When and Why Your Code Starts to Smell Bad

Michele Tufano 1, Fabio Palombay 2, 3 Gabriele Bavotaz, 4 Rocco Olivetox,

3 Massimiliano Di Pentaz, 2 Andrea De Luciay, 1 Denys Poshyvanyk

1 The College of William and Mary, Williamsburg, VA, USA – 2 University of Salerno, Fisciano (SA), Italy

3 University of Sannio, Benevento, Italy – 4 University of Molise, Pesche (IS), Italy

INTRODUCTION

- Common wisdom suggests that urgent maintenance activities and pressure to deliver features while prioritizing time-to-market over code quality are often the causes of such smells.
- Generally speaking, software evolution has always been considered as one of the reasons behind “software aging” or “increasing complexity”.

INTRODUCTION - Dataset

- Study empirical large-scale conducted on the evolution history of 200 open source projects belonging to three software ecosystems: Android, Apache and Eclipse (Table I).
  - 0,5M commits
  - 9.164 of them classified as smell-introducing

TABLE I
CHARACTERISTICS OF ECOSYSTEMS UNDER ANALYSIS

Ecosystem	#Proj.	#Classes	KLOC	#Commits	#Issues	Mean Story Length	Min-Max Story Length
Apache	100	4-5,052	1-1,031	207,997	3,486	6	1-15
Android	70	5-4,980	3-1,140	107,555	1,193	3	1-6
Eclipse	30	142-16,700	26-2,610	264,119	124	10	1-13
Overall	**200**	-	-	**579,671**	**4,803**	**6**	**1-15**

STUDY DESIGN- RQs

- RQ1: When smells are introduced in software projects?
  - ✓ Smells are introduced as soon as a code entity is created, or
  - ✓ Smells are introduced in the context of specific maintenance activities, or
  - ✓ Smells appear “gradually” during software evolution.
- RQ2: Why they are introduced?
  - ✓ Factors that are indicated as possible causes for code smell introduction:
    - ✓ The developer implementing a new feature or fixing a bug?,
    - ✓ The project status (e.g., is the change performed in proximity to a major release deadline?),
    - ✓ The developer status (e.g., is the developer a newcomer or a senior project member?).

STUDY DESIGN– Type of Smells

- Focus on the following types of smells :

  - ✓ 1) **Blob Class**: a large class with different responsibilities that monopolizes most of the system's processing.
  - ✓ 2) **Class Data Should be Private**: a class exposing its attributes, violating the information hiding principle.
  - ✓ 3) **Complex Class**: a class having a high cyclomatic complexity.
  - ✓ 4) **Functional Decomposition**: a class where inheritance and polymorphism are poorly used, declaring many private fields and implementing few methods.
  - ✓ 5) **Spaghetti Code**: a class without structure that declares long methods without parameters.

STUDY DESIGN- Data Extraction and Analysis

- RQ1: When are code smells introduced?

  - ✓ Firstly cloned the 200 Git repositories
  - ✓ Then, analyzed each repository “ri” using a tool HistoryMiner
  - ✓ HistoryMiner – purpose of identifying smell-introducing commits.
    - ✓ The tool checks each commit in chronological order, and runs an implementation of the DECOR smell detector.

      DECOR identifies smells using detection rules based on the values of internal quality metrics. The choice of using DECOR was driven by the fact that:

      (i) it is a state-of-the-art smell detector having a high accuracy in detecting smells;

      (ii) it applies simple detection rules that allow it to be very efficient.
  - ✓ As an output, tool produces, for each source code file fj ϵ ri the list of commits in which fj has been involved, specifying if fj has been added, deleted, or modified and if fj was affected, in that specific commit, by one of the five considered smells.

STUDY DESIGN- Data Extraction and Analysis – RQ1

- RQ1: When are code smells introduced?

- When analyzing the number of commits needed for a smell to affect a code component, we can fall into two possible scenarios:

  – In the first scenario smell instances are introduced during the creation of source code artifacts, i.e., in the first commit involving a source code file.

STUDY DESIGN- Data Extraction and Analysis - RQ1

– In the second scenario, smell instances are introduced after several commits and, thus as result of multiple maintenance activities.

- Besides running the DECOR smell detector for the project snapshot related to each commit, the HistoryMiner also computes, for each snapshot and for each source code artifact, a set of quality metrics (Table II).

- For example, expect that classes becoming Blobs will exhibit a higher growth rate than classes that are not going to become Blobs.

TABLE II
QUALITY METRICS MEASURED IN THE CONTEXT OF $\mathbf{RQ}_1$

Metric	Description
Lines of Code (LOC)	The number of lines of code excluding white spaces and comments
Weighted Methods per Class (WMC) [16]	The complexity of a class as the sum of the McCabes cyclomatic complexity of its methods
Response for a Class (RFC) [16]	The number of distinct methods and constructors invoked by a class
Coupling Between Object (CBO) [16]	The number of classes to which a class is coupled
Lack of COhesion of Methods (LCOM) [16]	The higher the pairs of methods in a class sharing at least a field, the higher its cohesion
Number of Attributes (NOA)	The number of attributes in a class
Number of Methods (NOM)	The number of methods in a class

STUDY DESIGN- Data Extraction and Analysis - RQ1

- Function that best approximates the data distribution: linear function [1].
  - For each file fj ϵ ri, the regression line of its quality metric values:
    - If file fj is affected by a specific smell k, we compute the regression line considering the quality metric values computed for each commit involving fj from the first commit (i.e., where the file was added to the versioning system) to the commit where the instance of smell k was detected in fj.

STUDY DESIGN- Data Extraction and Analysis - RQ1

- Function that best approximates the data distribution: linear function.
  - For each file fj ϵ ri, the regression line of its quality metric values:
    - Instead, if fj is not affected by any smell, we considered only the first n commits involving the file fj , where n is the average number of commits required by smell k to affect code instances.

STUDY DESIGN- Data Extraction and Analysis - RQ1

- Function that best approximates the data distribution: linear function.
  - For each file fj ϵ ri, the regression line of its quality metric values:
    - Then, for each metric reported in Table II, we compared the distributions of regression line slopes for cleanly and smelly files.
      - The comparison is performed using a two-tailed Mann-Whitney U test.
      - Also estimated the magnitude of the observed differences using the Cliff's Delta (or d).

STUDY DESIGN- Data Extraction and Analysis – RQ2

- RQ2: Why are code smells introduced?
  - To identify the smell-introducing commits for a file fj affected by an instance of a specific smell (smell k), we designed the following heuristic:

    - If smell k has been introduced in the commit c1 where fj has been added to the repository, then c1 is the smell-introducing commit.

STUDY DESIGN- Data Extraction and Analysis - RQ2

- RQ2: Why are code smells introduced?
  - To identify the smell-introducing commits for a file fj affected by an instance of a specific smell (smellk), we designed the following heuristic:

    - else given C = {c1, c2, ..., cn} the set of commits involving fj and leading to the detection of smell k in cn we use the results of RQ1 to select the set of quality metrics "M" allowing to discriminate between the groups of files that are affected and not affected in their history by smell k.

STUDY DESIGN- Data Extraction and Analysis

- RQ2: Why are code smells introduced?

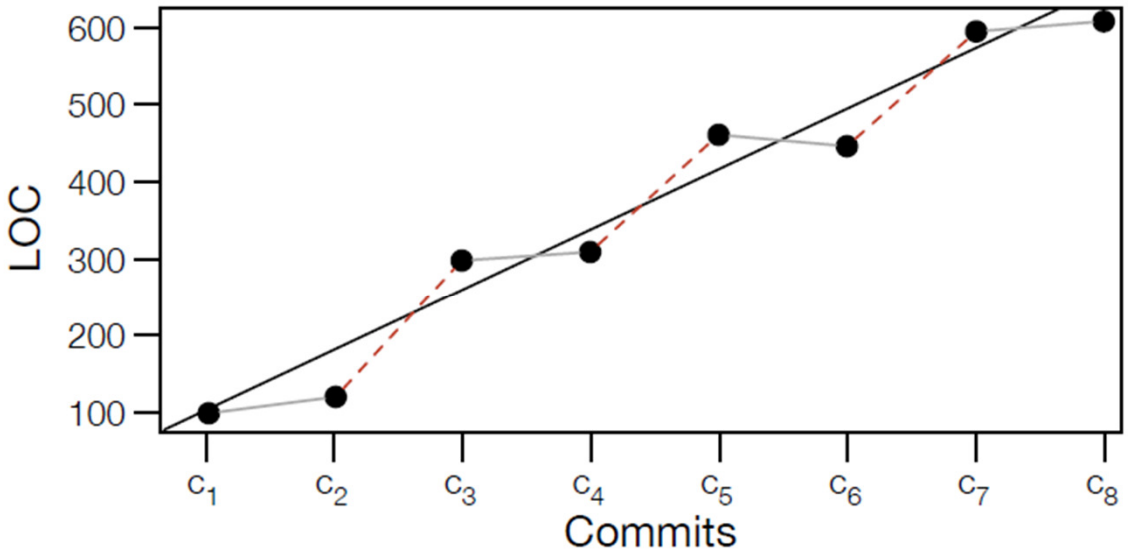

Fig. 1. Example of smell-introducing commit identification.

STUDY DESIGN- Data Extraction and Analysis

Overall, we obtained 9,164 smell-introducing commits in the 200 systems, that we used to answer RQ2. Were classified by assigning to each commit one or more tags among those reported in Table III.

- 471 (JIRA or BUGZILLA issue trackers)
- 8.693 (two of the authors manually analyzed)

TABLE III
TAGS ASSIGNED TO THE SMELL-INTRODUCING COMMITS

Tag	Description	Values
	COMMIT GOAL TAGS	
Bug fixing	The commit aimed at fixing a bug	[true,false]
Enhancement	The commit aimed at implementing an enhancement in the system	[true,false]
New feature	The commit aimed at implementing a new feature in the system	[true,false]
Refactoring	The commit aimed at performing refactoring operations	[true,false]
	PROJECT STATUS TAGS	
Working on release	The commit was performed [value] before the issuing of a major release	[one day, one week, one month, more than one month]
Project startup	The commit was performed [value] after the starting of the project	[one week, one month, one year, more than one year]
	DEVELOPER STATUS TAGS	
Workload	The developer had a [value] workload when the commit has been performed	[low,medium,high]
Ownership	The developer was the owner of the file in which the commit introduced the smell	[true,false]
Newcomer	The developer was a newcomer when the commit was performed	[true,false]

ANALYSIS OF THE RESULTS

RQ1: When are code smells introduced?

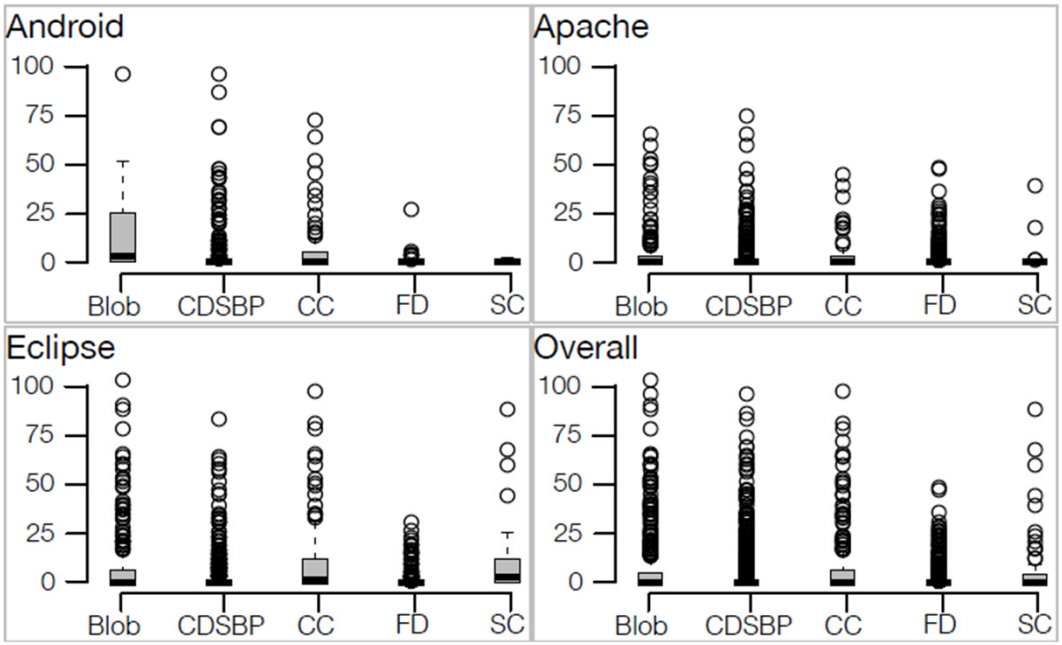

Fig. 2. Number of commits required by a smell to manifest itself.

ANALYSIS OF THE RESULTS

RQ1: When are code smells introduced?

- ✓ Fig. 2, in almost all cases the median number of commits needed by a smell to affect code components is zero, except for Blob on Android (median=3) and Complex Class on Eclipse (median=1).

- ✓ Contrary the common wisdom that smells are generally the result of continuous maintenance activities.

ANALYSIS OF THE RESULTS – RQ1

TABLE IV

RQ_1: SLOPE AFFECTED *vs* SLOPE NOT AFFECTED - MANN-WHITNEY TEST (ADJ. P-VALUE) AND CLIFF'S DELTA (*d*)

Ecosys.	Smell	Affected	LOC mean	LOC med	cmp	LCOM mean	LCOM med	cmp	WMC mean	WMC med	cmp	RFC mean	RFC med	cmp	CBO mean	CBO med	cmp	NOM mean	NOM med	cmp	NOA mean	NOA med	cmp
Android	Blob	NO	0.68	0		0.55	0		0.17	0		0.13	0		0.15	0		0.07	0		0.09	0	
		YES	32.90	12.51	↑	13.80	2.61	↑	3.78	1.81	↑	5.39	3.47	↑	1.34	0.69	↑	1.15	0.57	↑	0.49	0.13	↑
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			<0.01		
		Cliff's *d*	0.65 (L)			0.38 (M)			0.53 (L)			0.64 (L)			0.66 (L)			0.51 (L)			0.56 (L)		
	CDSP	NO	0.42	0		0.12	0		0.12	0		0.05	0		0.09	0		0.05	0		0.06	0	
		YES	4.43	1.68	↑	0.83	0	—	0.33	0	—	0.27	0	—	0.36	0.18	↑	0.17	0	—	2.60	0.69	↑
		p-value	<0.01			0.26			0.88			0.86			<0.01			0.71			<0.01		
		Cliff's *d*	0.45 (M)			0.06 (N)			-0.01 (N)			-0.01 (N)			0.26 (S)			0.02 (N)			0.78 (L)		
	CC	NO	0.67	0		0.48	0		0.19	0		0.14	0		0.15	0		0.08	0		0.09	0	
		YES	7.71	6.81	↑	11.16	4.12	↑	2.61	2.20	↑	2.42	1.01	↑	0.33	0.28	↑	0.67	0.50	↑	0.18	0.10	↑
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			<0.01		
		Cliff's *d*	0.63 (L)			0.45 (M)			0.76 (L)			0.64 (L)			0.32 (S)			0.67 (L)			0.39 (M)		
	FD	NO	0.99	0		0.62	0		0.29	0		0.31	0		0.40	0		0.11	0		0.11	0	
		YES	-10.56	-1.00	↓	-2.65	0	↓	-2.74	-0.60	↓	-3.49	0	↓	0.78	0.49	—	-1.13	-0.30	↓	-0.91	0	↓
		p-value	<0.01			<0.01			<0.01			0.02			0.09			<0.01			0.01		
		Cliff's *d*	-0.55 (L)			-0.49 (L)			-0.59 (L)			-0.42 (M)			0.32 (S)			-0.76 (L)			-0.45 (M)		
	SC	NO	1.42	0		0.96	0		0.31	0		0.42	0		0.29	0		0.11	0		0.13	0	
		YES	144.2	31.0	↑	69.17	100.00	↑	10.17	10.00	↑	6.33	5.00	↑	0.67	1.00	—	3	3	↑	0.16	0	↑
		p-value	<0.01			<0.01			<0.01			<0.01			0.50			<0.01			0.04		
		Cliff's *d*	0.99 (L)			0.98 (L)			0.99 (L)			0.95 (L)			0.22 (S)			0.99 (L)			0.68 (L)		
Apache	Blob	NO	0.40	0		0.42	0		0.13	0		0.13	0		0.05	0		0.05	0		0.03	0	
		YES	91.82	33.58	↑	384.70	12.40	↑	17.79	4.92	↑	27.61	7.09	↑	2.17	0.50	↑	7.64	1.72	↑	0.77	0.05	↑
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			<0.01		
		Cliff's *d*	0.92 (L)			0.52 (L)			0.77 (L)			0.74 (L)			0.44 (M)			0.82 (L)			0.22 (S)		
	CDSP	NO	0.43	0		0.54	0		0.12	0		0.12	0		0.10	0		0.05	0		0.03	0	
		YES	8.69	2.03	↑	2.44	0	—	0.61	0	—	0.59	0	—	0.55	0.06	↑	0.23	0	—	3.28	1.07	↑
		p-value	<0.01			0.28			0.46			0.45			<0.01			0.37			<0.01		
		Cliff's *d*	0.63 (L)			-0.04 (N)			-0.03 (N)			0.03 (N)			0.25 (S)			-0.03 (N)			0.86 (L)		
	CC	NO	0.36	0		0.47	0		0.12	0		0.13	0		0.09	0		0.05	0		0.04	0	
		YES	121.80	25.86	↑	886.50	152.40	↑	31.87	10.36	↑	39.81	7.21	↑	3.45	0.53	↑	13.99	3.56	↑	0.17	0	↑
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			0.02		
		Cliff's *d*	0.81 (L)			0.70 (L)			0.83 (L)			0.74 (L)			0.53 (L)			0.82 (L)			0.23 (S)		
	FD	NO	0.52	0		0.812	0		0.16	0		0.14	0		0.10	0		0.07	0		0.030	0	
		YES	-13.78	-3.32	↓	-5.98	-0.30	↓	-6.16	-1.00	↓	-4.81	-0.52	↓	-0.28	0	↓	-2.82	-0.53	↓	-0.40	0	↓
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			<0.01		
		Cliff's *d*	-0.72 (L)			-0.46 (M)			-0.66 (L)			-0.49 (L)			-0.14 (N)			-0.67 (L)			-0.35 (M)		
	SC	NO	0.54	0		0.11	0		0.11	0		0.12	0		0.14	0		0.04	0		0.04	0	
		YES	273.00	129.90	↑	232.30	4.50	—	7.09	6.50	↑	10.81	10.15	↑	0.96	0.92	—	3.41	3.00	↑	2.29	2.08	↑
		p-value	<0.01			0.52			<0.01			<0.01			0.12			<0.01			0.02		
		Cliff's *d*	0.94 (L)			0.17 (S)			0.91 (L)			0.95 (L)			0.44 (M)			0.94 (L)			0.63 (L)		
Eclipse	Blob	NO	0.02	0		0.02	0		-0.01	0		-0.03	0		0.13	0		-0.01	0		0.01	0	
		YES	69.51	28.15	↑	1208.00	14.71	↑	17.10	2.92	↑	18.15	2.44	↑	0.58	0.01	↑	7.11	1.09	↑	3.11	0.09	↑
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			<0.01		
		Cliff's *d*	0.86 (L)			0.54 (L)			0.76 (L)			0.65 (L)			0.20 (S)			0.75 (L)			0.50 (L)		
	CDSP	NO	0.01	0		0.34	0		<-0.01	0		-0.02	0		0.13	0		<-0.01	0		0.01	0	
		YES	12.58	2.50	↑	749.1	0	↑	2.77	0	↑	0.70	0	↑	0.37	0	—	2.10	0	↑	4.01	1	↑
		p-value	<0.01			<0.01			<0.01			<0.01			0.53			<0.01			<0.01		
		Cliff's *d*	0.65 (L)			0.13 (N)			0.16 (S)			0.12 (N)			0.03 (N)			0.18 (S)			0.90 (L)		
	CC	NO	0.02	0		0.21	0		-0.01	0		-0.05	0		0.11	0		-0.01	0		0.02	0	
		YES	57.72	18.00	↑	2349.00	141.70	↑	19.86	4.86	↑	10.46	0.82	↑	0.68	0.01	↑	10.23	1.94	↑	3.10	<0.01	↑
		p-value	<0.01			<0.01			<0.01			<0.01			0.02			<0.01			<0.01		
		Cliff's *d*	0.82 (L)			0.75 (L)			0.84 (L)			0.54 (L)			0.15 (S)			0.83 (L)			0.42 (M)		
	FD	NO	-0.02	0		0.67	0		-0.02	0		-0.02	0		0.13	0		-0.01	0		0.02	0	
		YES	-15.09	-5.40	↓	-5.23	-0.95	↓	-5.15	-1.71	↓	-4.06	-0.60	↓	-0.16	0.16	↑	-2.39	-0.60	↓	-0.35	0	—
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			0.88		
		Cliff's *d*	-0.72 (L)			-0.61 (L)			-0.71 (L)			-0.51 (L)			0.23 (S)			-0.74 (L)			-0.01 (N)		
	SC	NO	0.07	0		1.19	0		0.02	0		-0.06	0		0.15	0		-0.01	0		0.02	0	
		YES	114.40	42.74	↑	698.4	137.3	↑	16.65	4.03	↑	9.47	0.03	↑	1.37	0	—	6.44	2.39	↑	9.30	1.17	↑
		p-value	<0.01			<0.01			<0.01			<0.01			0.97			<0.01			<0.01		
		Cliff's *d*	0.92 (L)			0.52 (L)			0.61 (L)			0.44 (M)			0.01 (N)			0.51 (L)			0.65 (L)		
Overall	Blob	NO	0.25	0		0.25	0		0.07	0		0.06	0		0.09	0		0.02	0		0.02	0	
		YES	73.76	29.14	↑	849.90	9.57	↑	16.26	3.30	↑	20.17	3.04	↑	1.15	0.20	↑	6.81	1.12	↑	2.15	0.08	↑
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			<0.01		
		Cliff's *d*	0.87 (L)			0.52 (L)			0.74 (L)			0.67 (L)			0.32 (S)			0.75 (L)			0.42 (M)		
	CDSP	NO	0.26	0		0.43	0		0.07	0		0.06	0		0.11	0		0.03	0		0.02	0	
		YES	9.36	2.10	↑	290.50	0	—	1.39	0	↑	0.57	0	↑	0.44	0	↑	0.94	0	↑	3.42	1.00	↑
		p-value	<0.01			0.3			0.04			0.02			<0.01			0.01			<0.01		
		Cliff's *d*	0.61 (L)			0.05 (N)			0.05 (N)			0.05 (N)			0.17 (S)			0.06 (N)			0.87 (L)		
	CC	NO	0.21	0		0.34	0		0.06	0		0.04	0		0.10	0		0.02	0		0.03	0	
		YES	63.00	12.60	↑	1573.00	46.81	↑	19.36	3.81	↑	15.68	1.93	↑	1.25	0.18	↑	9.29	1.40	↑	1.88	0.01	↑
		p-value	<0.01			<0.01			<0.01			<0.01			<0.01			<0.01			<0.01		
		Cliff's *d*	0.79 (L)			0.69 (L)			0.82 (L)			0.61 (L)			0.30 (S)			0.81 (L)			0.39 (M)		
	FD	NO	0.29	0		0.73	0		0.08	0		0.07	0		0.12	0		0.03	0		0.02	0	
		YES	-14.09	-4.00	↓	-5.59	-0.50	↓	-5.67	-1.37	↓	-4.50	-0.54	↓	-0.19	0	—	-2.60	-0.57	↓	-0.40	0	↓
		p-value	<0.01			<0.01			<0.01			<0.01			0.75			<0.01			<0.01		
		Cliff's *d*	-0.71 (L)			-0.51 (L)			-0.67 (L)			-0.49 (L)			0.01 (N)			-0.69 (L)			-0.22 (S)		
	SC	NO	0.17	0		1.02	0		0.04	0		-0.02	0		0.15	0		0.01	0		0.03	0	
		YES	134.00	36.29	↑	597.0	100.0	↑	15.09	6.34	↑	9.36	1.00	↑	1.27	0	—	5.84	3.00	↑	7.80	0.57	↑
		p-value	<0.01			<0.01			<0.01			<0.01			0.49			<0.01			<0.01		
		Cliff's *d*	0.93 (L)			0.52 (L)			0.66 (L)			0.52 (L)			0.06 (N)			0.59 (L)			0.67 (L)		

ANALYSIS OF THE RESULTS – RQ1

- Overall, from the analysis of Table V we can conclude that:

- (i) LOC characterizes the introduction of all the smells;
- (ii) LCOM, WMC, RFC and NOM characterize all the smells but Class Data Should be Private;
- (iii) CBO does not characterize the introduction of any smell; and
- (iv) the only metrics characterizing the introduction of Class Data Should be Private are LOC and NOA.

ANALYSIS OF THE RESULTS – RQ1

- Most of the smell instances are introduced when files are created. However, there are also cases, especially for Blob and Complex Class, where the smells manifest them selves after several changes performed on the file.

ANALYSIS OF THE RESULTS – RQ2

- RQ2 - Why are code smells introduced?

TABLE VII

RQ_2: COMMIT-GOAL TAGS TO SMELL-INTRODUCING COMMITS. BF: BUG FIXING; E: ENHANCEMENT; NF: NEW FEATURE; R: REFACTORING

Smell	Android				Apache				Eclipse				Overall			
	BF	E	NF	R	BF	E	NF	R	BF	E	NF	R	BF	E	NF	R
Blob	15	59	23	3	5	83	10	2	19	55	19	7	14	65	17	4
CDSP	11	52	30	7	6	63	30	1	14	64	18	4	10	60	26	4
CC	0	44	56	0	3	89	8	0	17	52	24	7	13	66	16	5
FD	8	48	39	5	16	67	14	3	18	52	24	6	16	60	20	4
SC	0	0	100	0	0	81	4	15	8	61	22	9	6	66	17	11

ANALYSIS OF THE RESULTS – RQ2

- RQ 2 - Why are code smells introduced?

TABLE VIII

RQ2: PROJECT-STATUS TAGS TO SMELL-INTRODUCING COMMITS

Ecosystem	Smell	Working on Release				Project Startup			
		One Day	One Week	One Month	More	One Week	One Month	One Year	More
Android	Blob	7	54	35	4	6	3	35	56
	CDSP	14	20	62	4	7	17	33	43
	CC	0	6	94	0	0	12	65	23
	FD	1	29	59	11	0	4	71	25
	SC	0	0	100	0	0	0	0	100
Apache	Blob	19	37	43	1	3	7	54	36
	CDSP	10	41	46	3	3	8	45	44
	CC	12	30	57	1	2	14	46	38
	FD	5	14	74	7	3	8	43	46
	SC	21	18	58	3	3	7	15	75
Eclipse	Blob	19	37	43	1	3	20	32	45
	CDSP	10	41	46	3	6	12	39	43
	CC	12	30	57	1	2	12	42	44
	FD	5	14	73	8	2	5	35	58
	SC	21	18	58	3	1	5	19	75
Overall	Blob	15	33	50	2	5	14	38	43
	CDSP	10	29	58	3	6	12	39	43
	CC	18	28	53	1	4	13	42	41
	FD	7	22	66	5	3	7	42	48
	SC	16	20	58	6	2	6	17	75

ANALYSIS OF THE RESULTS – RQ2

- RQ2 - Why are code smells introduced?

TABLE IX

RQ2: DEVELOPER-STATUS TAGS TO SMELL-INTRODUCING COMMITS

Ecosystem	Smell	Workload			Ownership		Newcomer	
		High	Medium	Low	True	False	True	False
Android	Blob	44	55	1	73	27	4	96
	CDSP	79	10	11	81	19	11	89
	CC	53	47	0	100	0	6	94
	FD	68	29	3	100	0	8	92
	SC	100	0	0	100	0	100	0
Apache	Blob	67	31	2	64	36	7	93
	CDSP	68	26	6	53	47	14	86
	CC	80	20	0	40	60	6	94
	FD	61	36	3	71	29	7	93
	SC	79	21	0	100	0	40	60
Eclipse	Blob	62	32	6	65	35	1	99
	CDSP	62	35	3	44	56	9	91
	CC	66	30	4	47	53	9	91
	FD	65	30	5	58	42	11	89
	SC	43	32	25	79	21	3	97
Overall	Blob	60	36	4	67	33	3	97
	CDSP	68	25	7	56	44	11	89
	CC	69	28	3	45	55	3	97
	FD	63	33	4	67	33	8	92
	SC	55	28	17	79	21	15	85

ANALYSIS OF THE RESULTS – RQ2

- Why are code smells introduced?

- Smells are generally introduced by developers when enhancing existing features or implementing new ones.
- As expected, smells are generally introduced in the last month before issuing a deadline, while there is a considerable number of instances introduced in the first year from the project startup.
- Finally, developers that introduce smells are generally the owners of the file and they are more prone to introducing smells when they have higher workloads.

THREATS TO VALIDITY

- The results can be affected by the presence of false positives and false negatives. DECOR a precision above 60% and a recall of 100%.
- Workload tag measures the developers' activity within a single project (busy on other projects or different other activities).

CONCLUSION AND LESSONS LEARNED

- Lesson 1. Most of times code artifacts are affected by bad smells since their creation.
- Lesson 2. Code artifacts becoming smelly as consequence of maintenance and evolution activities are characterized by peculiar metrics trends, different from those of clean artifacts.

CONCLUSION AND LESSONS LEARNED

- Lesson 3. While implementing new features and enhancing existing ones are, as expected, the main activities during which developers tend to introduce smells, we found almost 400 cases in which refactoring operations introduced smells.
- Lesson 4. Newcomers are not necessary responsible for introducing bad smalls, while developers with high workloads and release pressure are more prone to introducing smell instances.

REFERENCES

- [1] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk. (2014) Replication package. [Online]. Available: http://tinyurl.com/ormdqpy.

END