

An Exploratory Study of the Impact of Code Smells on Software Change- proneness



Foutse Khomh, Massimiliano Di Penta,
and Yann-Gaël Guéhéneuc

WCRE'09

2009/10/13



Context and Motivation

- Code smells are “poor” solutions to recurring implementation problems
- Code smells are in between design and implementation
- Code smells are conjectured in the literature to hinder object-oriented software evolution
- No previous study has investigated the relationship between code smells and change-proneness



Objective

- Study the impact of code smells on the change-proneness of classes
- Study the impact of the composition of code smells on the change-proneness of classes



Outline

- Study Definition
- Study Context
- Study Design
- Analysis Method
- Results
- Conclusion and Future Work



Study Definition

(1/2)

- **Goal:** investigate the relation between the presence of smells in classes and their change-proneness
- **Quality focus:** the increase of maintenance effort and cost is due to the presence of code smells



Study Definition

(2/2)

- **Perspective:** both researchers, practitioners, and managers should be aware of the impact of code smells on classes to make inform design and implementation choices
 - The presence of change-prone classes increases maintenance effort and cost



Study Context

(1/2)

■ Programs

- 9 versions of Azureus (5,858,041 LOCs)
- 13 versions of Eclipse (31,579,975 LOCs)

■ Change history between each analysed releases in the programs' concurrent Versions System (CVS)

Study Context

(2/2)

- 29 code smells

AbstractClass	ChildClass
ClassGlobalVariable	ClassOneMethod
ComplexClassOnly	ControllerClass
DataClass	FewMethods
FieldPrivate	FieldPublic
FunctionClass	HasChildren
LargeClass	LargeClassOnly
LongMethod	LongParameterListClass
LowCohesionOnly	ManyAttributes
MessageChainsClass	MethodNoParameter
MultipleInterface	NoInheritance
NoPolymorphism	NotAbstract
NotComplex	OneChildClass
ParentClassProvidesProtected	RareOverriding
TwoInheritance	

- Detected using our tool, DECOR (based on metrics and lexical information)

Study Design

(1/3)

- **RQ1:** What is the relation between smells and change proneness?
 - H_{01} : the proportion of classes undergoing at least one change between two releases does not significantly differ between classes with code smells and other classes.
- **RQ2:** What is the relation between the number of smells in a class and its change-proneness?
 - H_{02} : the numbers of smells in change-prone classes are not significantly higher than the numbers of smells in classes that do not change.

Study Design

(2/3)

- **RQ3:** What is the relation between particular kinds of smells and change proneness?
 - H_{03} : classes with particular kinds of code smells are not significantly more change-prone than other classes.



Study Design

(3/3)

■ Independent Variables

- 29 kind of code smells.
- Each variable $s_{i,j,k}$ indicates the number of times a class i has a smell j in a release r_k .

■ Dependent variables

- Classes change proneness.
- Measured as the number of changes $c_{i,k}$ that a “smelly class” underwent between release r_k and subsequent release r_{k+1} .

Analysis Method

(1/3)

- For **RQ1**, we:
 - Test the proportions of classes exhibiting (or not) at least one change with (some) smells vs. other classes.
 - Use Fisher's exact test.
 - We compute the Odds ratios (OR).

Analysis Method

(2/3)

- For **RQ2**, we:

- Compare the numbers of smells in change-prone classes vs. non-change-prone classes.
- Use a Mann-Whitney non-parametric test.
- Estimate the magnitude of differences with the Cohen d effect size
 - Small: $0.2 \leq d \leq 0.3$, medium: around 0.5, and large: ≥ 0.8 .



Analysis Method

(3/3)

- For **RQ3**, we:
 - Relate change-proneness with the presence of particular kinds of smells.
 - We use a logistic regression model.
 - We decide that a relation is significant if the null-hypothesis is rejected for at least 75% of the releases.

Results

(1/6)

RQ1: smells and changes (Azureus)

Releases	Smells-Changes	Smells-No Changes	No Smells-Changes	No Smells-No Changes	<i>p</i> -values	<i>OR</i>
3.1.0.0	220	1967	20	1433	< 0.01	8.01
3.1.1.0	564	1686	101	1381	< 0.01	4.57
4.0.0.0	83	2238	7	1519	< 0.01	8.05
4.0.0.2	106	2206	12	1510	< 0.01	6.04
4.0.0.4	435	1886	39	1484	< 0.01	8.77
4.1.0.0	50	2297	11	1533	< 0.01	3.03
4.1.0.2	112	2235	11	1533	< 0.01	6.98
4.1.0.4	112	2236	12	1532	< 0.01	6.39
4.2.0.0	37	2353	3	1580	< 0.01	8.28

Results

(2/6)

RQ1: smells and changes (Eclipse)

Releases	Smells-Changes	Smells-No Changes	No Smells-Changes	No Smells-No Changes	<i>p</i> -values	<i>OR</i>
1.0	2042	1731	417	448	< 0.01	1.27
2.0	3673	1373	767	236	0.02	0.82
2.1.1	2224	3838	193	964	< 0.01	2.89
2.1.2	2400	3664	359	798	< 0.01	1.46
2.1.3	2942	3125	516	642	0.01	1.17
3.0	3415	4880	684	1032	0.32	1.06
3.0.1	6216	2087	1294	423	0.69	0.97
3.0.2	5784	2520	1194	524	0.91	1.01
3.2	1819	9621	115	2210	< 0.01	3.63
3.2.1	2778	8680	291	2038	< 0.01	2.24
3.2.2	3321	8144	409	1921	< 0.01	1.92
3.3	1778	10844	145	2364	< 0.01	2.67
3.3.1	4337	8290	682	1830	< 0.01	1.40

Results

(3/6)

- RQ2:
composition
of smells and
changes
(Azureus)

Releases	M-W <i>p</i>	<i>t</i> -test <i>p</i>	Cohen <i>d</i>
3.1.0.0	< 0.01	< 0.01	0.72
3.1.1.0	< 0.01	< 0.01	0.71
4.0.0.0	< 0.01	< 0.01	1.01
4.0.0.2	< 0.01	< 0.01	0.86
4.0.0.4	< 0.01	< 0.01	0.83
4.1.0.0	< 0.01	< 0.01	0.59
4.1.0.2	< 0.01	< 0.01	0.93
4.1.0.4	< 0.01	< 0.01	0.85
4.2.0.0	< 0.01	< 0.01	1.02

Results

(4/6)

- RQ2:
composition
of smells and
changes
(Eclipse)

Releases	M-W <i>p</i>	<i>t</i> -test <i>p</i>	Cohen <i>d</i>
1.0	0.79	0.03	0.06
2.0	< 0.01	< 0.01	-0.08
2.1.1	< 0.01	< 0.01	0.31
2.1.2	< 0.01	< 0.01	0.13
2.1.3	0.04	< 0.01	0.07
3.0	0.07	0.10	0.03
3.0.1	0.11	0.26	-0.03
3.0.2	0.12	0.28	-0.02
3.2	< 0.01	< 0.01	0.41
3.2.1	< 0.01	< 0.01	0.29
3.2.2	< 0.01	< 0.01	0.25
3.3	< 0.01	< 0.01	0.41
3.3.1	< 0.01	< 0.01	0.18

Results

- RQ3: kinds of smells and changes (Azureus)

Smells	Proneness to Changes
AbstractClass	5
ChildClass	3
ClassGlobalVariable	2
ClassOneMethod	1
ComplexClassOnly	2
ControllerClass	2
DataClass	4
FewMethods	2
FieldPrivate	1
FieldPublic	2
FunctionClass	2
HasChildren	1
LargeClass	5
LargeClassOnly	–
LongMethod	–
LongParameterListClass	1
LowCohesionOnly	2
ManyAttributes	–
MessageChainsClass	4
MethodNoParameter	2
MultipleInterface	4
NoInheritance	3
NoPolymorphism	3
NotAbstract	7
NotComplex	2
OneChildClass	1
ParentClassProvidesProtected	–
RareOverriding	1
TwoInheritance	–

(5/6)

Results

- RQ3: kinds of smells and changes (Eclipse)

Smells	Proneness to Changes
AbstractClass	1
ChildClass	6
ClassGlobalVariable	2
ClassOneMethod	4
ComplexClassOnly	8
ControllerClass	4
DataClass	4
FewMethods	2
FieldPrivate	6
FieldPublic	8
FunctionClass	1
HasChildren	11
LargeClass	8
LargeClassOnly	–
LongMethod	9
LongParameterListClass	6
LowCohesionOnly	5
ManyAttributes	9
MessageChainsClass	10
MethodNoParameter	8
MultipleInterface	5
NoInheritance	–
NoPolymorphism	3
NotAbstract	1
NotComplex	10
OneChildClass	2
ParentClassProvidesProtected	–
RareOverriding	4
TwoInheritance	–

(6/6)



Discussion

- Classes with smells are more change-prone, some odds ratio 3 to 8 times bigger for these classes.
- HasChildren, MessageChains, NotComplex, and NotAbstract lead almost consistently to change-prone classes.
- Existing smells are generally removed from the system while some new are introduced in the context of new features addition.

Threats to the validity (1/2)

■ Construct validity

- The count of changes occurred to classes is based on the CVS change log.
- Subjectivity of the definitions of code smells
 - We were interested in code smells **as defined** by our tool.
- Dependence between code smells
 - We perform a Spearman rank correlation analysis.
 - Our logistic regression only selected non-correlated smells.

Threats to the validity

(2/2)

■ Internal validity

- We do not claim causation, only relation.

■ Conclusion validity

- Statistic tests properly used.
 - We verified their underlying assumptions.

■ Reliability validity

- We provide all the details for the replication of our study.

■ External validity

- The generalisation will require further studies.
 - We chose programs from different domains and a representative set of code smells.

Conclusion

(1/2)

- We provide empirical evidence of the negative impact of code smells on classes change-proneness.
- We show that:
 - Classes with smells are significantly more likely to be the subject of changes, than other classes.
 - The more a class have smells, the more it changes.
 - Some specific code smells, are more likely to be of concern during evolution: e.g., MessageChains, NotComplex (and Lazy classes)

Conclusion

(2/2)

- Previous studies correlated source code metrics with change-proneness.
 - But, we believe that smells can provide to developers recommendations easier to understand than what metric profiles can do.
 - However, we found that metrics still perform better than smells on building change-proneness/fault-proneness prediction models.



Future work

- Our future work includes:
 - Replicating this study
 - Studying the effect of anti patterns which are at a higher level of abstraction
 - Relates smells, anti patterns to fault-proneness

The data of our study are available online

<http://www.ptidej.net/downloads/experiments/prop-WCRE09>

Questions



Thank you for listening