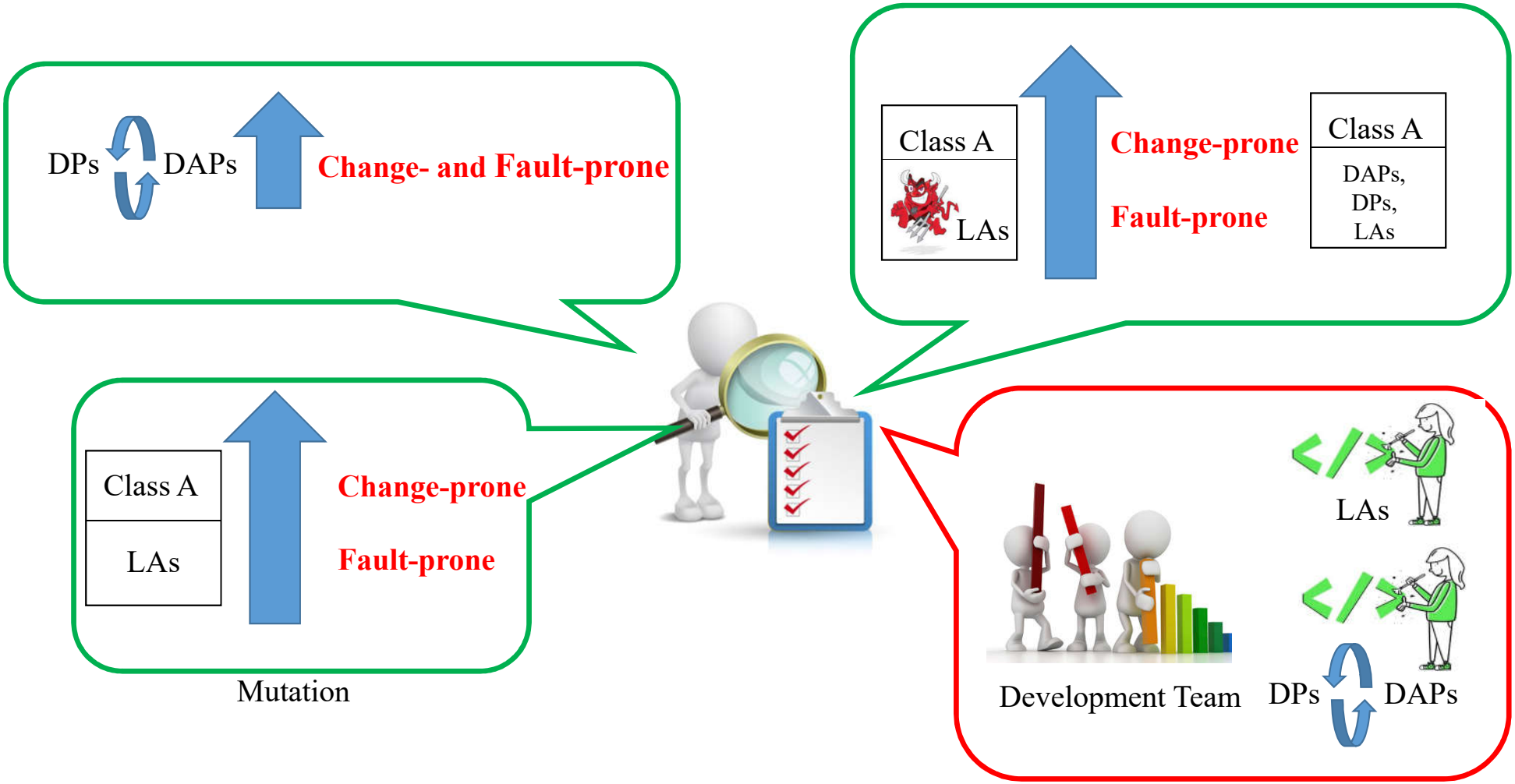
"**Renaming**", "**Comment**", and "**Operator**" are the **most prevalent change types** that lead to **faults**.

RQ5. Do the occurrences of **LAs** increase **change-** and **fault-proneness** during **DPs** and-or **DAPs**?



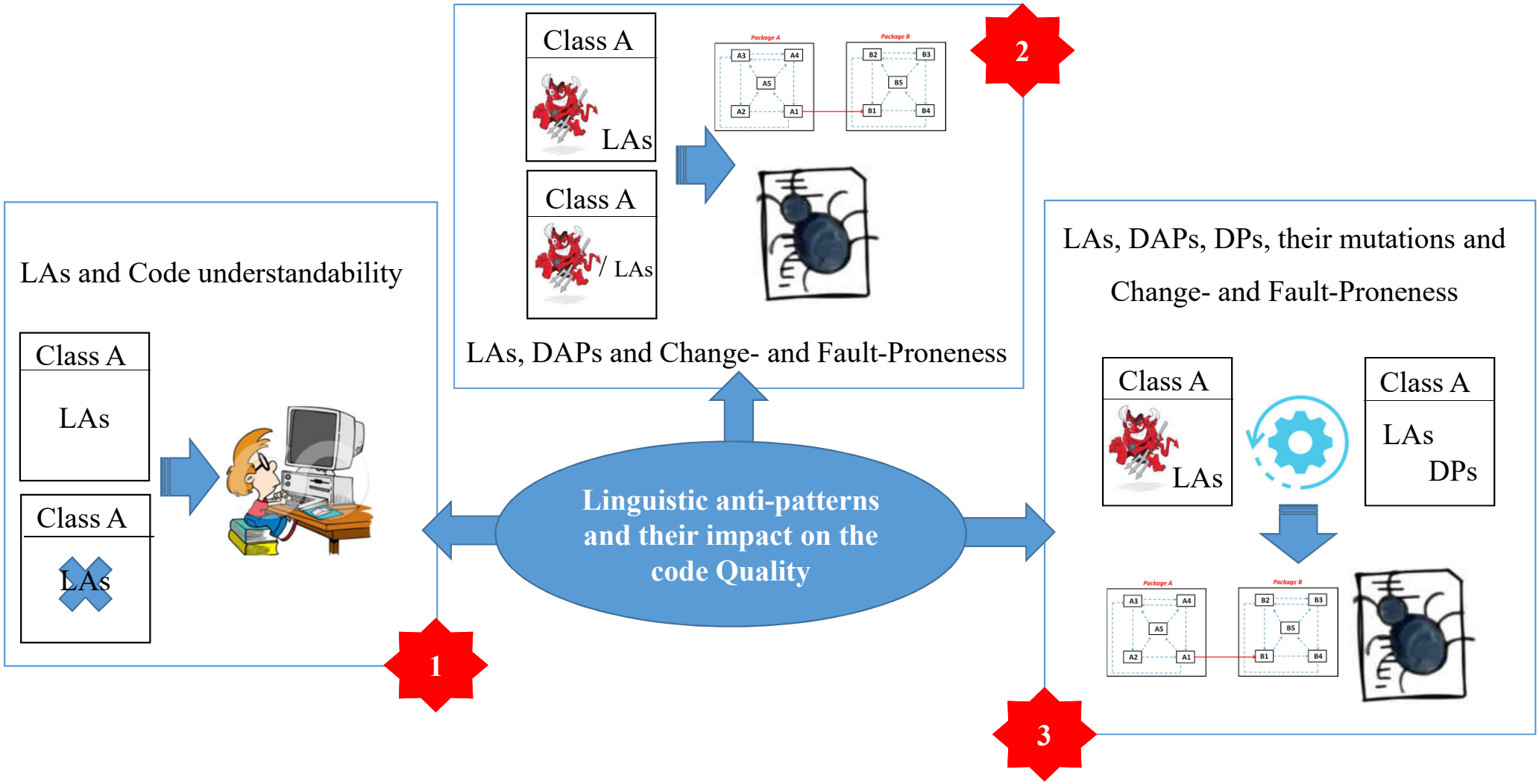
LAs make classes have mutations more **change-** and **fault-prone**.

Classes with **LAs** and **DAPs** are more change- and fault-prone than classes with only **DAPs**, **DPs**, **LAs**.


DPs
DAPs
Change- and Fault-prone
Class A
LAs
Change-prone
Fault-prone
Class A
DAPs,
DPs,
LAs
Class A
LAs
Change-prone
Fault-prone
Mutation
LAs
Development Team
DPs
DAPs

Thesis statement

LAs have a noticeable impact on the code quality.


Class A
LAs
Class A
LAs
LAs and Code understandability
1
Class A
LAs
Class A
/ LAs
LAs, DAPs and Change- and Fault-Proneness
2
Linguistic anti-patterns and their impact on the code Quality
LAs, DAPs, DPs, their mutations and Change- and Fault-Proneness
Class A
LAs
Class A
LAs
DPs
3

Future Direction

❑ **Study 1**

- Studying other types of LAs;
- Using an eye-tracking system;
- Improving LAs detection tool.

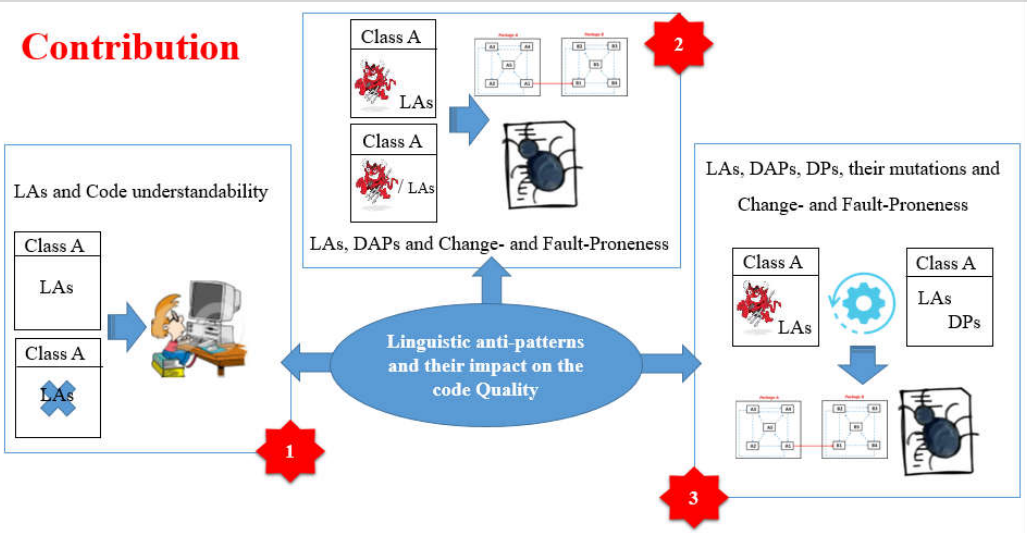
❑ **Study 2**

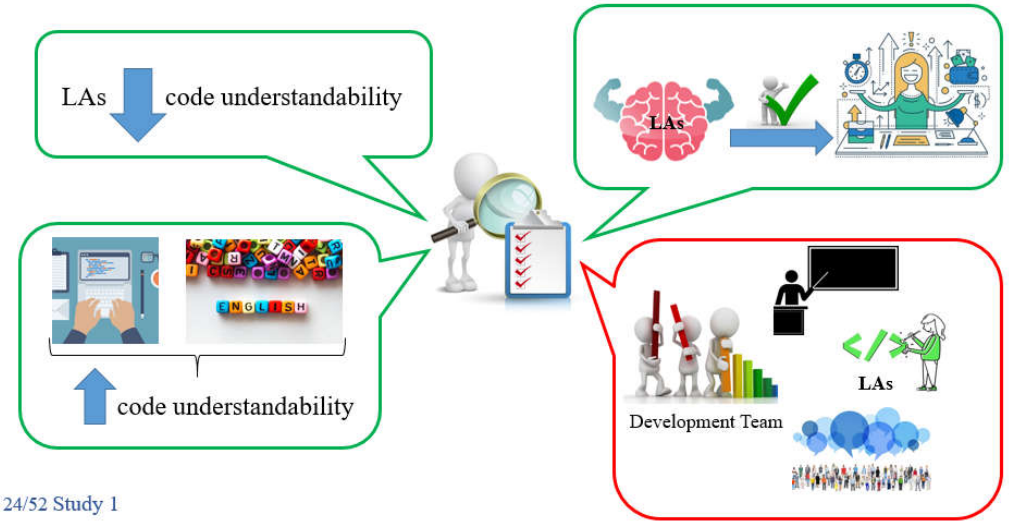
- Conducting a user study involving professional developers;
- Identifying specific type of LAs, and DAPs impact more on change- and fault-proneness.

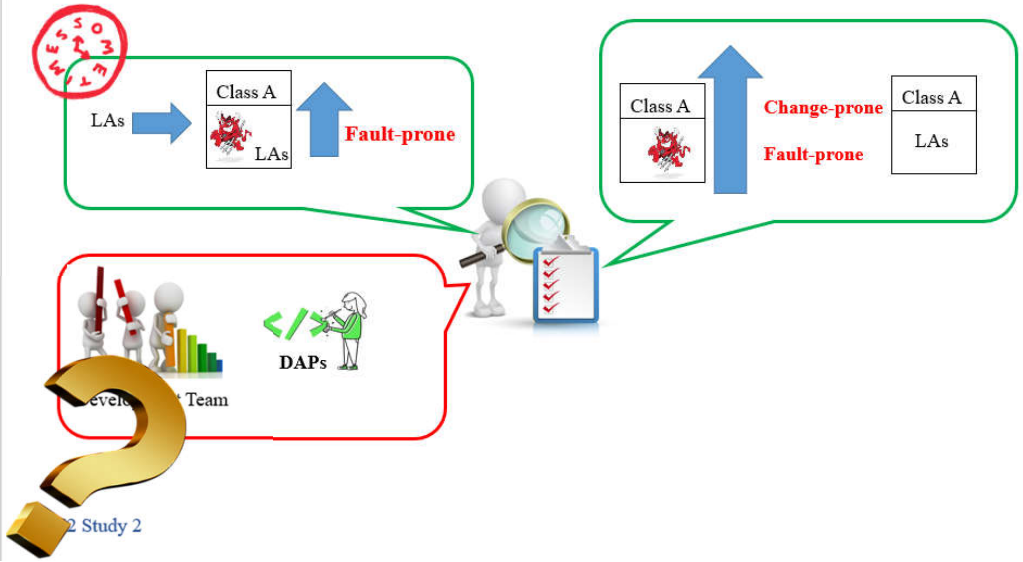
❑ **Study 3**

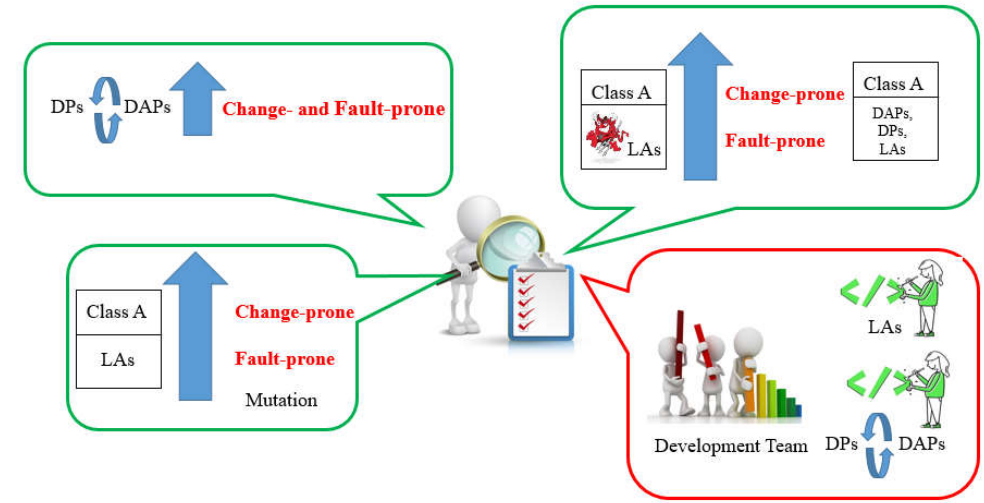
- Identifying the reason for the emergence of faults after mutation;
- Building Markov models for LAs mutation to DAPs and DPs;
- Studying such mutations impact on change- and fault-proneness;
- Specifying the type(s) of LAs, DAPs and DPs may leads to become high severity fault.

Contribution









LAs Categories

Methods		Attributes	
Name	**Description**	**Name**	**Description**
A.1	"Get"- more than an accessor	D.1	Says one but contains many
A.2	"Is" returns more than a Boolean	D.2	Name suggests Boolean but type does not
A.3	"Set" method returns	E.1	Says many but contains one
A.4	Expecting but not getting a single instance	F.1	Attribute name and type are opposite
B.1	Not implemented condition	F.2	Attribute signature and comment are opposite
B.2	Validation method does not confirm		
B.3	"Get" method does not return		
B.4	Not answered question		
B.5	Transform method does not return		
B.6	Expecting but not getting a collection		
C.1	Method name and return type are opposite		
C.2	Method signature and comment are opposite		

Do more than they say

Do less than they say

Do the opposite of what they say

Name says more than the entity contains

Name says less than the entity contains

Name says the opposite of what the entity contains

LAs Examples

B4. Example of Not answered question (Eclipse-1.0)

```
public void isValid(
        Object[] selection, StatusInfo res) {
  // only single selection
  if (selection.length == 1
        && (selection[0] instanceof IFile))
    res.setOK();
  else res.setError(""); //$NON-NLS-1$
}
```

LAs Examples

E1. Example of Says many but contains one (ArgoUML-0.10.1)

```
private static boolean _stats = true;
```

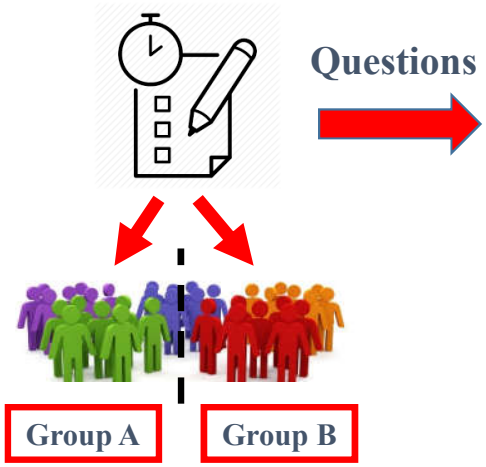
D1. Example of Says one but contains many (ArgoUML-0.10.1)

```
Vector _target;
```

F1. Example of Attribute name and type are opposite (ArgoUML0.10.1)

```
MAssociationEnd start = null;
```

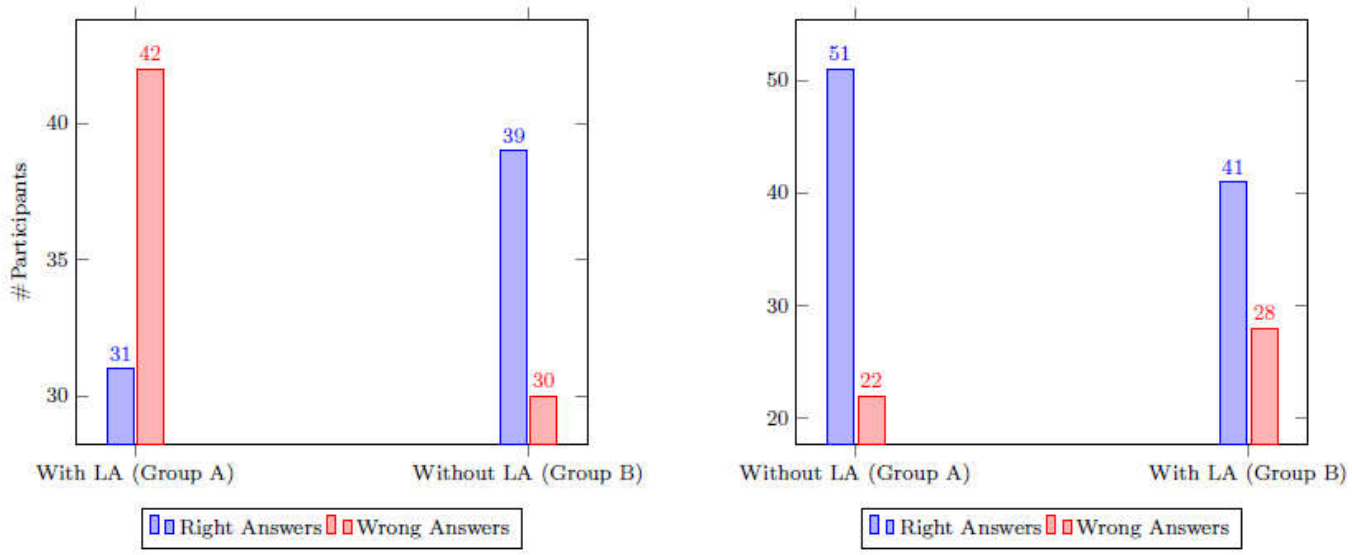
Questions



Question	ExpBefore		ExpAfter	
	Group A	Group B	Group A	Group B
Q1	A2 (with)	A2 (without)	A3 (with)	A3 (without)
Q2	E1 (without)	E1 (with)	B4 (without)	B4 (with)
Q3	F1	F1	D1	D1
Q(4.a)	Correct	A3	E1	E1
Q(4.b)	B4	D1	F2	F2
Q(4.c)	Correct	Correct	F1	F1
Q(4.d)	F2	Correct	A2	A2
Q(4.e)	A3	F2	A3	A3

http://www.ptidej.net/downloads/replications/sqj19a/Questionnaire/

RQ1. Do LAs affect developers' **understanding**?

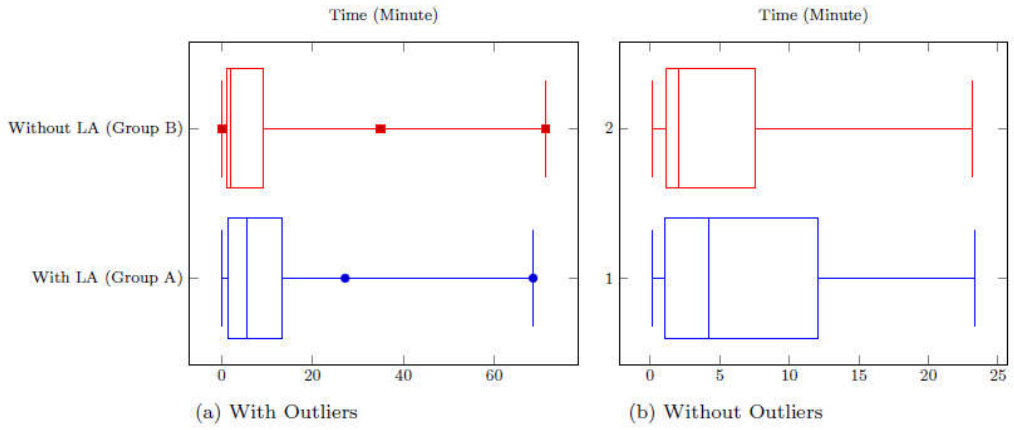


(a) (Q1:A2) (b) (Q2:E1)

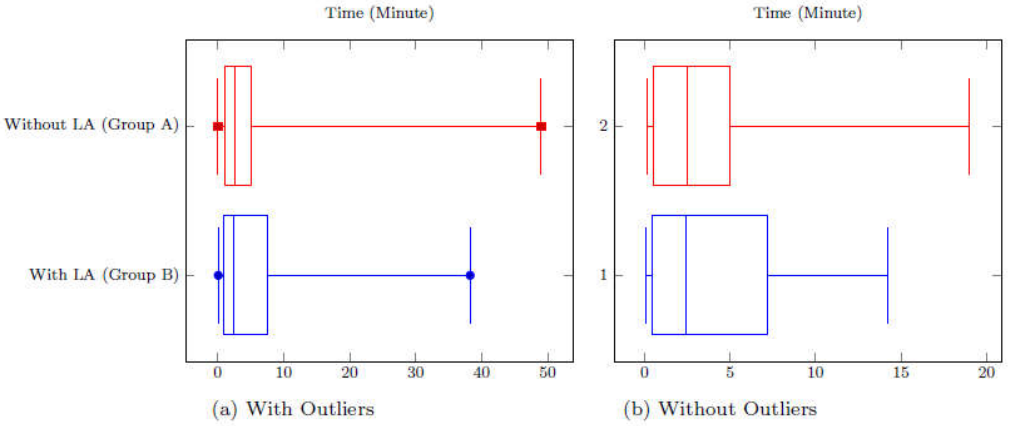
Impact of occurrence of LA on "Correctness"

Correctness of the answers

RQ1. Do LAs affect developers' **understanding**?



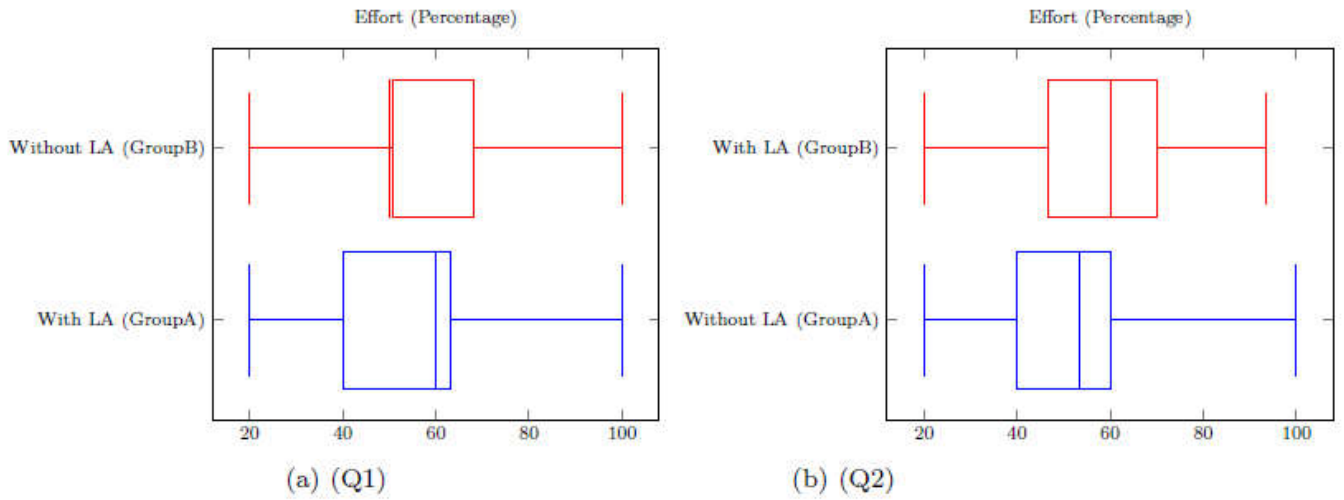
Impact of the occurrence of LA on "Time"(Q1)



Impact of the occurrence of LA on "Time"(Q2)

Speed of code understanding

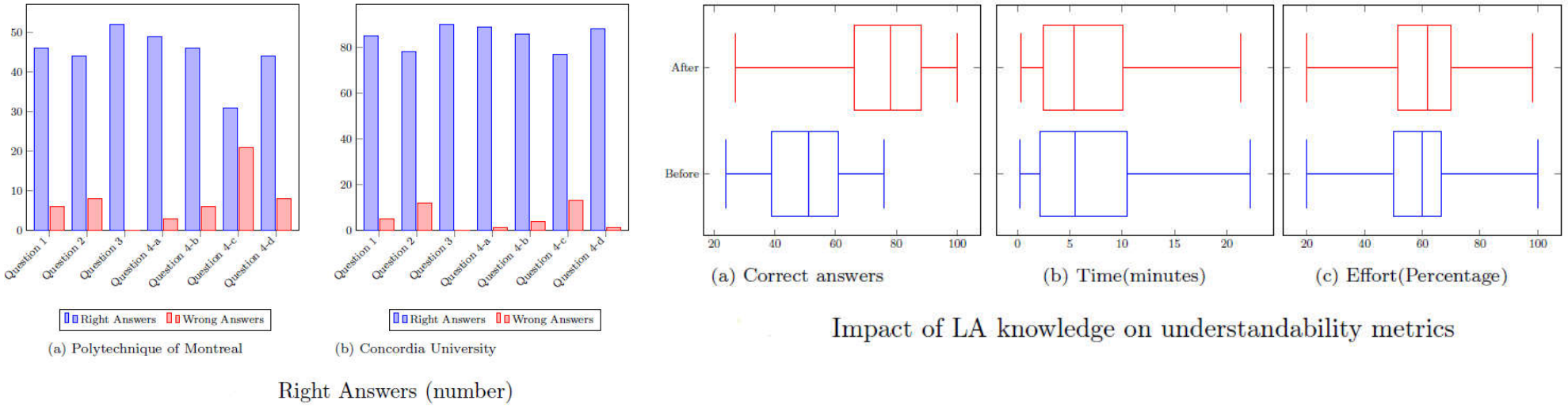
RQ1. Do LAs affect developers' **understanding**?



Impact of the occurrence of LA on "Effort"

Effort for code understanding

RQ4. Can **knowledge** about LAs **mitigate** the impact of LAs on **understandability**?



Having knowledge about LAs helps **improve the understandability** of code that contain LAs.

Therefore, **teaching developers about LAs** can help mitigate the negative impact of LAs.

Discussion

- D1 – "Says one but contains many":

We observed that developers prefer to choose "simple" names, like "tmp" for stack arrays, "x" and "y" for arrays of dimensions, or "v" for a collection of vectors.
Exception: Do not identify such names as linguistic anti-patterns.

- E1 – "Says many but contains one":

When the type of an attribute is "int", our tool expected to have a singular name because "int" is a single type. we found that the names of variables that hold number of these attributes could be plural because they are numeric of things.

Example of an exception of "E1" (JFreeChart 1.0.19)

```
// ** The maximum number of lines for category labels. */
                private int maximumCategoryLabelLines;
```

Exception: Do not identify such names as linguistic anti-patterns.

- "Opposite meaning": `System.err.println("Computation successful");`

Propose to create a new LA which could be described as "the message ad attribute name are opposite".

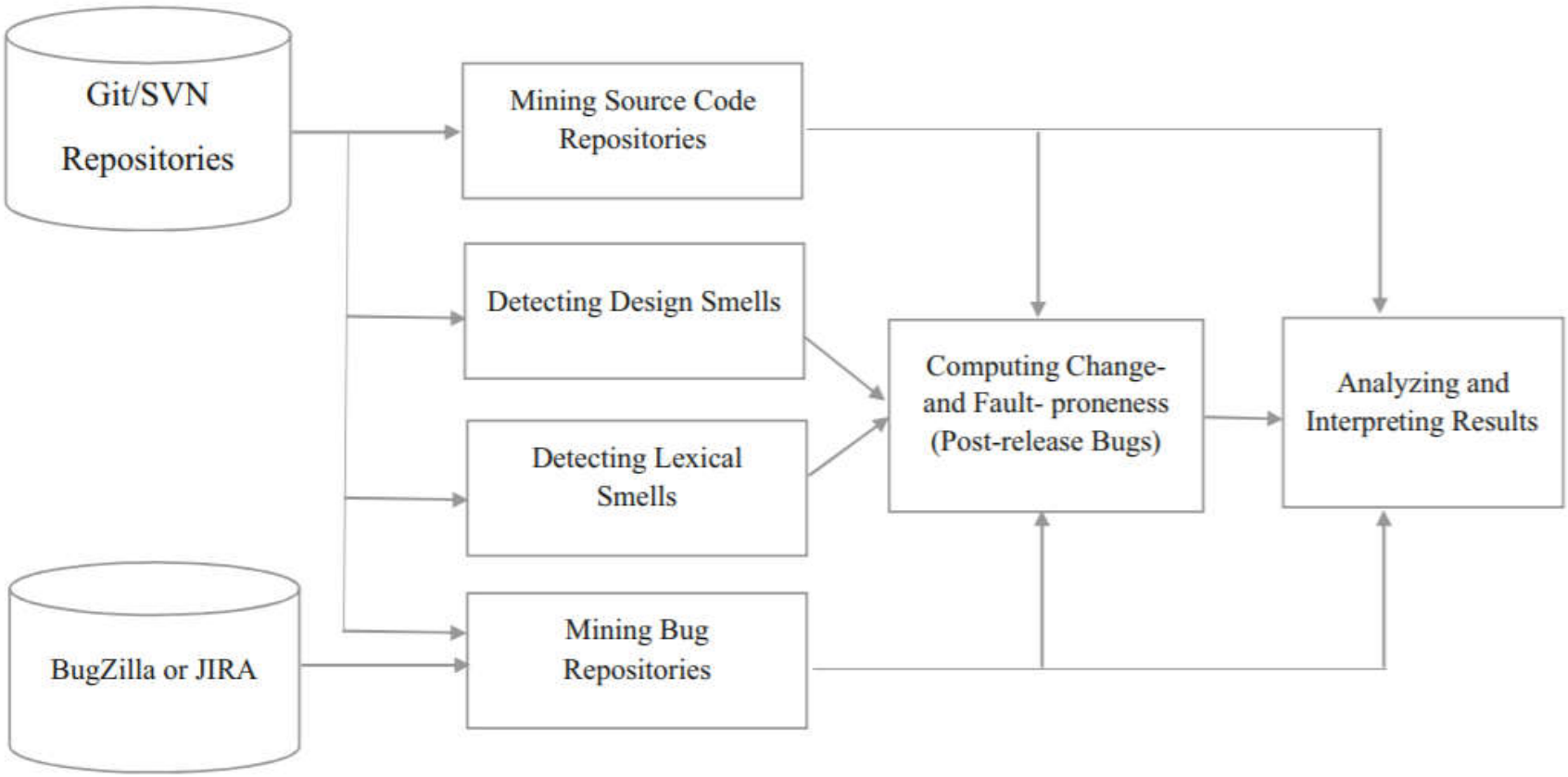
Summary

- LAs have a negative impact on code understandability.
- development teams should consider (1) educating their team members about LAs, (2) removing LAs from their software systems as soon as possible, and (3) using a common, well-known language for their identifiers and comments (not necessarily English).

Studied Systems

Characteristics of the analyzed projects

Projects	#Rel.	#Dev.	#Size (LOCs)	#All Classes	#Changes	#Classes Changed	#Faulty Changes
ANT	7	51	1,660,256	14,067	15,353	64,167	587
ArgoUML	13	25	644,829	27,822	5300	23,153	201
Hibernate	10	89	7,239,075	21,876	9075	89,658	179



Study Design Diagram

RQ1. Are classes with a particular family of smells (**DAPs**, **LAs**, or **both**) more **change-prone** than others?

1. Classes containing both DAPs and LAs versus classes with DAPs

Change-proneness results: design and lexical smells versus design smells (only)

Release	Design and lexical versus design smells				Adj. p value	OR
	#Design-Lexical	#Design	#No-Design-Lexical	#No-Design		
ANT 151	27	266	0	119	**<0.0001**	–
ANT 152	29	269	0	119	**<0.0001**	–
ANT 154	26	244	0	57	**0.012**	–
ANT 170	42	146	48	331	**0.0047**	1.98
ANT 180	93	357	5			
ANT 192	83	292	14			
ANT 15(MAIN)	23	162	4			
Hibernate 3.6.1	100	736	2			
Hibernate 3.6.2	77	589	17			
Hibernate 3.6.3	0	538	0			
Hibernate 3.6.4	0	452	0			
Hibernate 3.6.7	0	304	0			
Hibernate 3.6.8	0	315	0			
Hibernate 4.2.5	0	512	0			
Hibernate 4.2.7	0	492	0			
Hibernate 4.3.0	0	469	0			
ArgoUML 0.14	24	365	36	471	0.68	0.86
ArgoUML 0.16	26	397	44	437	0.10	0.65
ArgoUML 0.18	41	514	44	1077	**0.003**	1.95
ArgoUML 0.18.1	43	576	30	201	**0.0083**	0.50
ArgoUML 0.20	41	459	46	364	0.14	0.70
ArgoUML 0.22	45	653	75	285	**<0.0001**	0.26
ArgoUML 0.24	48	496	74	483	0.02	0.63
ArgoUML 0.26	42	435	66	525	0.22	0.76
ArgoUML 0.26.2	50	606	42	328	0.052	0.64
ArgoUML 0.28	96	374	64	591	0.232	0.79
ArgoUML 0.28.1	38	540	81	418	**<0.0001**	0.36
ArgoUML 0.30	241	370	965	595	**<0.0015**	0.50
ArgoUML 0.30.1	231	520	88	445	**<0.0001**	0.36

Significant p-values are highlighted in bold face

2. Classes having DAPs and LAs versus classes containing LAs

Change-proneness results: design and lexical smells versus lexical smells (only)

Release	Design and lexical versus lexical smells				Adj. p value	OR
	#Design-Lexical	#Lexical	#No-Design-Lexical	#No-Lexical		
ANT 151	27	58	0	13	**0.01**	–
ANT 152	29	57	0	13	**0.0093**	–
ANT 154	26	51	0	2	1	–
ANT 170	42	59	48	110	0.08	1.62
ANT 180	93	157	5	17	0.24	2.00
ANT 192	83	129	14	51	**0.011**	2.33
ArgoUML 0.18	41	50	44	84	0.12	1.56
ArgoUML 0.18.1	43	53	30	91	**0.0023**	2.45
ArgoUML 0.20	41	43	46	104	**0.0073**	2.14
ArgoUML 0.22	45	53	75	95	0.79	1.075
ArgoUML 0.24	48	62	74	131	0.22	1.36
ArgoUML 0.26	42	54	66	138	0.07	1.52
ArgoUML 0.26.2	50	69	42	219	**<0.0001**	3.76
ArgoUML 0.28	32	44	64	244	**0.00031**	2.76
ArgoUML 0.28.1	38	53	81	228	**0.0059**	2.01
ArgoUML 0.30	30	41	96	241	0.033	1.83
ArgoUML 0.30.1	38	51	88	231	**0.0091**	1.95

Significant p-values are highlighted in bold face

3. Classes containing DAPs versus classes with LAs

Change-proneness results: design smells versus lexical smells

Release	Design smells versus lexical smells				Adj. p value	OR
	#Design	#Lexical	#No-Design	#No-Lexical		
ANT 151	266	58	119	13	0.0328	0.50
ANT 152	269	57	119	13	0.044	0.51
ANT 154	244	51	57	2	**0.0044**	0.16
ANT 170	146	59	331	110	0.33	0.72
ANT 180	357	157	183	17	**<0.0001**	0.21
					0.0039	0.58
					0.25	0.73
					<0.0001	75.16
					<0.0001	11.58
					<0.0001	4.38
					<0.0001	2.47
					<0.0001	5.28
					<0.0001	5.39
					<0.0001	44.55
					<0.0001	26.44
					<0.0001	8.20
					0.027	1.72
ArgoUML 0.16	397	30	437	59	**0.0137**	1.78
ArgoUML 0.18	514	50	1077	84	0.25	0.80
ArgoUML 0.181	576	53	201	91	**<0.0001**	4.91
ArgoUML 0.20	459	43	364	104	**<0.0001**	3.04
ArgoUML 0.22	653	53	285	95	**<0.0001**	4.10
ArgoUML 0.24	496	62	483	131	**<0.0001**	2.16
ArgoUML 0.26	435	54	525	138	**<0.0001**	2.11
ArgoUML 0.262	606	69	328	219	**<0.0001**	5.85
ArgoUML 0.28	374	44	591	244	**<0.0001**	3.50
ArgoUML 0.281	540	53	418	228	**<0.0001**	5.54
ArgoUML 0.30	370	41	595	241	**<0.0001**	7.20
ArgoUML 0.30.1	520	51	445	231	**<0.0001**	5.28

Significant p-values are highlighted in bold face

Design anti-patterns contribute more to the **change-proneness** of **linguistic anti-pattern** classes than linguistic anti-patterns do to the change-proneness of design anti-pattern classes.

RQ2. Are classes with a particular family of smells (**DAPs**, **LAs**, or **both**) more **fault-prone** than others? (using post-release defects)

1. Classes containing both DAPs and LAs versus classes with DAPs

Fault-proneness results: design and lexical smells versus design (only)

Release	Design and lexical versus design smells				Adj. p value	OR
	#Design-Lexical	#Design	#No-Design-Lexical	#No-Design		
ANT 151	17	183	10	202	0.16	1.87
ANT 152	16	158	13	230	0.17	1.78
ANT 154	18	154	8	147	0.10	2.14
ANT 170	55	191	35	286	**0.00029**	2.34
ANT 180	50	174	48	366	**0.00051**	2.18
ANT 192	57	189	40			
ANT 15(MAIN)	20	194	7			
Hibernate 3.6.1	6	28	96			
Hibernate 3.6.2	6	29	88			
Hibernate 3.6.3	0	28	0			
Hibernate 3.6.4	0	33	0			
Hibernate 3.6.7	0	33	0			
Hibernate 3.6.8	0	32	0			
Hibernate 4.2.5	0	32	0			
Hibernate 4.2.7	0	43	0			
Hibernate 4.3.0	0	41	0			
ArgoUML 0.14	6	79	53	573	0.83	0.82
ArgoUML 0.16	5	100	55	736	0.53	0.66
ArgoUML 0.18	7	110	63	724	0.57	0.73
ArgoUML 0.18.1	13	141	72	1450	0.053	1.85
ArgoUML 0.20	12	143	60	634	0.87	0.88
ArgoUML 0.22	15	163	63	660	1	0.96
ArgoUML 0.24	12	165	93	773	0.13	0.60
ArgoUML 0.26	16	188	72	791	0.88	0.93
ArgoUML 0.26.2	17	190	61	770	0.65	1.12
ArgoUML 0.28	14	194	78	740	0.22	0.68
ArgoUML 0.28.1	13	188	78	777	0.26	0.68
ArgoUML 0.30	14	188	105	770	0.045	0.54
ArgoUML 0.30.1	15	178	111	817	0.10	0.62

Significant p-values are highlighted in bold face

2. Classes having DAPs and LAs versus classes containing LAs

Fault-proneness results: design and lexical smells versus lexical smells (only)

Release	Design and lexical versus lexical smells				Adj. p value	OR
	#Design-Lexical	#Design	#No-Design-Lexical	#No-Design		
ANT 151	17	29	10	42	0.06	2.43
ANT 152	16	26	13	44	0.12	2.06
ANT 154	18	27	8	26	0.15	2.14
ANT 170	55	75	35	94	**0.01**	1.96
ANT 180	50	70	48	104	0.09	1.54
ArgoUML 0.14	6	6	53	75	0.56	1.41
ArgoUML 0.16	5	6	55	78	1	1.18
ArgoUML 0.18	7	7	63	82	0.77	1.29
ArgoUML 0.18.1	13	17	72	117	0.68	1.24
ArgoUML 0.20	12	17	60	117	0.52	1.37
ArgoUML 0.22	15	17	63	117	0.23	1.63
ArgoUML 0.24	12	17	93	117	0.84	0.88
ArgoUML 0.26	16	17	72	117	0.33	1.52
ArgoUML 0.26.2	17	17	61	117	0.11	1.91
ArgoUML 0.28	14	19	78	269	0.017	2.53
ArgoUML 0.28.1	0.68	17	78	264	0.024	2.58
ArgoUML 0.30	14	17	105	264	0.06	2.06
ArgoUML 0.30.1	15	17	111	264	0.04	2.09

Significant p-values are highlighted in bold face

3. Classes containing DAPs versus classes with LAs

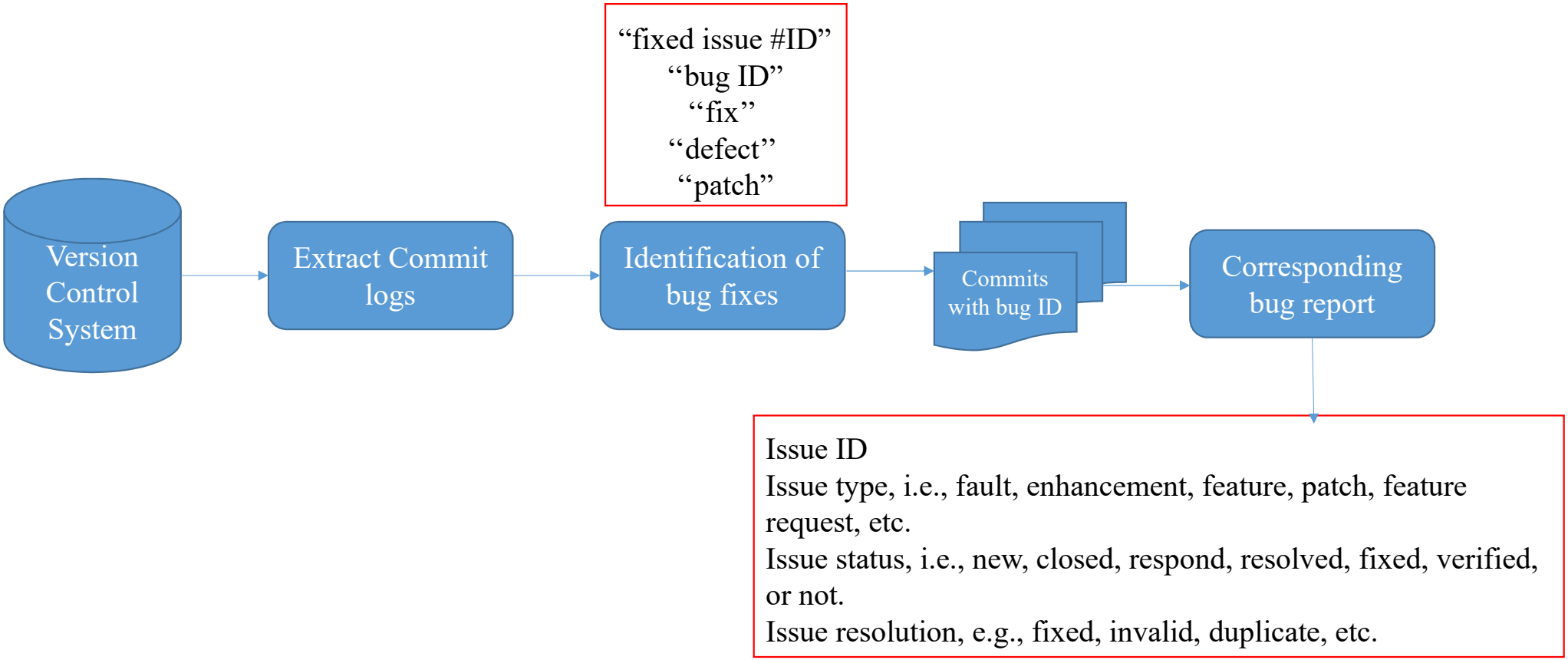
Fault-proneness results: design smells versus lexical smells

Release	Design smells versus lexical smells				Adj. p value	OR
	#Design	#Lexical	#No-Design	#No-Lexical		
ANT 151	183	29	202	42	0.36	1.31
ANT 152	158	26	230	44	0.59	1.16
ANT 154	154	27	147	26	1	1.08
ANT 170	191	75	286	94	0.36	0.83
ANT 180	174	70	366	104	0.05	0.70
ANT 192	189	77	301	103	0.32	2.84
				43	0.25	1.34
				04	**0.01**	2.75
				09	**0.0089**	2.97
				14	**0.00041**	5.20
				15	**0.000169**	4.89
				19	**0.00016**	4.90
				19	**0.0002**	4.79
				95	**<0.0001**	7.14
				44	**0.00052**	8.75
				11	**0.00084**	3.32
				58	0.027	1.72
ArgoUML 0.16	397	30	437	59	**0.0137**	1.78
ArgoUML 0.18	514	50	1077	84	0.25	0.80
ArgoUML 0.18.1	576	53	201	91	**<0.0001**	4.91
ArgoUML 0.20	459	43	364	104	**<0.0001**	3.04
ArgoUML 0.22	653	53	285	95	**<0.0001**	4.10
ArgoUML 0.24	496	62	483	131	**<0.0001**	2.16
ArgoUML 0.26	435	54	525	138	**<0.0001**	2.11
ArgoUML 0.26.2	606	69	328	219	**<0.0001**	5.85
ArgoUML 0.28	374	44	591	244	**<0.0001**	3.50
ArgoUML 0.28.1	540	53	418	228	**<0.0001**	5.54
ArgoUML 0.30	370	41	595	241	**<0.0001**	7.20
ArgoUML 0.30.1	520	51	445	231	**<0.0001**	5.28

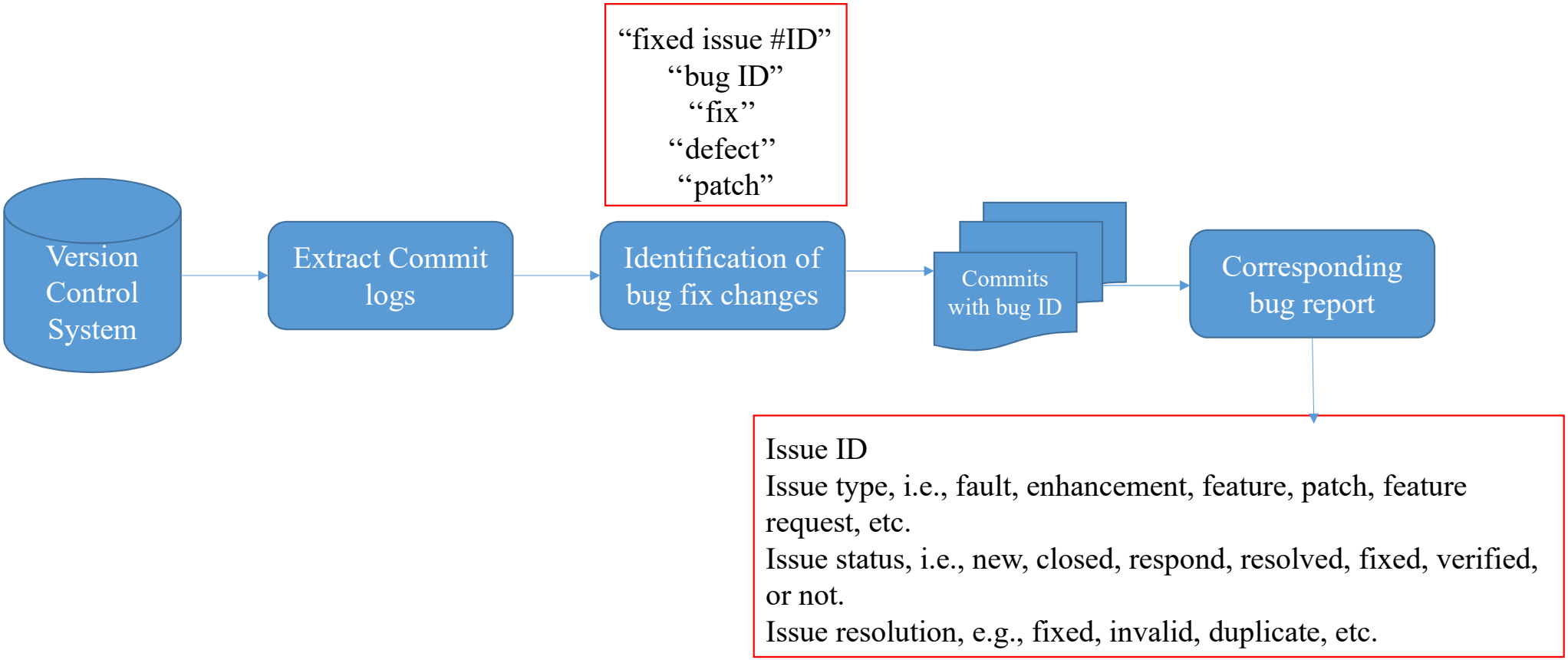
Significant p-values are highlighted in bold face

The occurrence of **design anti-patterns** in a class that experienced a **linguistic anti-pattern** has a strong relationship with **fault-proneness** than the occurrence of linguistic anti-pattern in a class that experienced a design anti-pattern.

Identifying post-release defects



Identifying defect-inducing changes



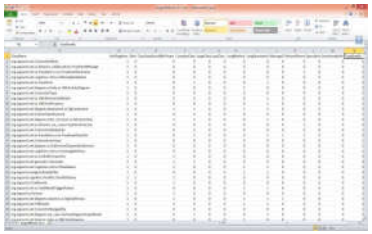
Summary

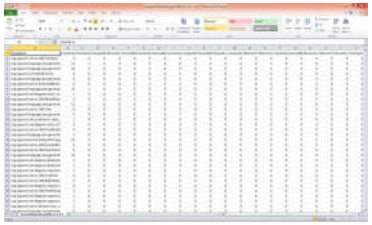
- LAs can make, in some cases, **classes with DAPs** more **fault-prone** when both occur in classes of object-oriented systems.
- In a lot of cases, **classes containing DAPs** are **more change- and fault-prone** than classes with **LAs**.
- Development teams and quality assurance teams should better focus their refactoring efforts on components with **design anti-patterns (while not neglecting linguistic anti-patterns)** to assure good quality for their systems

RQ1. Do **design patterns and–or design anti-patterns mutate** during the evolution of software systems? What is the **probability** of occurrence of different types of **mutations**?

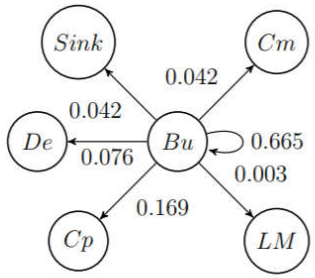
Building Markov Model

DAPs detection results

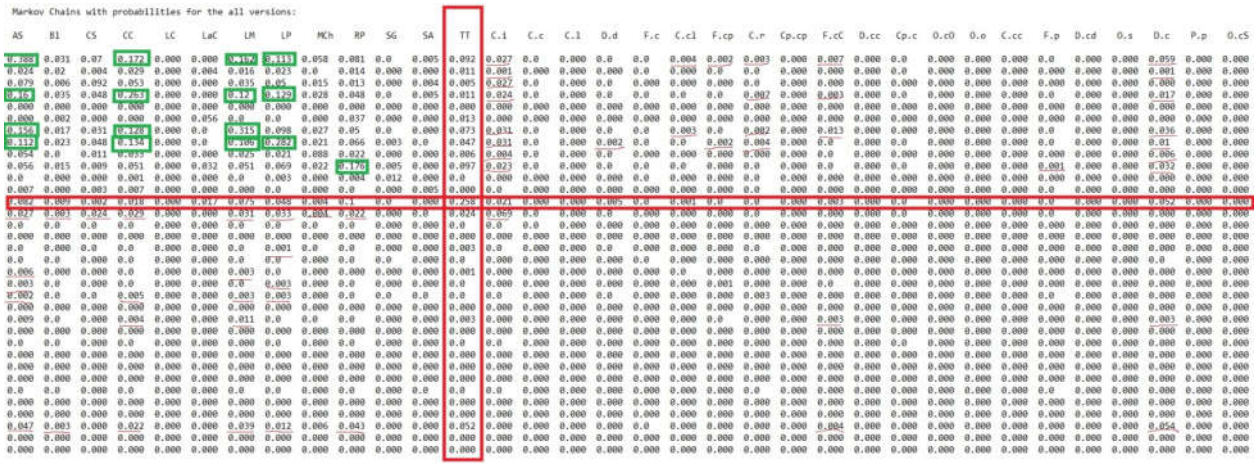




DPs detection results



Builder (Bu) mutation among the different revisions of Matsim.



RQ1. Do **design patterns and–or design anti-patterns mutate** during the evolution of software systems? What is the **probability** of occurrence of different types of **mutations**?

Change probabilities of design anti-patterns and design patterns in Mule

	Source	AS	Bl	CS	CC	LC	LZC	LM	LP	MCh	RP	SC	SG	SA	Bu	Cm	Cp	FM	De	Ob	PT	Si	Sink
AS	0.032	**0.937**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.030	0.000	0.000	0.000	0.000	0.000
Bl	0.000	**0.313**	**0.313**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.374**	0.000	0.000	0.000	0.000	0.000
CS	0.003	0.000	0.006	**0.963**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.028	0.000	0.000	0.000	0.000	0.000
CC	0.001	0.000	0.000	0.071	**0.843**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.084	0.000	0.000	0.000	0.000	0.000
LC	0.000	0.000	0.000	0.000	0.000	**1**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LZC	0.000	0.000	0.000	0.000	0.000	0.030	**0.946**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.024	0.000	0.000	0.000	0.000	0.000
LM	0.000	0.000	0.000	0.010	0.000	0.000	0.039	**0.907**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.044	0.000	0.000	0.000	0.000	0.000
LP	0.000	0.000	0.000	0.014	0.000	0.000	0.009	**0.115**	**0.676**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.186**	0.000	0.000	0.000	0.000	0.000
MCh	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**1**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
RP	0.000	0.000	0.000	0.033	0.000	0.000	0.067	0.000	0.000	**0.233**	**0.233**	0.000	0.000	0.000	0.000	0.000	0.000	**0.433**	0.000	0.000	0.000	0.000	0.000
SC	**0.096**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.074	**0.649**	0.000	0.000	0.000	0.000	0.000	**0.181**	0.000	0.000	0.000	0.000	0.000
SG	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.053	**0.947**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
SA	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**1**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Bu	0.000	0.000	0.000	0.003	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.708**	0.000	0.001	**0.152**	0.000	0.067	0.000	0.065	0.000
Cm	0.000	0.000	0.000	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.019	0.000	**0.687**	**0.096**	**0.193**	0.000	0.000	0.000	0.000	0.000
Cp	0.000	0.000	0.000	0.002	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.058	**0.882**	0.054	0.000	0.000	0.000	0.000	0.000
FM	0.000	0.000	0.000	0.003	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.042	0.087	**0.764**	**0.099**	0.000	0.000	0.000	0.000
De	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.014	**0.971**	0.013	0.000	0.000	0.000
Ob	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.024	0.024	**0.951**	0.000	0.000	0.000
PT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**1**	0.000	0.000
Si	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.029	0.000	0.000	0.006	**0.964**	0.000

RQ1. Do **design patterns and–or design anti-patterns mutate** during the evolution of software systems? What is the **probability** of occurrence of different types of **mutations**?

Most representative design pattern and design anti-patterns mutations with mutation probabilities

System	Mutation Type	From	To	Probability
Apache Ignite	Design Anti-pattern → Design Anti-pattern	Blob (Bl)	AntiSingleton (AS)	0.375
	Design Anti-pattern → Design pattern	-	-	-
	Design pattern → Design Anti-pattern	-	-	-
Mule	Design Anti-pattern → Design pattern	RefusedParentBequest (RP)	FactoryMethod (FM)	0.433
	Design pattern → Design Anti-pattern	Command (Cm)	SwissArmyKnife (SA)	0.019
	Design pattern → Design pattern	Command (Cm)	FactoryMethod	0.193
Nuxeo	Design Anti-pattern → Design Anti-pattern	Blob (bl)	AntiSingleton(AS)	0.283
	Design Anti-pattern → Design pattern	Blob (Bl)	FactoryMethod (FM)	0.297
	Design pattern → Design Anti-pattern	Singleton (Si)	LazyClass (ZC)	0.004
	Design pattern → Design pattern	Singleton (Si)	FactoryMethod (FM)	0.133
Ovirt	Design Anti-pattern → Design Anti-pattern	Blob (bl)	AntiSingleton(AS)	0.299
	Design Anti-pattern → Design pattern	-	-	-
	Design pattern → Design Anti-pattern	Singleton (Si)	AntiSingleton (AS)	0.001
	Design pattern → Design pattern	Singleton (Si)	Prototype (PT)	0.097

DPs and **DAPs** mutate during the evolution of software systems.

In most of the studied systems, more than half of the **DAP** occurrences mutated during the evolution.

In most of the systems, almost all the **DPs** occurrences remained stable during the evolution process.

Blob and **Command** are the **DAPs** and **DPs**, which have higher mutation probabilities.

RQ2. What types of **changes** lead to a **mutation** between **design patterns** and-or **design anti-patterns**?

. Change types identified from the source code of the systems studied

Change type	**srcML tag(s)**
Access	*super, public, private, protected, extern*
Class	*extends, class, interface, implements, class_decl*
Code block	*expr_stmt, expr, block*
Comment	*annonation, comment, @type, @format*
Control flow	*while, do, if, else, elseif, break, goto, for, foreach, control, continue, then, switch, case, return, incr, default, condition*
Declaration	*decl_stmt, modifier, specifier, decl, function_decl, literal, label, empty_stmt, construction_decl, annonation_dfn*
Exception	*assert, try, catch, throw, throws, finally*
Import	*import, package*
Invocation	*call*
Method	*constructor, default, static, type, lambda, function, function_decl, unit*
Operator	*index, synchronized, enum, operator, ternany*
Parameter	*argument, param, parameter_list, argument_list, parameter*
Renaming	*renaming, name*

RQ2. What types of **changes** lead to a **mutation** between **DPs** and-or **DAPs**?

Change Type	**srcML tag(s)**
Access	super, public, private, protected, extern
Class	extends, class, interface, implements, class_decl
Code block	expr_stmt, expr, block
Comment	annonation, comment, @type, @format
Control Flow	while, do, if, else, elseif, break, goto, for, foreach, control, continue, then, switch, case, return, incr, default, condition
Declaration	decl_stmt, modifier, specifier, decl, function_decl, literal, label, empty_stmt, construction_decl, annonation_dfn
Exception	assert, try, catch, throw, throws, finally
Import	import, package
Invocation	call
Method	constructor, default, static, type, lambda, function, function_decl, unit
Operator	index, synchronized, enum, operator, ternany
Parameter	argument, param, parameter_list, argument_list, parameter
Renaming	renaming, name

RQ2. What types of **changes** lead to a **mutation** between **design patterns** and-or **design anti-patterns**?

Number of different types of changes in design patterns and design anti-patterns

Systems →	Eclipse		Nuxeo		oVirt		Matsim		Apache Solr		Apache Ignite		Mule	
Change Types ↓	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	AP	**DP**	**AP**	**DP**	**AP**	**DP**	**AP**	**DP**	**AP**	**DP**	**AP**	**DP**	**AP**	**DP**
Access	33	6	85	0	174	9	271	117	34	15	11	55	20	12
Class	268	91	236	6	3804	90	1697	690	721	155	781	218	443	198
Code block	4197	1075	1070	13	13923	233	8805	3203	3873	459	3903	203	1430	413
Comment	32939	11388	9269	109	15013	411	22519	9287	9298	2616	14554	1567	6354	3780
Control Flow	10487	1870	966	10	6440	118	5398	2094	3166	636	3433	133	916	384
Declaration	9721	2789	3133	24	27605	487	24244	9609	8803	1392	9527	349	3904	1103
Exception	996	341	946	1	619	29	1602	314	1696	457	2076	64	505	173
Import	2566	835	2734	23	18819	211	13013	4679	4024	491	4584	394	3234	793
Invocation	1707	394	556	4	7312	91	8520	3026	2598	287	2069	75	945	240
Method	4060	942	1487	29	13792	292	4702	1922	3215	511	3364	266	1940	747
Operator	13540	2803	3533	8	35513	403	57112	24326	7963	702	7207	525	5241	1975
Parameter	5629	1541	2179	3	24488	292	22080	5149	8375	1069	9024	332	3252	756
Renaming	59254	14707	16259	23	262491	3536	294661	145720	44422	4811	63961	4396	28110	9738
# Changed classes	10957	3155	5402	81	34780	55	32596	13768	7956	1192	9290	857	5684	2175
Total classes	20331	7574	39051	1263	142537	2482	62272	79480	32332	5490	27080	5796	47146	17553

AP, DP= Number of changes in design anti-patterns and design patterns respectively

RQ2. What types of **changes** lead to a **mutation** between **design patterns** and-or **design anti-patterns**?

Number of different types of changes in design patterns and design anti-patterns mutation.

Systems→	Eclipse		Nuxeo		oVirt		Matsim		Apache Solr		Apache Ignite		Mule	
Change Types↓	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	APDP	DPAP	APDP	DPAP	APDP	DPAP	APDP	DPAP	APDP	DPAP	APDP	DPAP	APDP	DPAP
Access	0	0	0	1	1	1	0	12	3	0	0	0	0	1
Class	3	1	2	0	10	0	4	29	1	7	2	4	6	3
Code block	1	2	1	4	21	22	10	132	4	1	1	3	27	17
Comment	102	192	7	4	69	31	38	739	129	242	58	23	160	85
Control Flow	28	38	0	3	7	6	12	96	44	31	16	4	21	3
Declaration	56	78	3	2	82	38	56	412	141	90	33	16	49	32
Exception	4	6	0	0	4	1	6	28	19	13	14	4	3	2
Import	20	14	3	2	32	18	28	257	60	34	17	16	28	18
Invocation	6	7	0	0	15	20	7	183	1	6	5	6	10	3
Method	16	18	2	2	41	24	7	96	64	43	10	12	23	13
Operator	38	37	2	0	43	66	95	523	22	20	6	13	87	32
Parameter	22	41	0	0	37	20	16	423	25	47	22	29	13	22
Renaming	675	469	3	2	297	1036	462	5096	165	406	100	63	334	280
# Changed classes	71	77	7	6	61	54	58	681	58	66	34	29	53	39

APDP, DPAP= Number of changes in design anti-patterns to design patterns and design patterns to design anti-patterns mutations respectively

In general, some of the change types affect the mutation from **DPs** and–or **DAPs**.

The most representative change types leading to mutations in all the studied systems are "**Renaming**", "**Comment**", "**Declaration**", and "**Operator**".

RQ3. What is the **fault-proneness** of mutated **design patterns** and **design anti-patterns**? What transitions lead to more **fault-prone mutations**?

Design anti-pattern and design-pattern mutations

Systems	**# of faulty classes**		**# of clean classes**
	Design Anti-patterns	Design Patterns	
Apache Ignite	10,984	1,051	81,093
Apache Solr	11,156	219	109,225
Eclipse	15,240	5,182	19,928
Matsim	4,053	1,888	896,510
Mule	17,794	5,924	197,574
Nuxeo	18,724	396	146,180
Ovirt	12,605	110	217,565

RQ3. What is the **fault-proneness** of mutated **design patterns** and **design anti-patterns**? What transitions lead to more **fault-prone mutations**?

: Transitions Fault-proneness

System	Mutation Type	From	To	Probability
Apache Ignite	Design Anti-pattern → Design Anti-pattern	LongParameterList	LongMethod	57.1%
	Design Anti-pattern → Design Anti-pattern	LongMethod	LazyClass	28.5%
Apache Solr	Design Anti-pattern → Design Anti-pattern	RefusedParentBequest	MessageChain	42.7%
	Design Anti-pattern → Design Anti-pattern	LongMethod	LazyClass	15.6%
	Design pattern → Design Anti-pattern	FactoryMethod	LongMethod	5.6%
Matsim	Design pattern → Design pattern	Builder	FactoryMethod	67.7%
	Design Anti-pattern → Design Anti-pattern	SpagettiCode	RefusedParentBequest	15.2%
	Design Anti-pattern → Design pattern	AntiSingleton	FactoryMethod	11.4%
Mule	Design pattern → Design pattern	Builder	FactoryMethod	47.9%
	Design Anti-pattern → Design pattern	ComplexClass	FactoryMethod	26.4%
	Design Anti-pattern → Design Anti-pattern	ComplexClass	ClassDataShouldBePrivate	22.3%
Nuxeo	Design Anti-pattern → Design Anti-pattern	LazyClass	LargeClass	28.5%
	Design pattern → Design pattern	Singleton	FactoryMethod	49.5%
Ovirt	Design Anti-pattern → Design Anti-pattern	Blob	AntiSingleton	72.2%
	Design pattern → Design pattern	Singleton	Prototype	16.6%

DAPs are more **fault-prone** than **DPs**. Mutations from **DAPs** to **DPs** are more **faulty** than other types of mutations.

RQ4. Do specific types of **changes** lead to increase **fault-proneness** during **design patterns** and-or **design anti-patterns mutations**?

Numbers of change types in the studied systems leading to faults

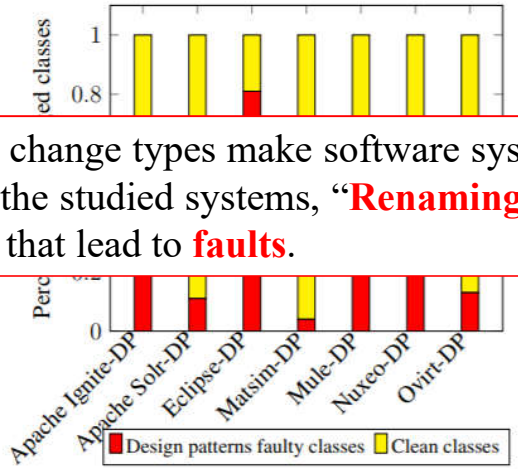
Systems →	**Apache Ignite**	**Apache Solr**	**Eclipse**	**Matsim**	**Mule**	**Nuxeo**	**oVirt**
Change Types ↓	# of changes	# of changes	# of changes	# of changes	# of changes	# of changes	# of changes
Access	18	18	37	22	11	62	25
Class	689	431	306	306	422	208	763
Code block	2,972	2,102	4,919	1,284	1,306	854	3,833
Comment	12,169	5,498	38,150	2,813	6,270	6,161	4,653
Control flow	2,903	1,935	11,678	660	1067	819	2,406
Declaration	5,912	5,386	11651	3,129	3,191	2,628	7,484
Exception	1,696	1221	1255	140	526	786	210
Import	2,831	2,400	2,958	1,425	2,443	2,064	4,268
Invocation	1,550	1,196	1,986	1,061	840	476	1,882
Method	2,509	1,851	4,093	637	1,697	1,229	3,619
Operator	5,120	4,364	16,094	6,215	4,228	2,675	8,134
Parameter	6,418	3,504	5,108	2,655	2,607	1,815	5,337
Renaming	47,811	24,640	67,040	32,445	22,968	12,736	65,245
Total changed classes	5,505	4,163	11,934	3,073	4,324	3,514	7,150

RQ4. Do specific types of **changes** lead to increase **fault-proneness** during **design patterns** and-or **design anti-patterns mutations**?

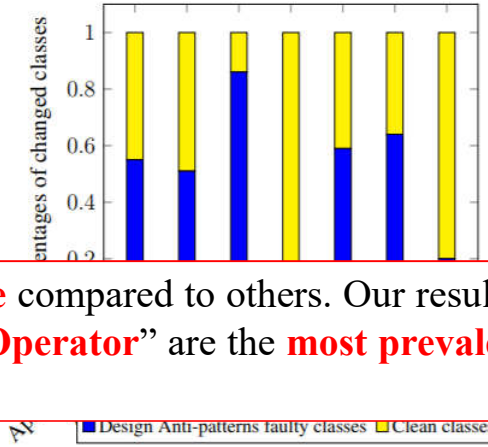
Numbers of faulty and clean changed classes

Systems	Patterns	# Faulty classes	# Clean classes
Apache Ignite	Design Anti-patterns	5,112	4,178
	Design Patterns	393	464
Apache Solr	Design Anti-patterns	4,035	3,921
	Design patterns	128	1,064
Eclipse	Design Anti-patterns	9,406	1,551
	Design patterns	2,554	601
Matsim	Design Anti-patterns	2,549	30,042
	Design patterns	524	13,244
Mule	Design Anti-patterns	3,374	2,311
	Design patterns	950	1,225
Nuxeo	Design Anti-patterns	3,469	1,935
	Design patterns	45	36
oVirt	Design Anti-patterns	7,075	27,705
	Design patterns	75	482

RQ4. Do specific types of **changes** lead to increase **fault-proneness** during **design patterns** and-or **design anti-patterns mutations**?



Faulty changed classes percentages with design pattern in the studied systems



Faulty changed classes with design anti-patterns percentages in the studied systems

Some change types make software systems **more fault-prone** compared to others. Our result show that in all the studied systems, "**Renaming**", "**Comment**", and "**Operator**" are the **most prevalent change types** that lead to **faults**.

RQ5. Do the occurrences of **Linguistic anti-patterns** increase **change-** and **fault-proneness** during **design patterns** and-or **design anti-patterns mutations**?

Change-prone classes with linguistic anti-patterns (LAs), design patterns(DPs) and design anti-patterns(DAPs)

Systems	CCLA	DAPs and LAs		DPs and LAs	
		CCAP	CCLAAP	CCDP	CCLADP
Apache Ignite	5,914	9,290	4,727	857	432
Apache Solr	6,369	7,956	2,830	1,192	729
Eclipse	446	10,957	129	3,155	38
Matsim	284	32,596	101	13,768	45
Mule	260	5,684	117	2,175	17
Nuxeo	430	5,402	290	81	10
Ovirt	517	34,780	248	557	7

CCLA, CCAP, CCDP= NO. of classes containing LAs, DAPs and DPs respectively
CCLAAP= NO. of changed classes containing LAs and DAPs,
CCLADP=NO. of changed classes containing LAs and DPs

RQ5. Do the occurrences of **Linguistic anti-patterns** increase **change-** and **fault-proneness** during **design patterns** and-or **design anti-patterns mutations**?

Fault-proneness of classes with Linguistic Anti-patterns (LAs)

Systems	NCLA	NFC	NFCLA	PFCLA	PCLAF
Apache Ignite	5,914	3,303	438	13.26%	7.41%
Apache Solr	6,369	5,060	59	1.16%	0.91%
Eclipse	446	9,179	175	1.90%	39.24%
Matsim	284	747	14	1.87%	4.92%
Mule	260	5,176	78	1.50%	30%
Nuxeo	430	6,446	104	1.61%	24.19%
0Virt	517	698	11	1.58%	2.13%

NCLA = No. of classes containing LAs, **NFC** = No. of faulty classes
NFCLA = No. of faulty classes containing LAs
PFCLA = Percentage of faulty classes containing LAs
PCLAF = Percentage of classes containing LAs those are faulty