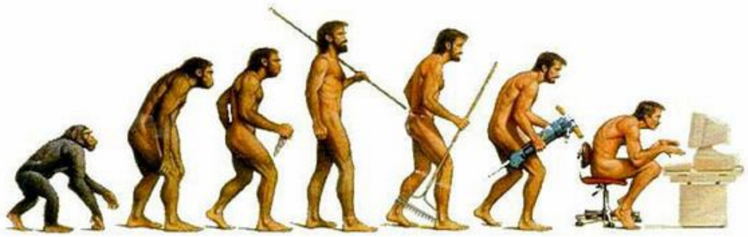
On the Analysis of Artifact Evolution: An Aggregated View and Lessons Learned

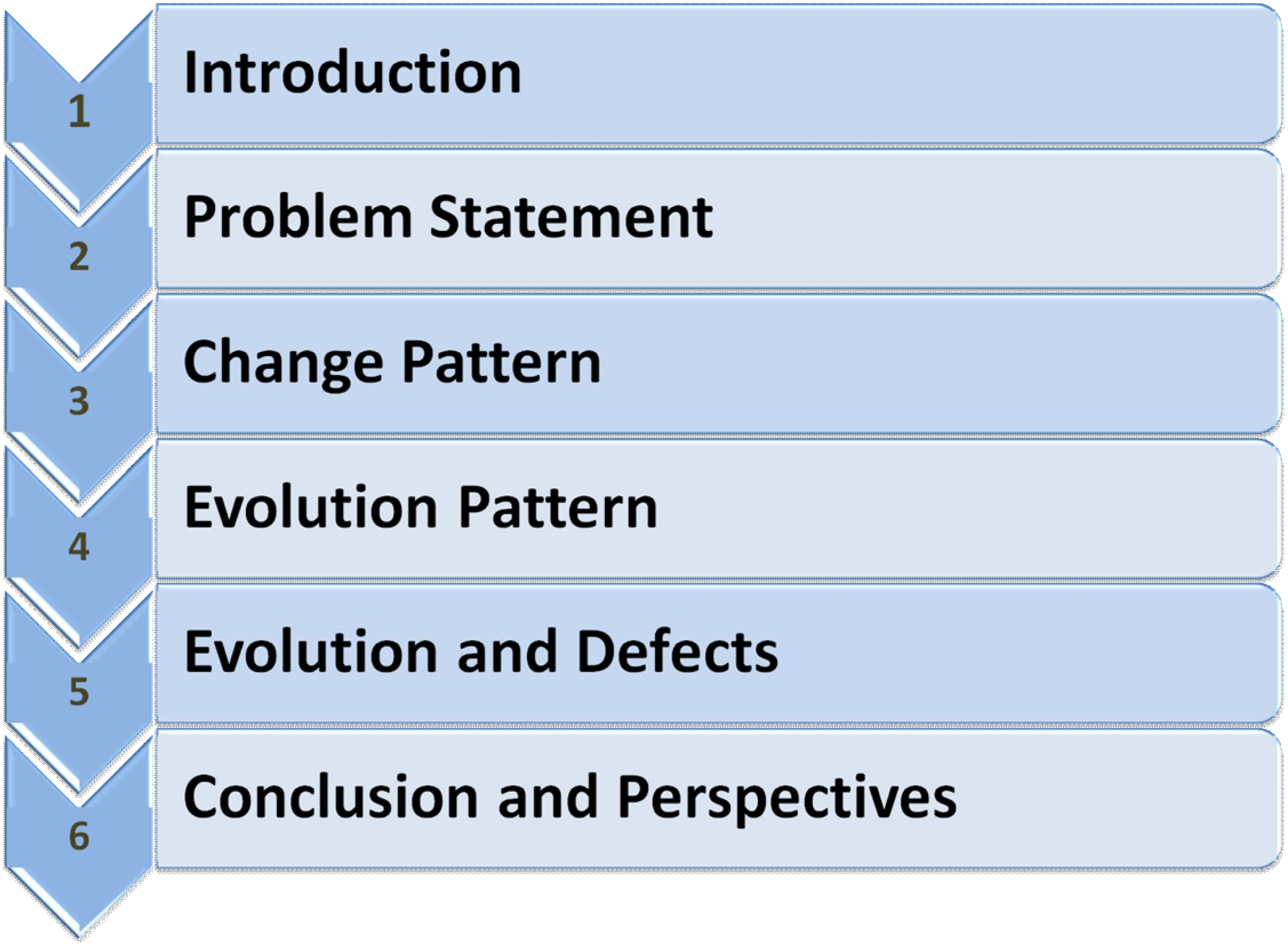


Fehmi Jaafar

Université de Montréal

POLYTECHNIQUE MONTRÉAL
WORLD-CLASS ENGINEERING

Content



1 Introduction

2 Problem Statement

3 Change Pattern

4 Evolution Pattern

5 Evolution and Defects

6 Conclusion and Perspectives

Software has become omnipresent and vital in our information-based society.

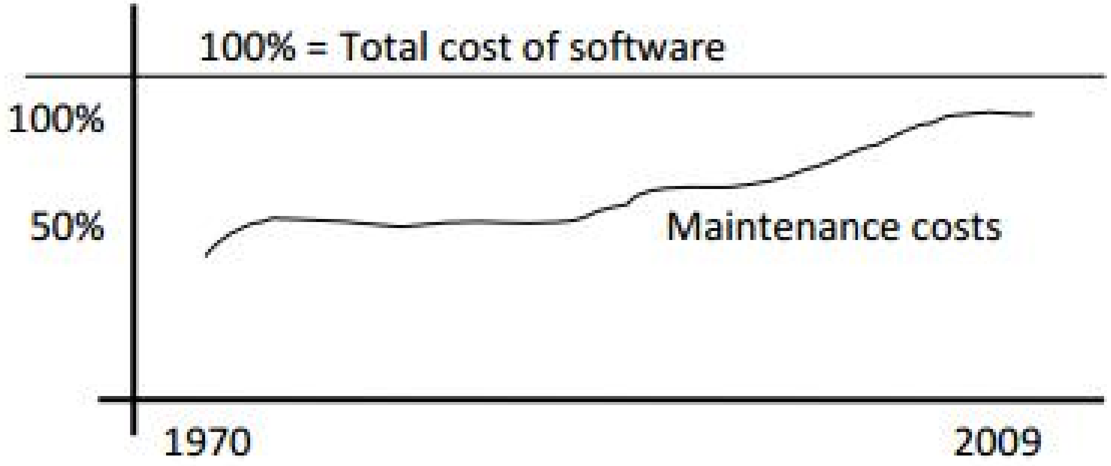
So all software producers should assume responsibility for its reliability.

Maintenance and Evolution

Under maintenance

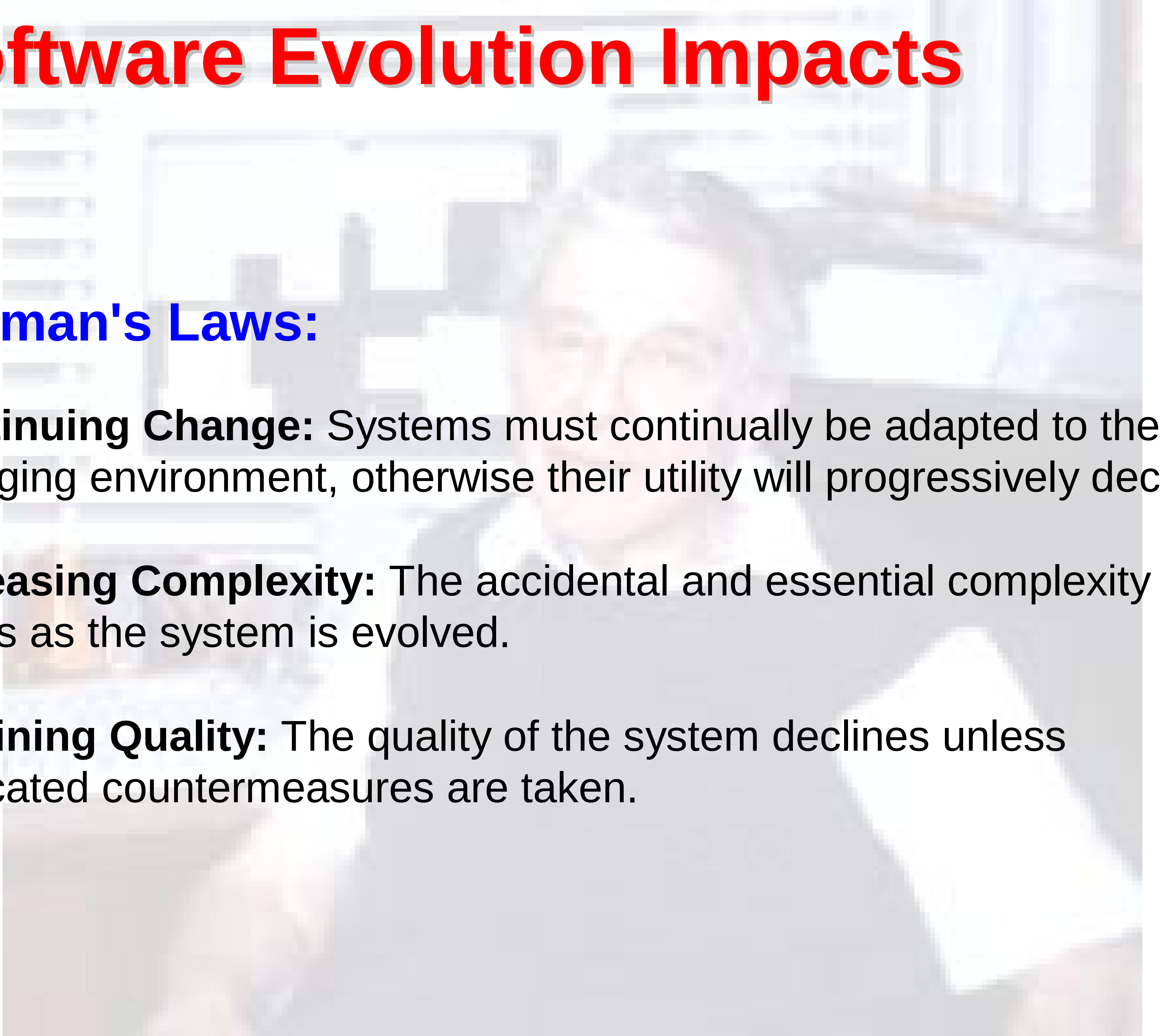
Fred Brooks, in the **Mythical Man-Month**, states that over **90%** of the costs of a typical system arise in the maintenance phase!

omnext
Software Analysis as Service



Development of Software maintenance costs as percentage of total cost

Software Evolution Impacts



Lehman's Laws:

Continuing Change: Systems must continually be adapted to the changing environment, otherwise their utility will progressively decline.

Increasing Complexity: The accidental and essential complexity grows as the system is evolved.

Declining Quality: The quality of the system declines unless dedicated countermeasures are taken.

Software Evolution Impacts

As Software systems evolved, their designs become **more complex** over time and harder to change.

In absence of knowledge on the artefacts’ **dependencies**, developers could introduce design defects and faults.

Motivation

⬀ Quality

⬀ Speed

⬀ Efficiency

⬂ Cost

Problems

How to detect **hidden** evolution relationships among artifacts?

How to analyse program evolution **effect**?

Previous Work

1 - Co-change Pattern

2 - Co-evolution Pattern

Synchrony Change Pattern

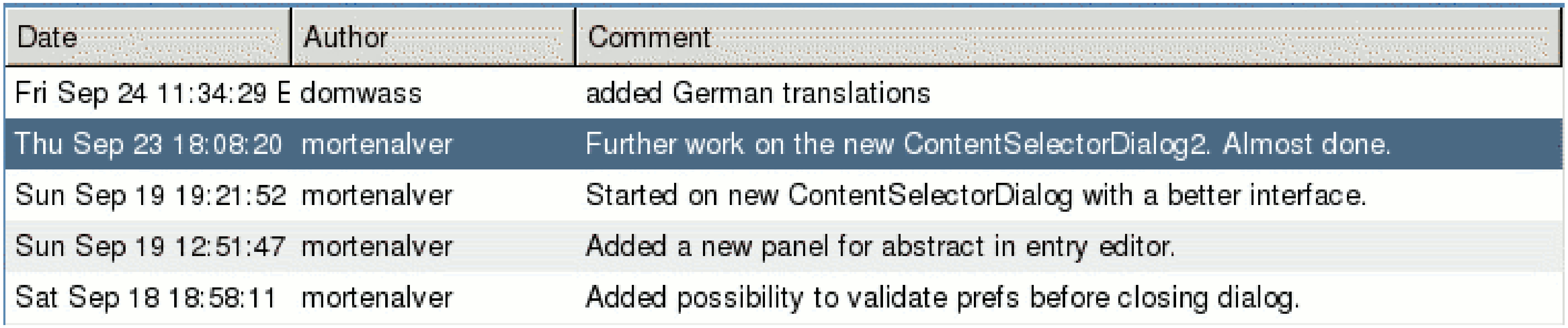
The development and maintenance of a system involves handling a large number of artifacts.

A **change** to one artifact may imply a large number of changes to various other artifacts.

Yann-Gaël Guéhéneuc Salah Bouktif and Giuliano Antoniol. Extracting change-patterns from cvs repositories. Working Conference on Reverse Engineering. 2006.

Co-change

Two artefacts are co-changing if they are changed by the **same author** and with the **same log message** in a time-window of **some ms**.



Date	Author	Comment
Fri Sep 24 11:34:29 E	domwass	added German translations
Thu Sep 23 18:08:20	mortenalver	Further work on the new ContentSelectorDialog2. Almost done.
Sun Sep 19 19:21:52	mortenalver	Started on new ContentSelectorDialog with a better interface.
Sun Sep 19 12:51:47	mortenalver	Added a new panel for abstract in entry editor.
Sat Sep 18 18:58:11	mortenalver	Added possibility to validate prefs before closing dialog.

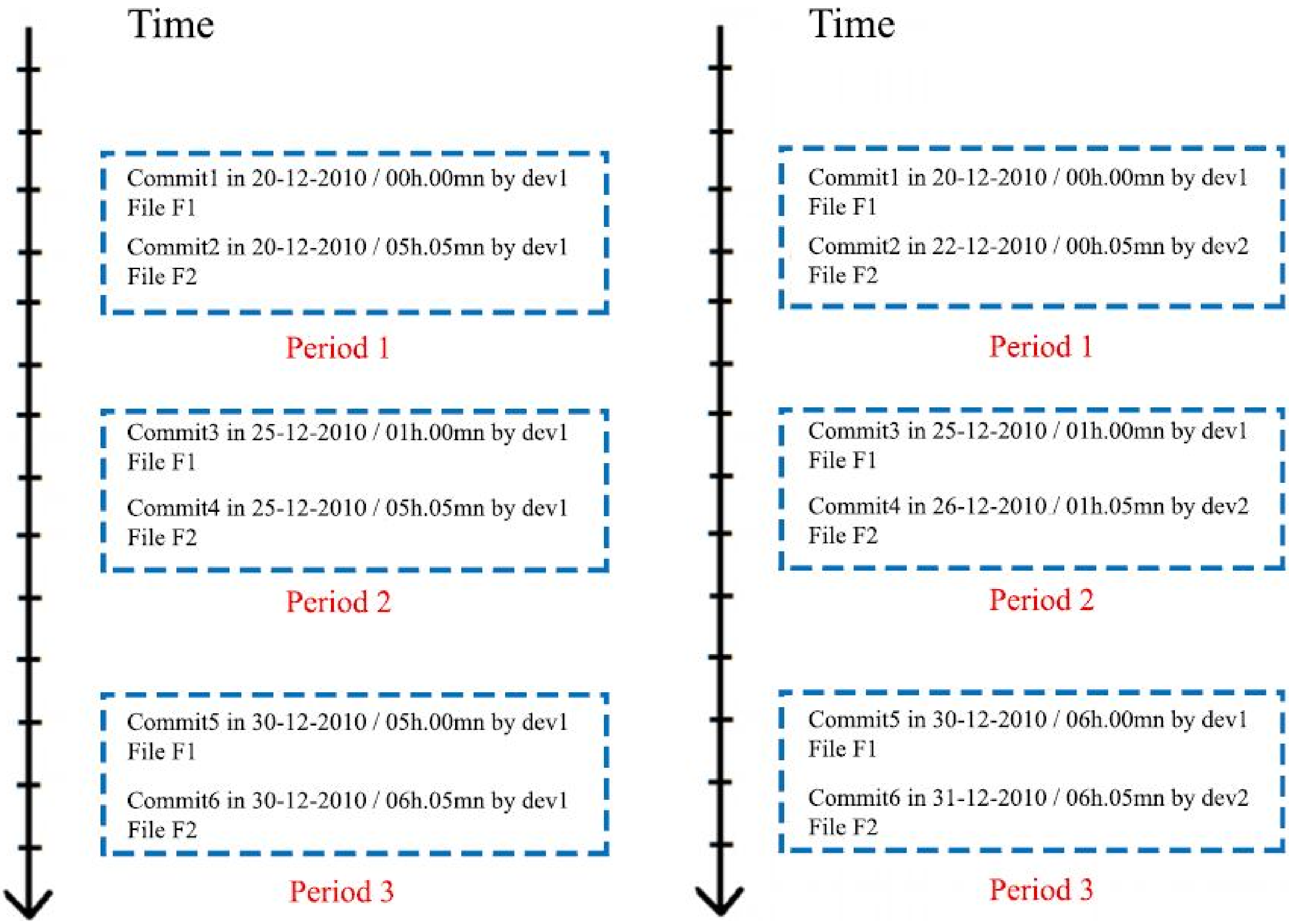
Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering, 2004.*

Example

In **ArgoUML**, **developers** maintained in the **same time** **NotationUtilityJava** and **ModelElementNameNotationUml**.
The bug **ID 29265** confirms that the two files have dependencies.

In the Bugzilla of **ArgoUML**, the bug **ID 53783** relates **ArgoDiagram** with **ModeCreateAssociationClass**.
Their changes were committed by the **same developer** but always separated by a **few hours**.

Missing Dependencies



Commit1 in 20-12-2010 / 00h.00mn by dev1
File F1

Commit2 in 20-12-2010 / 05h.05mn by dev1
File F2

Commit3 in 25-12-2010 / 01h.00mn by dev1
File F1

Commit4 in 25-12-2010 / 05h.05mn by dev1
File F2

Commit5 in 30-12-2010 / 05h.00mn by dev1
File F1

Commit6 in 30-12-2010 / 06h.05mn by dev1
File F2

Commit1 in 20-12-2010 / 00h.00mn by dev1
File F1

Commit2 in 22-12-2010 / 00h.05mn by dev2
File F2

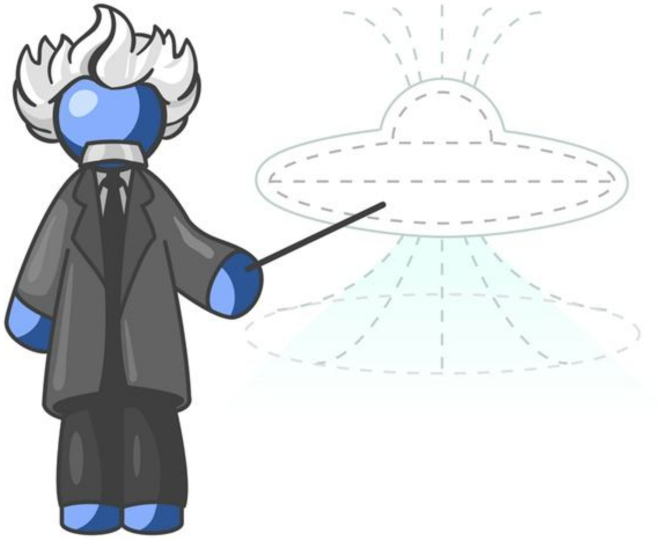
Commit3 in 25-12-2010 / 01h.00mn by dev1
File F1

Commit4 in 26-12-2010 / 01h.05mn by dev2
File F2

Commit5 in 30-12-2010 / 06h.00mn by dev1
File F1

Commit6 in 31-12-2010 / 06h.05mn by dev2
File F2

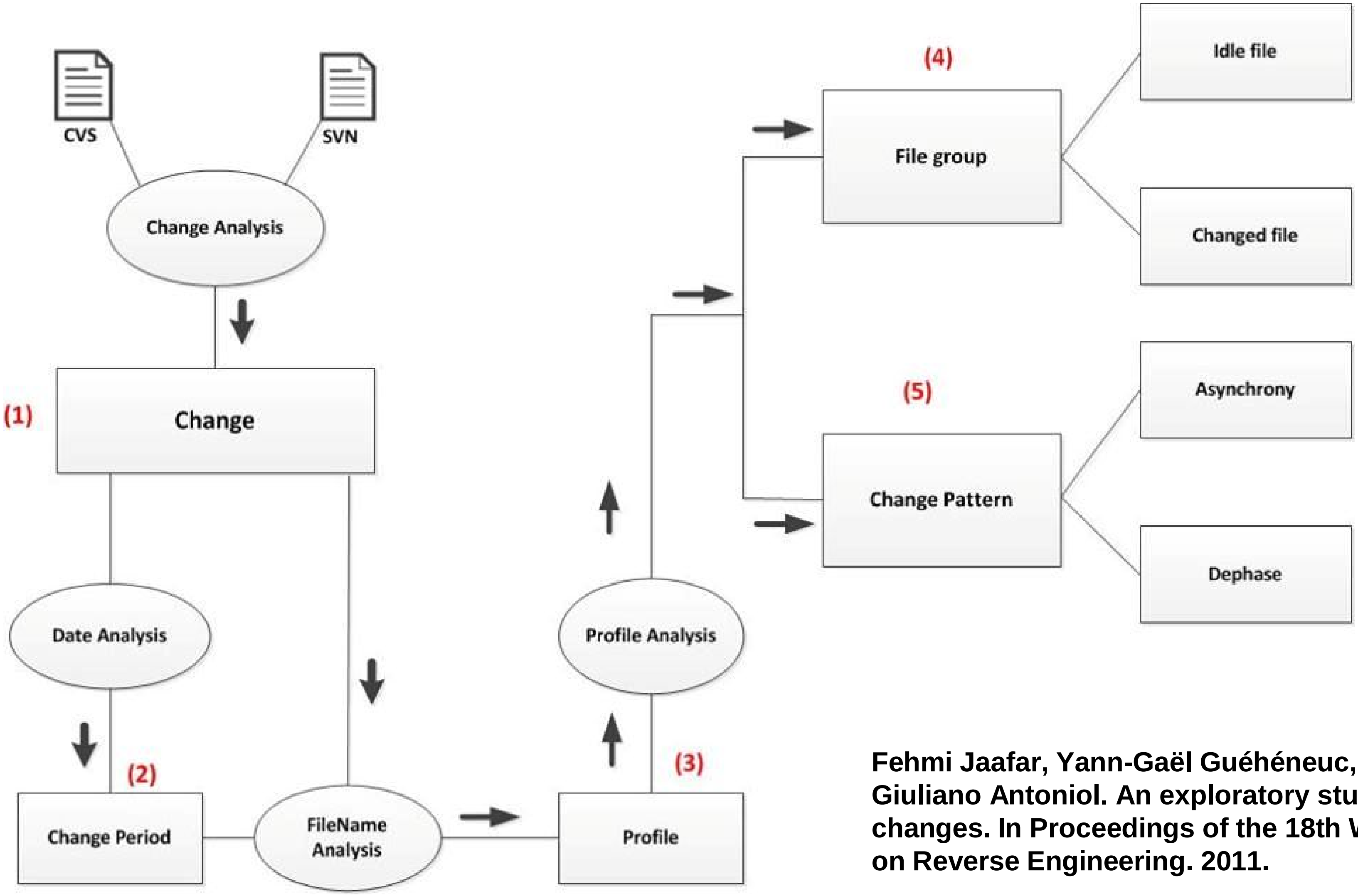
Goal 1: A New Model of Co-change



The **Asynchrony change pattern** describes a set of files that always change together in the same change periods.

A **change period** is a period of time during which several commits to different files occurred without “interruption”.

Approach: Macocha

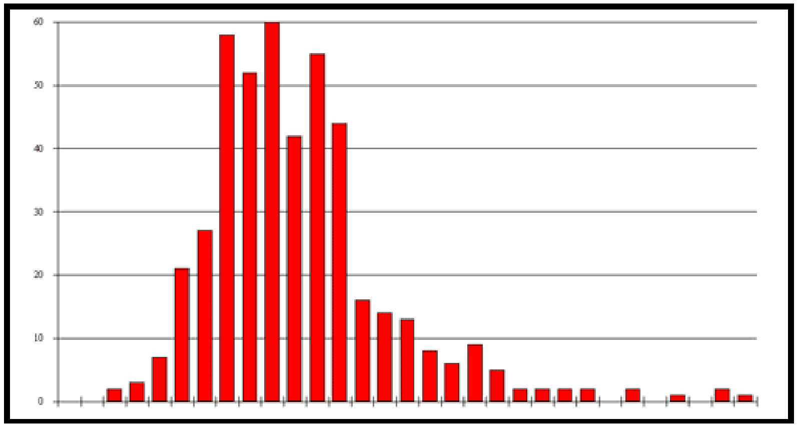


Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. An exploratory study of macro co-changes. In Proceedings of the 18th Working Conference on Reverse Engineering. 2011.

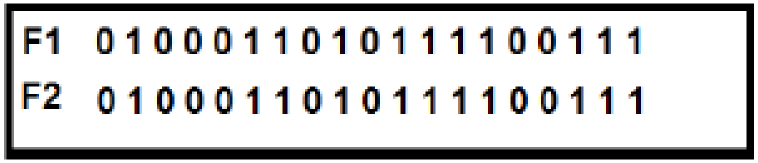
Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. Detecting Asynchrony and Dephase Change Patterns by Mining Software Repositories. Journal of Software Maintenance and Evolution: Research and Practice. 2013.

Approach: Macocha

KNN Algorithm



Bit Vector

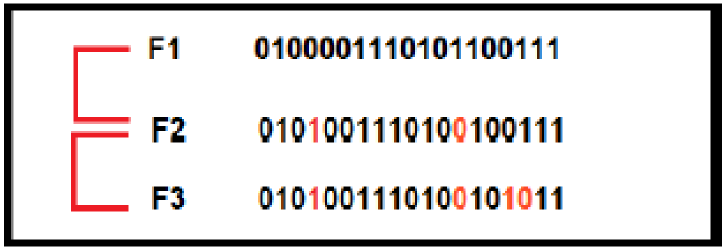


F1 0 1 0 0 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1

F2 0 1 0 0 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1

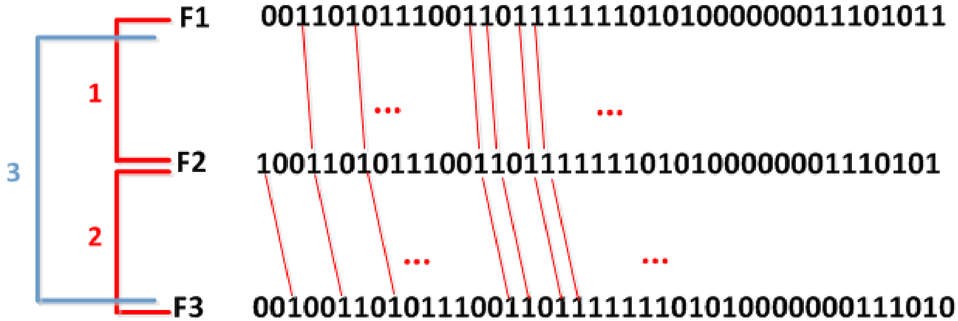
Approach: Macocha

Approximate Asynchrony Change Pattern



01010100110101
10101001101011

Dephase Change Pattern


F1 0011010111001101111111010100000001110101 1
1
3
...
...
F2 1001101011100110111111101010000000111010 1
2
...
...
F3 0010011010111001101111111010100000001110 10

The Dephase change pattern describes a set of files that always change together with some **shift** in time in their periods of changes.

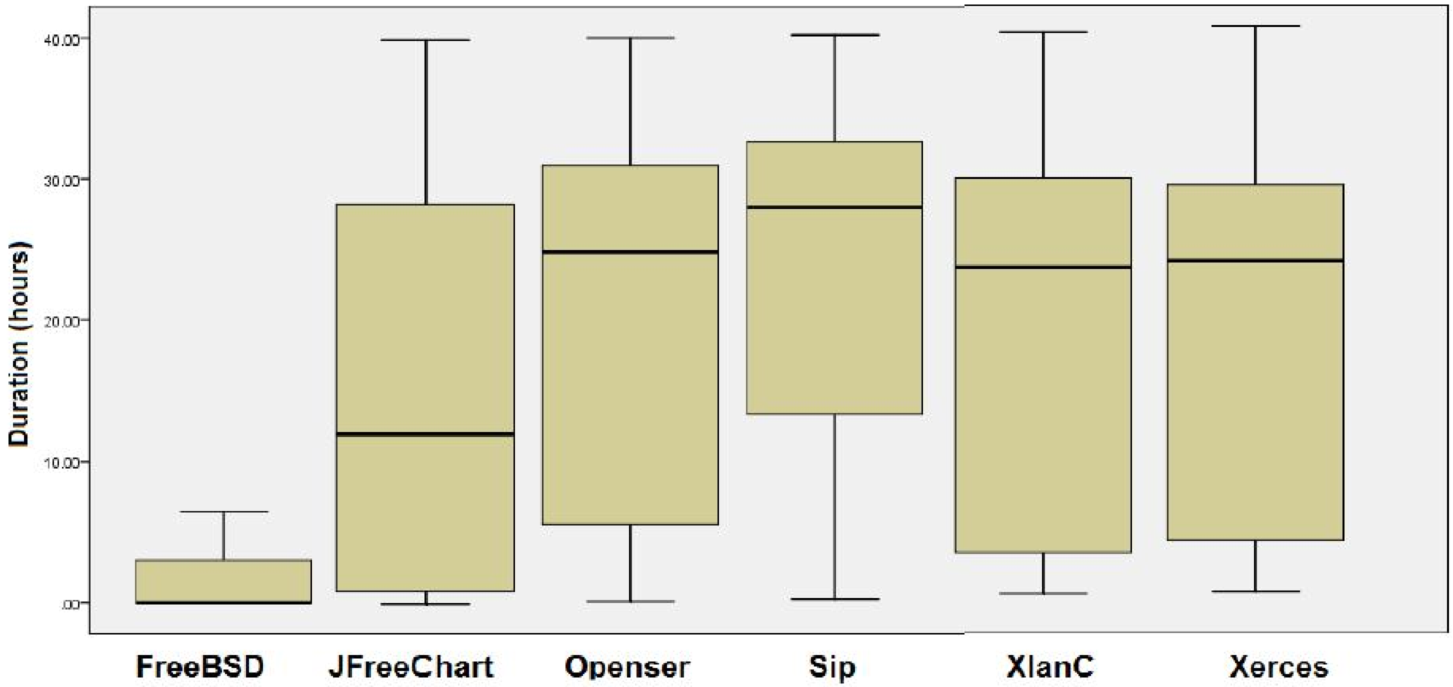
Research Questions

RQ1: Does Asynchrony and Dephase change patterns really exist in practice?

RQ2: How can they be useful?

Subjects

Systems		FreeBSD	JFreeChart	OpenSER	SIP	Xalan-C	apache Xerces
Languages	Java	C	Java	C	Java	C++	C++
# Versions	9	11	5	5	16	13	14
# Files	1,621	500	1,106	383	1,693	390	396
# Commits	6,943	50,145	1,752	5,960	6,100	3,621	3,971
# Developers	11	114	4	35	16	11	26



Analysis Methods

Quantitatively, we compare the findings of Macocha with that of the previous co-change analysis.

Qualitatively, we use external information provided by **bugs reports**, **mailing lists**, and **requirement descriptions** to validate the novel change patterns.

Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. An exploratory study of macro co-changes. In Proceedings of the 18th Working Conference on Reverse Engineering. 2011.

Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. Detecting Asynchrony and Dephase Change Patterns by Mining Software Repositories. Journal of Software Maintenance and Evolution: Research and Practice. 2013.

Results

RQ1: Does Asynchrony and Dephase change patterns really exist in practice? ✓ **YES**

RQ2: How can they be useful?

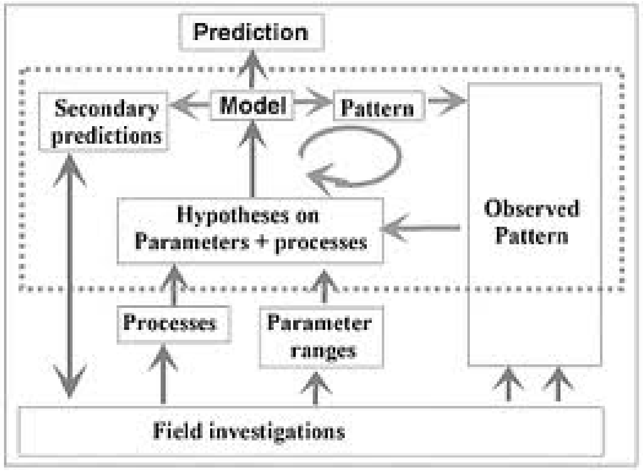
✓ **Change Propagation**

✓ **Fault Undrestanding**

✓ **Team Management**

We could detect change patterns in **long time intervals,** performed **by different developers** and with different log **messages.**

Software Evolution Impacts?



Prediction

Secondary predictions

Model

Pattern

Hypotheses on Parameters + processes

Observed Pattern

Processes

Parameter ranges

Field investigations

```
} retu
urn 1;
// NML Symbol Loo
const char *nml_x
{
switch(type)
case A_TYPE:
return "A";
default:
return"UNKNO
break;
}
}
return(N
```

Motivation

Change-Log Approaches use pr
versioning system: rec
most probab

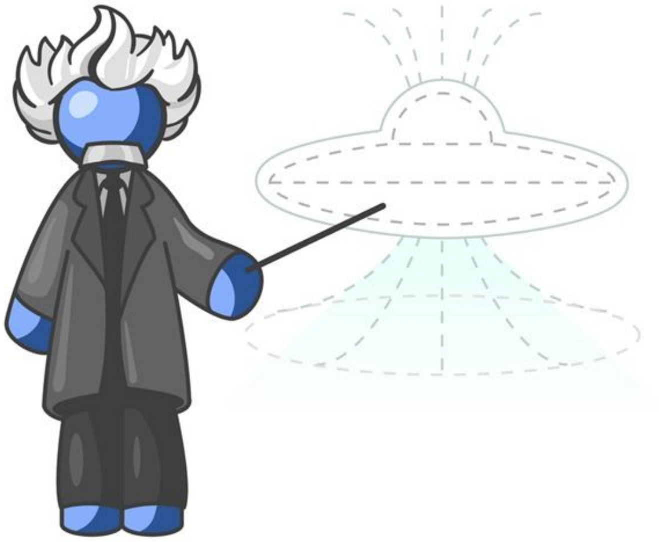
C code metrics: complex or larger
cl g.

Assuming that all classes are considered to have the same likelihood for fault-proneness is not realistic!

Ostrand et al. found that **20% of classes contains 80% of faults**.

Not all classes are there to last forever, some are meant for experimentation, so it could be expected that they have more **faults**.

Goal 2: Relating Software Evolution and Fault-proneness



Classes that exhibit similar evolution profiles may have interdependencies among them.

However, it is not **clear** how classes with similar evolution behavior are linked with **faults**.

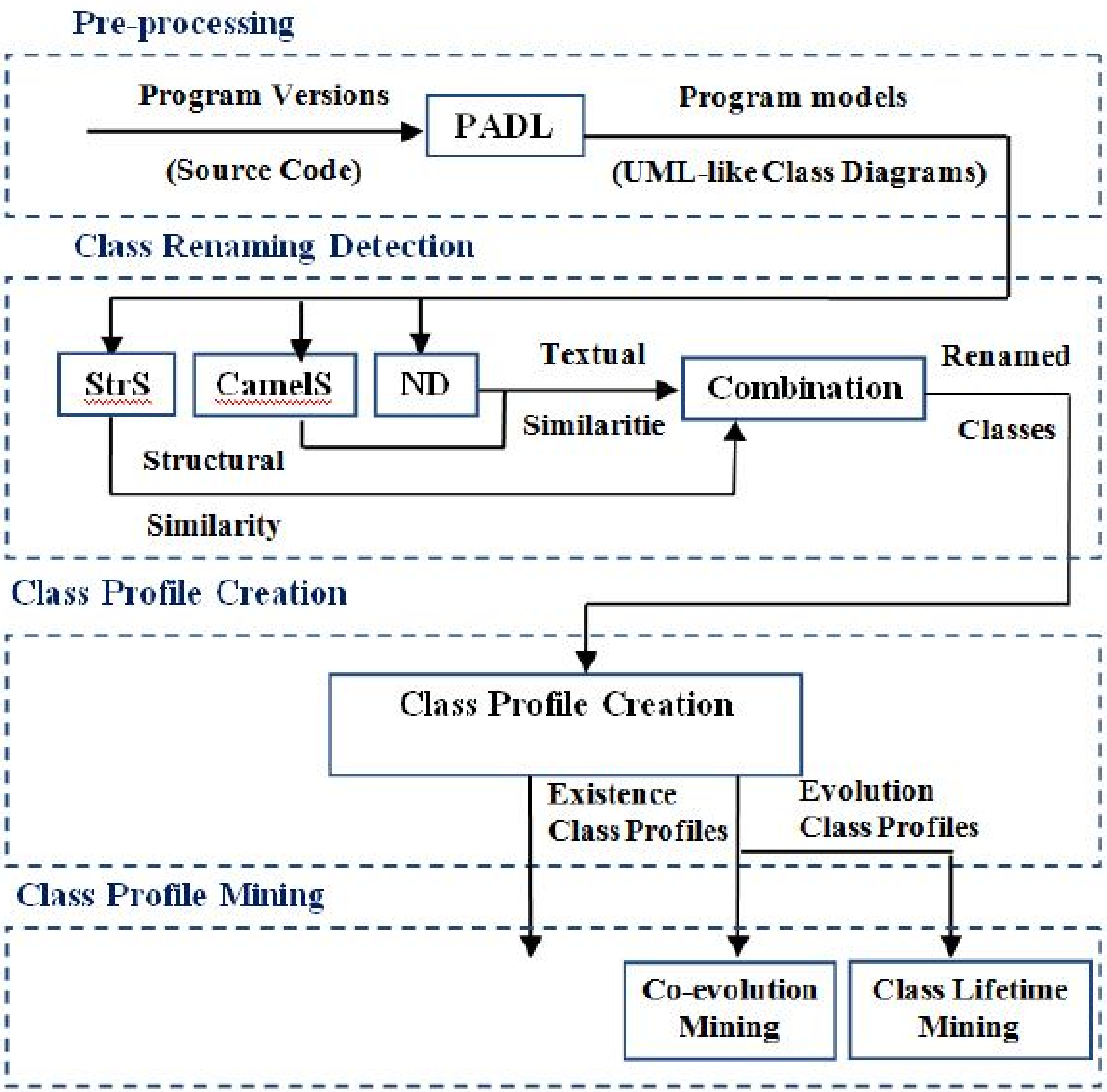
How we can relate the **evolution of classes** in object-oriented programs with **fault-proneness**?

Example

In **JFreeChart**, we find that **ChartPanel** and **CombinedDomainXYPlot** were introduced, changed, and renamed in the same versions but in different periods and by different developers.

The bug **ID 195003710** reported “a bug either in ChartPanel or CombinedDomainXYPlot when trying to zoom in/out on the range axis”.

Approach: Profilo



Fehmi Jaafar, Salima Hassaine, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Bram Adams. On the Relationship Between Program Evolution and Faultproneness: An Empirical Study. WCRE 2013, Genova, Italy.

Approach: Profilo

Short-lived classes: They have a very short lifetime.

Persistent classes: They never disappear after their first introduction

Transient classes: They appear and disappear many times.

Co-evolved classes: They have the same evolution profile and are related by static relationships.

Subjects

Systems		JFreeChart	apache Xerces
# Versions	18	46	36
Start study	2002-10-09	2000-12-01	2003-10-13
End study	2011-04-03	2011-11-20	2006-11-23
# Classes	2011	1938	892

Research Questions

RQ1: What is the relation between class lifetime and fault-proneness?

RQ2: What is the relation between class co-evolution and fault-proneness?

Analysis Methods

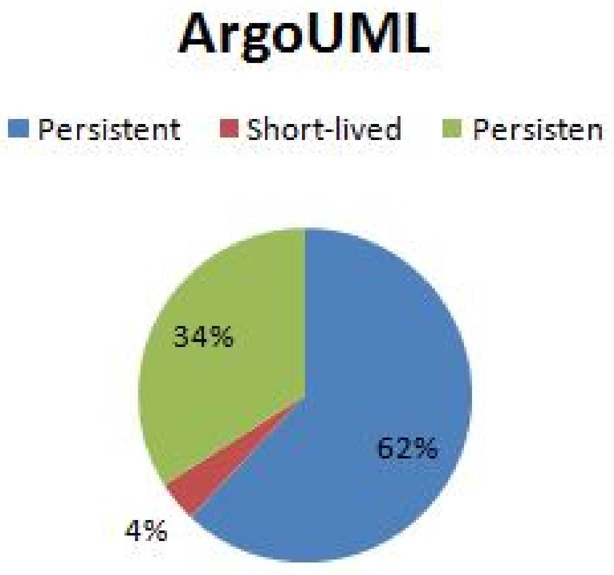
We use **Fisher's** exact test and the **Chi-Square** test to check two hypothesis.

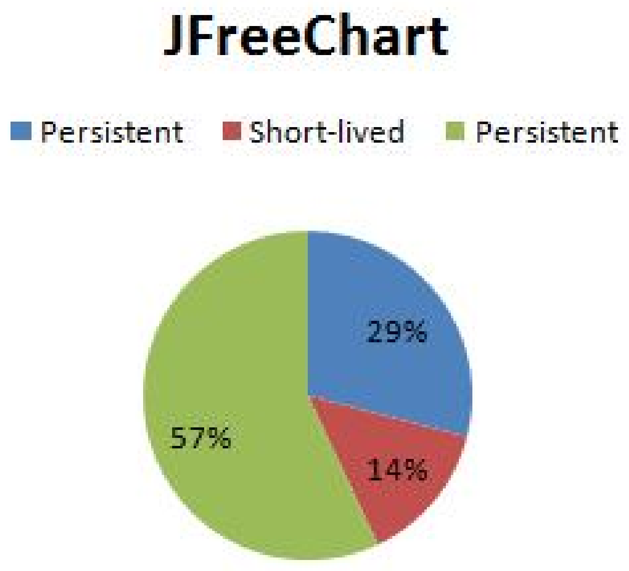
H_{RQ1}: There is no statistically significant difference between proportions of faults carried by Persistent, Shortlived, and Transient classes in systems.

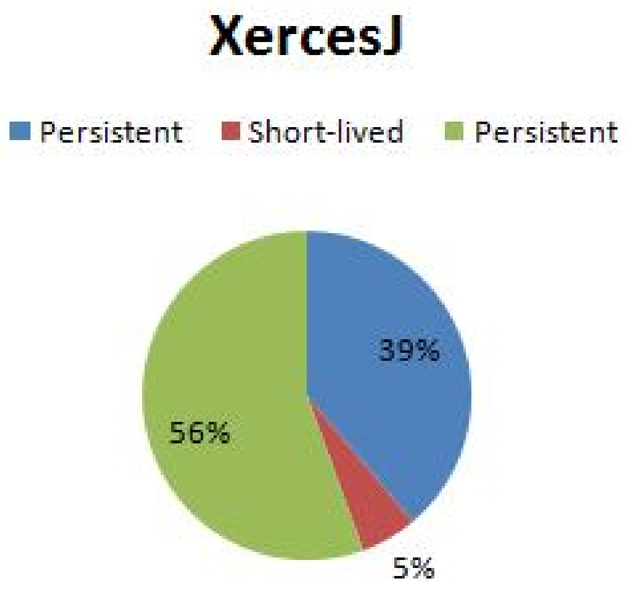
H_{RQ2}: There is no statistically significant difference between proportions of faults involving co-evolved classes or not co-evolved classes.

Results

Systems		JFreeChart	apache Xerces
# Transient	690	645	313
# Persistent	1241	1293	537
# Short-lived	80	324	42
# Co-evolution	42	11	23







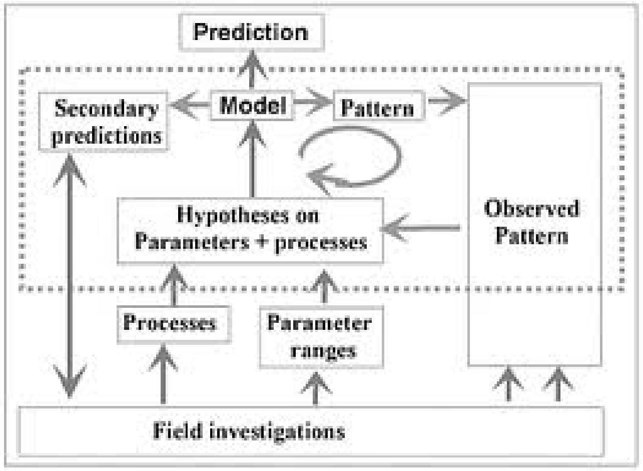
Results

Persistent classes are significantly **less fault-prone** than Short-lived and Transient classes? ✓YES

Faults fixed by maintaining **co-evolved classes** are significantly **more** than faults fixed using not co-evolved classes? ✓YES

Special attention must be given to these entities to keep the design intact during program evolution because they could have a **negative impact** on the **fault-proneness** of the program.

Software Evolution Impacts



Design Defects: Anti-patterns

Anti-patterns describe **poor solutions to design and implementation** problems…

Instead, **they indicate weaknesses in design** that may be **slowing down** development **or increasing the risk of bugs or failures** in the future.

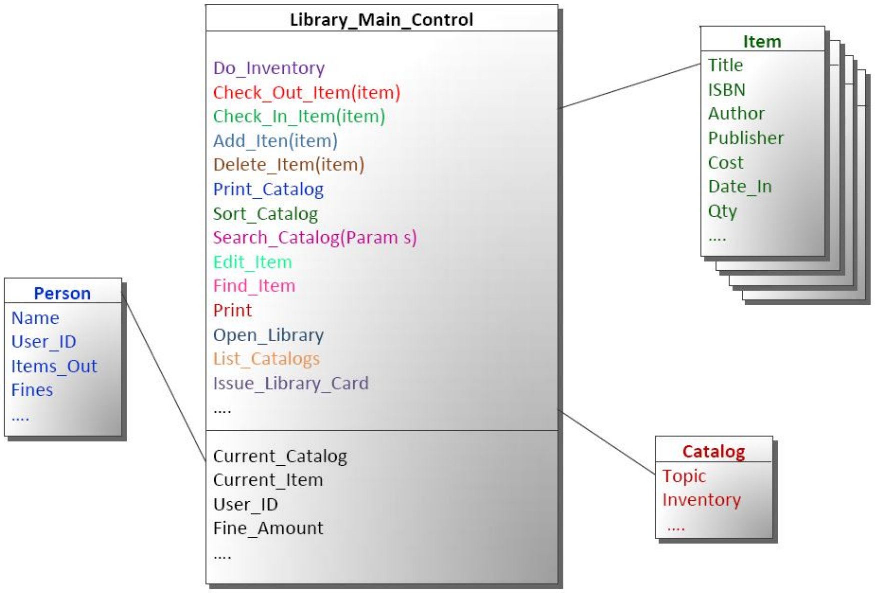
Anti Patterns

Refactoring Software, Architectures, and Projects in Crisis

William H. Brown Raphael C. Malveau
Hays W. "Skip" McCormick III Thomas J. Mowbray

Examples of Anti-patterns

Blob



Library_Main_Control

Do_Inventory
Check_Out_Item(item)
Check_In_Item(item)
Add_Iten(item)
Delete_Item(item)
Print_Catalog
Sort_Catalog
Search_Catalog(Param s)
Edit_Item
Find_Item
Print
Open_Library
List_Catalogs
Issue_Library_Card
....

Current_Catalog
Current_Item
User_ID
Fine_Amount
....

Item

Title
ISBN
Author
Publisher
Cost
Date_In
Qty
....

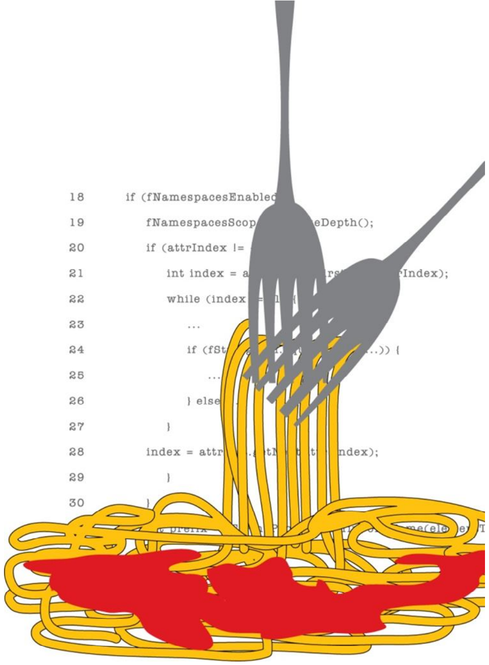
Person

Name
User_ID
Items_Out
Fines
....

Catalog

Topic
Inventory
....

Spaghetti Code

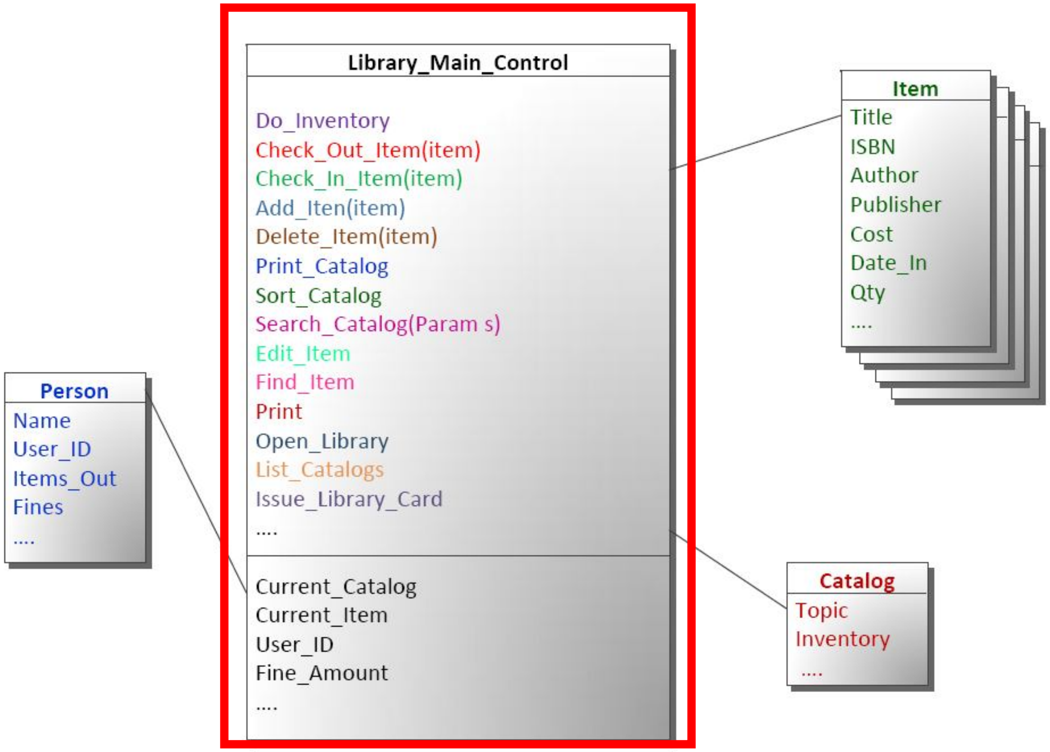


```
if (fNamespacesEnabled
    fNamespacesScop        eDepth();
    if (attrIndex !=
        int index = a          rs          Index);
        while (index      1 {
            ...
            if (fSt                    ...)) {
                ...
            } else
        }
    index = attr    .getN    t        ndex);
    }
}
```

Large controller class, **low cohesion**, associated with simple, **data-object classes…**

Process oriented methods, object **methods with no parameters**, class or **global** variables utilization, flow of execution **dictated by object implementation, not by the** clients of the objects.

Related Work



Library_Main_Control

Do_Inventory
Check_Out_Item(item)
Check_In_Item(item)
Add_Iten(item)
Delete_Item(item)
Print_Catalog
Sort_Catalog
Search_Catalog(Param s)
Edit_Item
Find_Item
Print
Open_Library
List_Catalogs
Issue_Library_Card
....

Current_Catalog
Current_Item
User_ID
Fine_Amount
....

Person

Name
User_ID
Items_Out
Fines
....

Item

Title
ISBN
Author
Publisher
Cost
Date_In
Qty
....

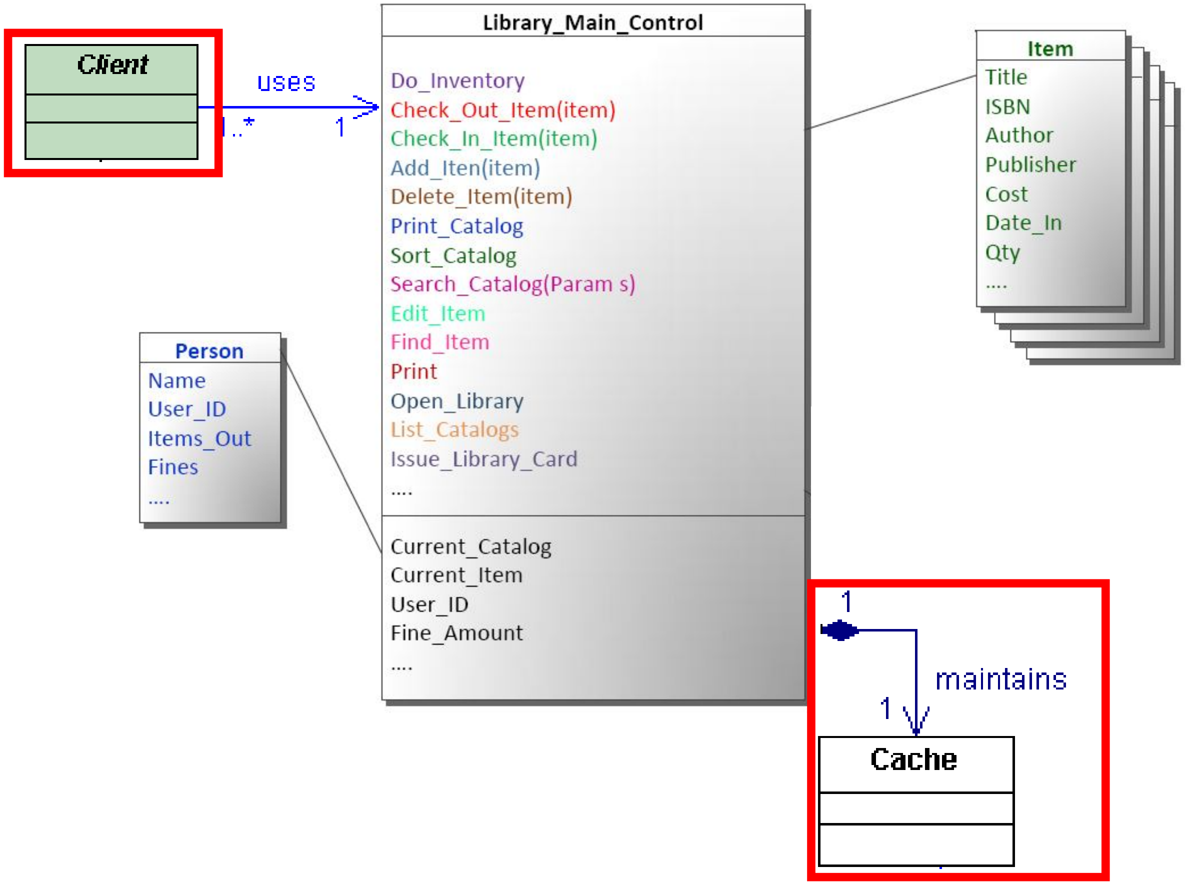
Catalog

Topic
Inventory
....

Many studies have investigated the impact of anti-patterns on

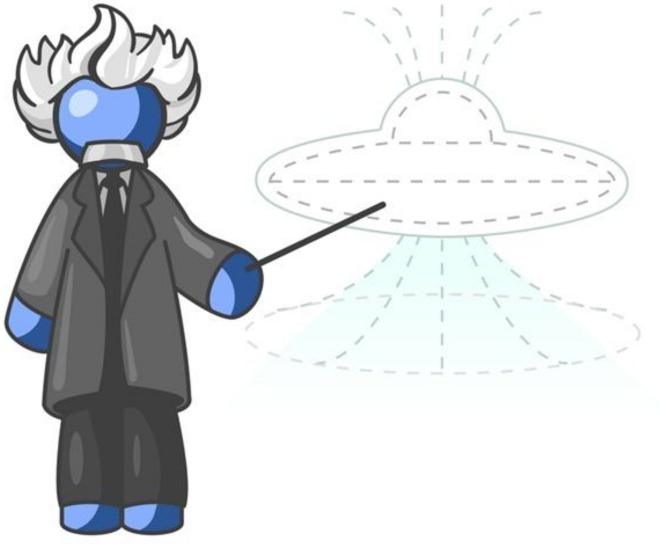
- **Maintenance [Yamashita, 2013]**
- **Fault-proneness [Khomh, 2012]**
- **Change-proneness [Romano, 2012]**
- **…**

Related Work



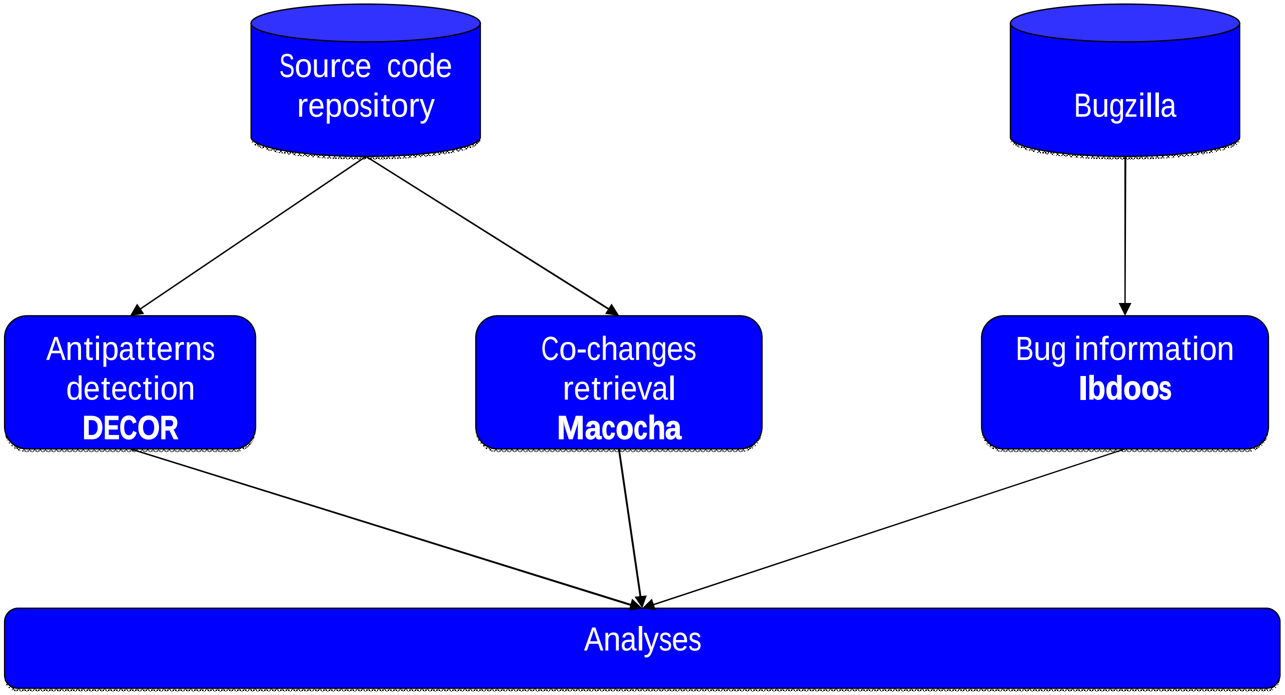
Yet, classes sharing dependencies with anti-patterns have been mostly ignored.

Goal 3: Relating Evolution, Dependencies, and Anti-patterns



Static and evolution dependencies with anti-patterns can impact the fault-proneness of classes without anti-patterns.

Approach: AntImpact



Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel , and Foutse Khomh. Mining the Relationship Between Anti-patterns Dependencies and Fault proneness. WCRE 2013.

Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Foutse Khomh. Analysing Anti-patterns Static Relationships with Design Patterns. Journal of Electronic Communications of the European Association of Software Science and Technology. 2014.

Subjects

Systems		JFreeChart	apache Xerces
# Classes	3,325	1,615	1,191
# Snapshots	4,480	2,010	159,196

Anti-patterns detected with DECOR:

- MessageChain
- RefusedParameterBequest
- SpaghettiCode
- SpeculativeGenerality
- SwissArmyKnife
- LongParameterList

- Antisingleton
- Blob
- ClassDataShouldBePrivate (CDSBP)
- ComplexClass
- LazyClass
- LongMethod

Research Questions

RQ1: Are classes that co-change with anti-patterns more fault-prone?

RQ2: Are classes that have static relationships with anti-patterns more fault-prone?

Analysis Methods

We divide classes in the systems based on their static relationships (respectively co-changes) with anti-patterns.

We use Fisher’s exact test and Odds ratios to test the hypothesis.

H_{RQ}: The proportions of faults carried by classes having static relationships (respectively co-changes) with anti-patterns and other classes are the same.

Results

Anti-patterns	Systems	# of CC	# of S.R.
Anti singleton		13	152
	JFreeChart	20	201
	Xerces	18	188
Blob		51	304
	JFreeChart	36	164
	Xerces	24	93
CDSBP		4	167
	JFreeChart	0	82
	Xerces	0	113
ComplexClass		2	192
	JFreeChart	0	146
	Xerces	0	96
LongMethod		42	282
	JFreeChart	51	314
	Xerces	0	266
LongParameterList		12	344
	JFreeChart	0	276
	Xerces	0	309

Anti-patterns	Systems	# of CC	# of S.R.
MessageChains		48	244
	JFreeChart	8	196
	Xerces	16	183
RefusedParentBequest		47	326
	JFreeChart	6	183
	Xerces	25	93
Spaghetti Code		0	0
	JFreeChart	0	0
	Xerces	0	0
SpeculativeGenerality		13	128
	JFreeChart	4	139
	Xerces	8	201
SwissArmyKnife		20	69
	JFreeChart	9	142
	Xerces	18	108

RQ1: Static relationships and anti-patterns on fault-proneness?

	Faults	No-Faults	Odd Ratios
Total of classes related to AP	1939	1350	2.22
Classes with S.R with AP and that are not AP.	945	778	1.88
Total of other classes	1117	1725	1
Classes with S.R. with AP	1062	1003	
Classes with S.R with AP and that are not AP	402	600	
Other classes	681	579	
Classes with S.R. with AP	432	226	
Classes with S.R with AP and that are not AP.	281	103	
Other classes	310	647	
Classes with S.R. with AP	445	121	
Classes with S.R with AP and that are not AP.	262	75	
Other classes	126	499	

✓YES

RQ2: Co-changes and anti-patterns on fault-proneness?

	Faults	No-Faults	Odd Ratios
Total of classes co-changing with AP	346	149	2.5
Classes co-changing with AP and that are not AP	173	81	2.3
Total of other classes	2710	2926	1

Classes co-changing with AP	241	102	
Classes co-changing with AP and that are not AP	120	59	
Other classes	1502	1480	

JFreeChart

Classes co-changing with AP	68	26	
Classes co-changing with AP and that are not AP	33	10	
Other classes	674	847	

apache Xerces

Classes co-changing with AP	37	21	
Classes co-changing with AP and that are not AP	20	12	
Other classes	534	599	

✓YES

Some Observations

We found **no class having a static dependency** (i.e., use, association, aggregation, and composition relationships) or that **co-changed** with **a SpaghettiCode**.

Many anti-patterns relationships were with classes **playing roles in design patterns**.

Classes having static relationships with Blob, ComplexClass, and SwissArmyKnife are significantly **more fault prone than** other classes with similar complexity, change history, and code size.

Classes that are co-changing with anti-patterns classes are significantly more fault prone than other classes with similar complexity, change history, and code size.

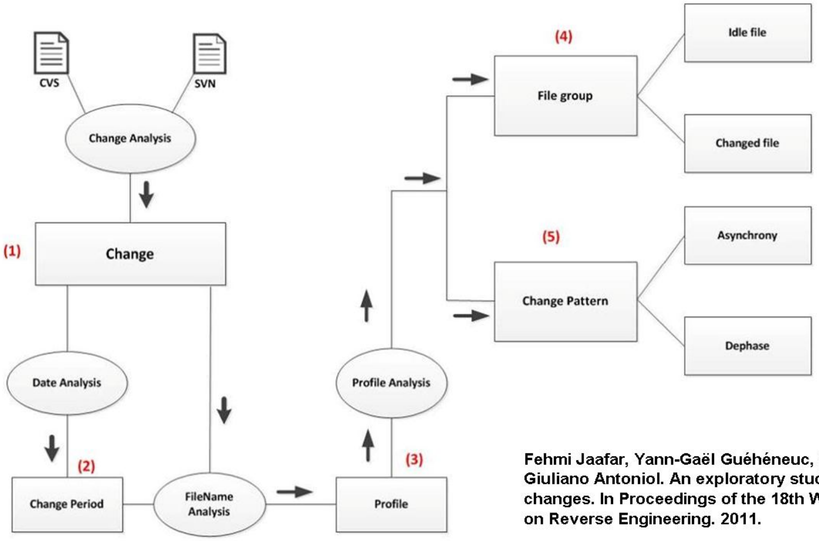
Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel , and Foutse Khomh. Mining the Relationship Between Anti-patterns and Design Patterns. PPAP 2013.

Software Evolution Impacts

As Software systems evolved, their designs become **more complex** over time and harder to change.

In absence of knowledge on the artefacts' **dependencies**, developers could introduce design defects and faults.

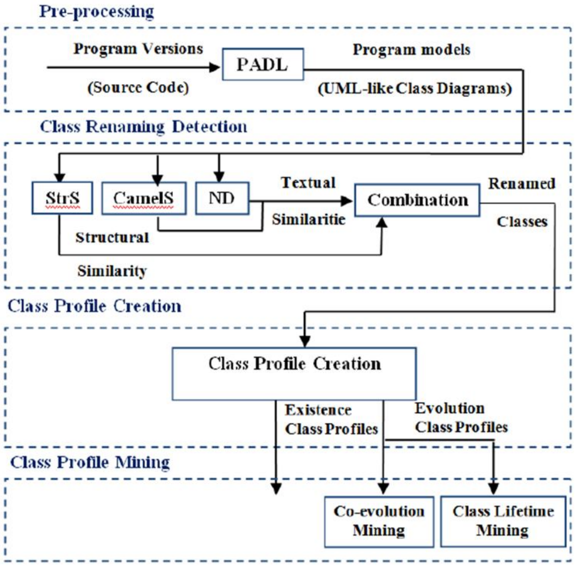
Approach: Macocha



Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. An exploratory study of macro co-changes. In Proceedings of the 18th Working Conference on Reverse Engineering. 2011.

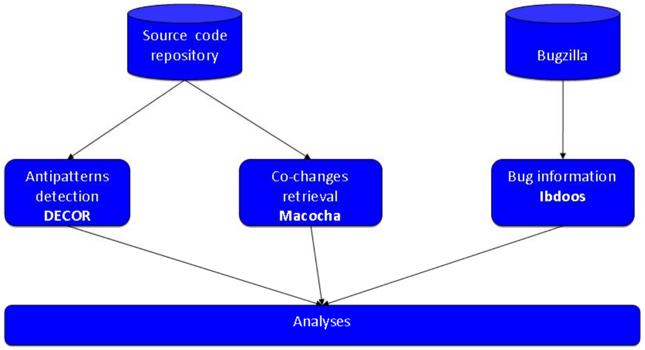
Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. Detecting Asynchrony and Dephase Change Patterns by Mining Software Repositories. Journal of Software Maintenance and Evolution: Research and Practice. 2013.

Approach: Profilo



Fehmi Jaafar, Salima Hassaine, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Bram Adams. On the Relationship Between Program Evolution and Faultproneness: An Empirical Study. WCRE 2013, Genova, Italy.

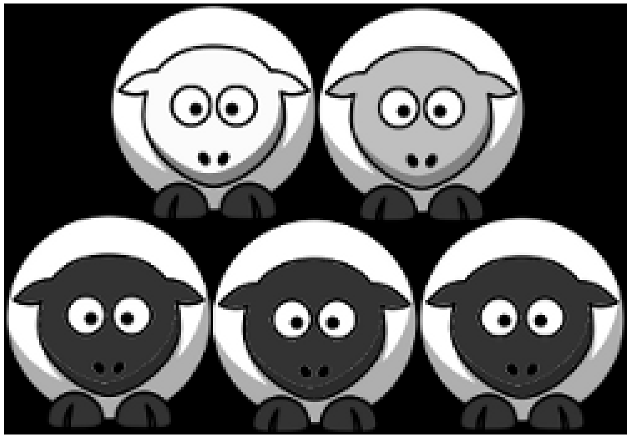
Approach: AntImpact



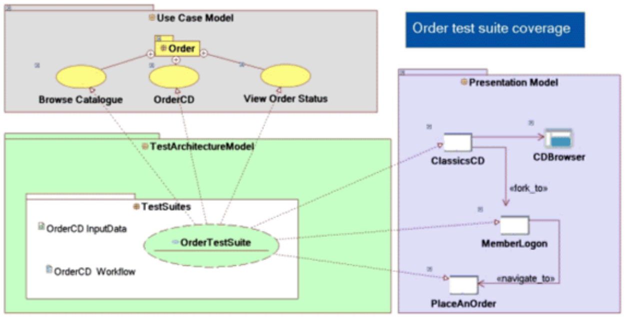
Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel , and Foutse Khomh. Mining the Relationship Between Anti-patterns Dependencies and Fault proneness. WCRE 2013.

Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Foutse Khomh. Analysing Anti-patterns Static Relationships with Design Patterns. Journal of Electronic Communications of the European Association of Software Science and Technology. 2014.

Perspectives



Co-change and **co-evolution** patterns in **other contexts**



Design defects **evolution**