Are all Code Smells Harmful? A Study of God Classes and Brain Classes in the Evolution of three Open Source Systems

Steffen M. Olbrich
Fraunhofer IESE
Kaiserslautern, Germany
steffen.olbrich@iese.fraunhofer.de

Daniela S. Cruzes
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
dcruzes@idi.ntnu.no

Dag I.K. Sjøberg
Department of Informatics
University of Oslo
Oslo, Norway
dagsj@ifi.uio.no

***Abstract*—Code smells are particular patterns in object-oriented systems that are perceived to lead to difficulties in the maintenance of such systems. It is held that to improve maintainability, code smells should be eliminated by refactoring. It is claimed that classes that are involved in certain code smells are liable to be changed more frequently and have more defects than other classes in the code. We investigated the extent to which this claim is true for God Classes and Brain Classes, with and without normalizing the effects with respect to the class size. We analyzed historical data from 7 to 10 years of the development of three open-source software systems. The results show that God and Brain Classes were changed more frequently and contained more defects than other kinds of class. However, when we normalized the measured effects with respect to size, then God and Brain Classes were *less* subject to change and had *fewer* defects than other classes. Hence, under the assumption that God and Brain Classes contain on average as much functionality per line of code as other classes, the presence of God and Brain Classes is not necessarily harmful; in fact, such classes may be an efficient way of organizing code.**

Keywords: Code smells, detection strategies, change frequency, defects, software evolution, open source

I. INTRODUCTION

The notion of *code smells*, which was introduced initially by Fowler and Beck [8], is now an established concept for referring to patterns, or aspects of design, in a software system that may cause problems for the further development and maintenance of the system [8], [9], [19], [24].

Empirical work on code smells has primarily investigated whether code with smells leads to changes being made more frequently and to the occurrence of more defects than code without smells. Khomh et al. [10] found that, overall, smelly classes in two open-source systems, Azureus and Eclipse, were changed more frequently than other classes, and that some specific smells, including one that they called Large Class, were correlated with more frequent changes. Lozano et al. [13] investigated the open-source system DnsJava and found that methods that had been cloned were changed more often than those that had not. A study by Olbrich et al. [23] of two open-source systems showed that classes with code smells (God Class and Shotgun Surgery) were changed more often than other classes. Furthermore, God Classes in particular were subject to *larger* changes than were other classes.

Regarding the effect of code smells on defects, Li and Shatnawi [11] found that the smells Shotgun Surgery, God Class, and God Methods were associated positively with the number of defects in three releases of Eclipse (3.0, 2.1, 2.0). A study by Rahman et al. [25] of the open-source systems Apache httpd, Nautilus, Evolution, and Gimp showed that most bugs have very little to do with clones. Deligiannis et al. conducted an experiment using students as subjects that showed that a system with a God Class led to more difficulties in maintenance tasks than did the same system without a God Class [5].

We wanted to investigate the effect of some of the smells in more depth. More precisely, we wanted to study the influence of God Classes and Brain Classes on the frequency of defects detected, and the frequency and size of changes made, over a span of the evolution of certain systems. Furthermore, God Classes and Brain Classes are related to the smell Large Class [8], because they are typically large classes. An important issue regarding the results of the studies described above is whether the God and Brain Classes are subject to relatively frequent changes and defects due to their size alone or whether the phenomenon is due to other structural characteristics. If it were found that size alone is the cause, it would be better to distribute the functionality in the system by splitting God and Brain Classes into several smaller classes. This issue is of particular interest because related work by Zhou and Leung showed, in an analysis of two releases of Eclipse (2.1, 2.0), that there was a confounding effect of class size on the associations between certain object-oriented metrics and proneness to change [28]. Thus, the issue arises as to whether this is also the case for code smells, which are often identified on the basis of such metrics.

We collected historical data from the three open-source systems Log4j, Apache Lucene, and Apache Xerces over, respectively, 7, 9, and 10 years of development. The data included detailed information about the changes that were performed and defects that were identified. We developed a tool to identify the occurrence of code smells over time, as the systems evolved. We then analyzed the differences between God/Brain Classes and other classes with respect to the rate at which defects occurred, and the size and frequency of changes made. To investigate the effect of size, we performed the analysis both without and with normalizing with respect to the size of the classes, measured in lines of code [7], [12].

The remainder of this paper is structured as follows. Section II describes God Classes and Brain Classes. Section III describes the strategies for detection as they were applied in the study. Section IV motivates and presents the hypotheses that were tested. Section V describes the experimental setup. Section VI presents the results, which are discussed in Section VII. Section VIII presents conclusions that were drawn from the results and states what the results of the study contribute to.

II. God and Brain Classes

Initially, Fowler et al. defined a set of 22 code smells that have different characteristics and may influence software systems in different ways [8]. Depending on the particular code smell, the system component that is measured can either be a class, method or subsystem. Other researchers later modified and extended the initial set of smells [6], [9], [16], [17], [18], [19]. For example, Marinescu divided the Large Class smell into the God Class and Brain Class smells [17], which are described below.

A. God Class

The term 'God Class' "refers to those classes that tend to centralize the intelligence of the system. An instance of a god-class performs most of the work, delegating only minor details to a set of trivial classes and using the data from other classes" [19]. A God Class features a high complexity, low inner-class cohesion, and heavy access to data of foreign classes. It violates the object-oriented design principle that each class should only have one responsibility [19]. God Classes tend to be very large, which in turn may make it more difficult to understand the system [8]. Furthermore, it is thought that, due to the omnipotent role that it assumes within a system, a God Class will be changed much more than other classes when maintenance is being performed. This may increase the risk of defects occurring in such classes. If more defects occur, changes will need to be made more frequently and the size of the changes that are required are more likely to be large. Thus, the maintenance effort increases [8], [19]. Nevertheless, not every God Class is assumed to hamper the evolution of a system. It has been shown that in some cases, a God Class might be the best solution (e.g., parser implementations). It has been reported that classes that have the structural characteristics of a God Class, but that do hardly undergo modification and reside in a stable part of the system, do not cause problems [26], [27].

B. Brain Class

Similar to God Classes, Brain Classes tend to be complex and centralize the functionality of the system [19]. It is therefore assumed that they are difficult to understand and maintain. However, contrary to God Classes, Brain Classes do not use much data from foreign classes and are slightly more cohesive. Furthermore, as for God Classes, the probability that defects will occur in Brain Classes is higher than for other classes [8], [19], but if a Brain Class is created intentionally and remains unmodified, the system may not experience any problems.

III. Detection Strategies

The detection of code smells in software systems can either be performed manually (e.g., by code reviews) [8] or by applying automated detection heuristics (e.g., on the basis of code metrics) [6], [9], [16], [17], [18], [19]. Even though manual detection by code reviews is thought to be the most reliable way of identifying code smells [8], it has certain disadvantages (e.g., that it is time-consuming and nonrepeatable), which Marinescu and Mäntylä have identified in a series of works [14], [15], [16]. The performance of different automated detection heuristics have been compared with manual approaches in various studies [6], [9], [13], [14], [15], [17], [21]. The results show that automated detection is a worthwhile alternative to manual detection. Automated detection strategies have been found to be as precise as, and have the same recall, as manual detection. In addition, they scale much better.

In the study reported herein, we used automatic heuristics to detect the smells, because the systems that we analyzed were too large for manual detection: they had up to 400 classes on average in up to 92 investigated revisions. Specifically, we used an implementation of Marinescu's *detection strategies* for detecting God and Brain Classes [19]. These detection strategies interpret a set of code metrics that are extracted from a specific system component (i.e., class or method) by using a set of threshold filter rules. Then, the results of these filters are combined and used to identify the more composite code smells [19]. Below, the strategies that were used to detect God Classes, Brain Classes and Brain Methods (used as part of the Brain Class detection) are presented.

A. God Class Detection Strategy

Equation 1 below shows the formula and states the thresholds for detecting a God Class (GC) as they are given in [19].

$$GC(C) = \begin{cases} 1, & ((WMC(C) \geq 47) \wedge (TCC(C) < 0.\overline{3}) \wedge \\ & (ATFD(C) > 5)) \\ 0, & else \end{cases} \quad (1)$$

Where:

C is the class being inspected.

Weighted Method Count (*WMC(C)*) is the sum of the cyclomatic complexity of all methods in *C* [3] [20].

Tight Class Cohesion (*TCC(C)*) is the relative number of directly connected methods in *C*. Two methods are directly connected if they access the same instance variables of *C* [2].

Access To Foreign Data (*ATFD(C)*) is the number of attributes of foreign classes accessed directly by class *C* or via accessor methods [17].

B. *Brain Class Detection Strategy*

Equation 2 shows the detection strategy used to identify a Brain Class (BC) [19]. The thresholds are based on Marinescu's work [19].

$$BC(C) = \begin{cases} 1, & GC(C) \wedge ((WMC(C) \geq 47) \wedge (TCC(C) < 0.5)) \wedge \\ & (((NBM(C) > 1) \wedge (LOC(C) \geq 197)) \vee \\ & ((NBM(C) \equiv 1) \wedge (LOC(C) \geq 2 \cdot 197) \wedge \\ & (WMC(C) \geq 2 \cdot 47))) \\ 0, & else \end{cases} \quad (2)$$

Where:

GC(C) is "true" if *C* is a God Class, otherwise "false" [19].

WMC(C) and *TCC(C)* are as described under God Class detection.

Number of Brain Methods (*NBM(C)*) is the number of methods identified as Brain Methods in *C*; see definition below.

Lines of Code (*LOC(C)*), is the number of lines of code in *C* (in our case, the sum of the *LOC* of all methods in *C*, including comments and blank lines).

C. *Brain Method Detection Strategy*

Equation 3 shows the formula and states the thresholds used for identifying a Brain Method [19].

$$BM(M) = \begin{cases} 1, & ((LOC(M) > 65) \wedge \\ & (CYCLO(M) / LOC(M) \geq 0.24) \wedge \\ & (MAXNESTING(M) \geq 5) \wedge (NOAV(M) > 8)) \\ 0, & else \end{cases} \quad (3)$$

Where:

M is the inspected method.

Lines of Code (*LOC(M)*), is the number of lines of code in *M* (including comments and blank lines).

Cyclomatic complexity (*CYCLO(C)*) of method *M* [3].

Maximum Nesting Level (*MAXNESTING(M)*) of control structures within the method [19].

Number of Accessed Variables (*NOAV(M)*) is the total number of variables accessed directly by method *M* [19].

IV. Research Questions

The first goal of the study was to investigate whether God and Brain Classes are changed more frequently, have a larger change size, and have more defects than have other classes in a system. The corresponding hypotheses for this investigation are formulated one-sided, because all former studies reported in the literature have shown worse values for God and Brain Classes than for other classes.

The second goal was to investigate whether the result would be different if the God and Brain Classes were normalized with respect to size, that is, whether the extent of change and defects is greater per line of code in God and Brain Classes than in other classes. Given that no other work has been reported that studied the effect of God and Brain Classes per LOC, we formulated the corresponding hypotheses as two-sided.

We divided the analysis of the effects of God and Brain Classes into (1) the change frequency (*CF* and *CF/LOC*), (2) the change size (*CS* and *CS/LOC*, on the basis of code churn [22]), and (3) weighted defect rate (*WDR* and *WDR/LOC*) of God and Brain Classes and other classes. Change Frequency *(CF)* was defined as how often a measured class gets changed in the repository within a certain time frame (*n* revisions). Change Size *(CS)* was defined as the number of line changes (added, modified or deleted) made in a class per revision within a certain time frame (*n* revisions). We used a weighted measure of the defect rate that was based on the severity of each defect, accordingly to the Orthogonal Defect Classification *(ODC)* [3]. The weighted defect rate *(WDR)* is the number of weighted defects that occur in a measured class within a certain time frame (*n* revisions).

Our hypotheses were as follows:

A. *Change Frequency (CF)*

$H1_0$: *The CF of God and Brain Classes is equal to or lower than the CF of other classes.*

$H1_A$: *The CF of God and Brain Classes is higher than the CF of other classes.*

$H2_0$: *The CF per LOC of God and Brain Classes is equal to the CF of other classes.*

$H2_A$: *The CF per LOC of God and Brain Classes is different from the CF per LOC of other classes.*

B. *Change Size (CS)*

$H3_0$: *The CS of God and Brain Classes is equal to or lower than the CS of other classes.*

$H3_A$: *The CS of God and Brain Classes is higher than the CS of other classes.*

$H4_0$: *The CS per LOC of God and Brain Classes is equal to the CS of other classes.*

$H4_A$: *The CS per LOC of God and Brain Classes is different from the CS per LOC of other classes.*

C. *Weighted Defect Rate (WDR)*

$H5_0$: *The WDR of God and Brain Classes is equal to or lower than the WDR of other classes.*

$H5_A$: *The WDR of God and Brain Classes is higher than the WDR of other classes.*

$H6_0$: *The WDR per LOC of God and Brain Classes is equal to the WDR of other classes.*

$H6_A$: *The WDR per LOC of God and Brain Classes is different from the WDR per LOC of other classes.*

V. Experimental Setup

A. Instrumentation

The research questions stated above require an analysis of the evolution of software systems. Therefore, we developed a tool called EvolutionAnalyzer (EvAn). This tool gathers and analyzes the evolution of Java applications that reside in Subversion [1] source code repositories. Figure 1 shows a schematic overview of EvAn. Figure 2 shows a schematic example view of the generated data model, which we call the system-evolution-meta-model.

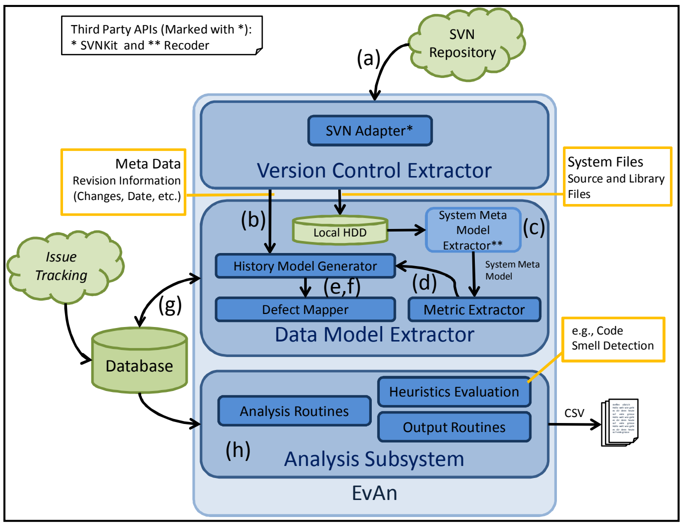

Figure 1. Schematic overview of the EvAn.

The functionalities of EvAn can be summarized as follows: EvAn can (a) gather data from source code repositories (using the SVNKit[2] API), (b) extract the meta-information of each revision (i.e., revision date, author, change data, etc.), (c) extract the meta-model of the system resident in the repository (using the Recoder[3] API), (d) enrich this system meta-model with the results for the code metrics that were extracted from the classes and methods that contain the code, (e) map the change and defect data (added manually; see the following subsection "Data Collection") to the corresponding elements in the system meta-model, (f) couple each element in a revision (i.e., source files, classes, methods) to its correspondent in the previous one, (g) store this data in a local database, and (h) perform further postprocessing and analysis.

In order to extract the system meta-model using Recoder, the system needs to be compilable and complete (e.g., it needs to contain all the libraries that will be used). To help solve the typical problem of missing libraries, EvAn facilitates the manual inclusion of missing libraries during the extraction process.

Each source file in the system meta-model is enriched with detailed information about the changes that were made. In addition, all source files, classes, and methods are coupled to their counterparts in the previous revision on the basis of their unique identifiers (i.e., source file path, class full-name, method name, and parameter list). Thereby, changes of the source file path and class name (and constructor names) are recognized and the coupling is performed accordingly.

The correctness of the different EvAn functionalities is assured by acceptance testing. The extraction of the system and change data, and the generation of the historical relations, were validated using sample repositories (i.e., the EvAn development and Apache Lucene repository) and WebSVN[4] as reference implementation. The extraction of the meta-model was validated using a set of sample systems as a basis (EvAn itself and some smaller test-case systems). The extraction of the metric was cross-validated by using Marinescu's code-smell detection and metric-extraction framework [5] as a reference implementation on the set of sample systems.

B. Defects Data

As our source for the data on defects, we used the project's issue-tracking systems, i.e., JIRA[6] for Xerces and Lucene, and Bugzilla[7] for Log4J. In contrast to the automated collection of most of the data that was needed for the analysis; the defect data was added to the data model manually and coupled manually to the respective revisions on the basis of the ID number of each defect. This ID number is also given in the commit message of the revisions in which the defect was handled. The defects were later coupled automatically to the particular source files that were modified in the associated revisions that dealt with those defects. In accordance with the issue-tracking template, we included issues of type *bug* that are marked as *fixed* with a corresponding progress status (i.e., *resolved* or *closed* and the additional Bugzilla status *verified*). Analogously to the severity levels in the Orthogonal Defect Classification [3], the defects were weighted with respect to the severity levels provided by JIRA and Bugzilla, as shown in Table I. The only difference between JIRA and Bugzilla with respect to the levels of severity of the defects is that Bugzilla has an extra severity level, named 'normal', which we handled as 'major' in accordance with the JIRA import rule[8].

TABLE I. Defect Weight after Severity Level

Defect Severity JIRA	*Defect Severity Bugzilla*	*Weight*
Blocker	Blocker	16
Critical	Critical	8
Major	Major/Normal	4
Minor	Minor	2
Trivial	Trivial	1

[1] Subversion <http://subversion.apache.org>
[2] SVNKit <http://svnkit.com>
[3] Recoder <http://recoder.sourceforge.net>

[4] WebSVN <http://websvn.tigris.org/>
[5] iPlasma <http://loose.upt.ro/iplasma/>
[6] JIRA <http://www.atlassian.com/software/jira>
[7] Bugzilla <http://www.bugzilla.org>
[8] Importing Bugzilla Data <http://www.atlassian.com/software/jira/docs/v3.13.2/bugzilla_import.htm>

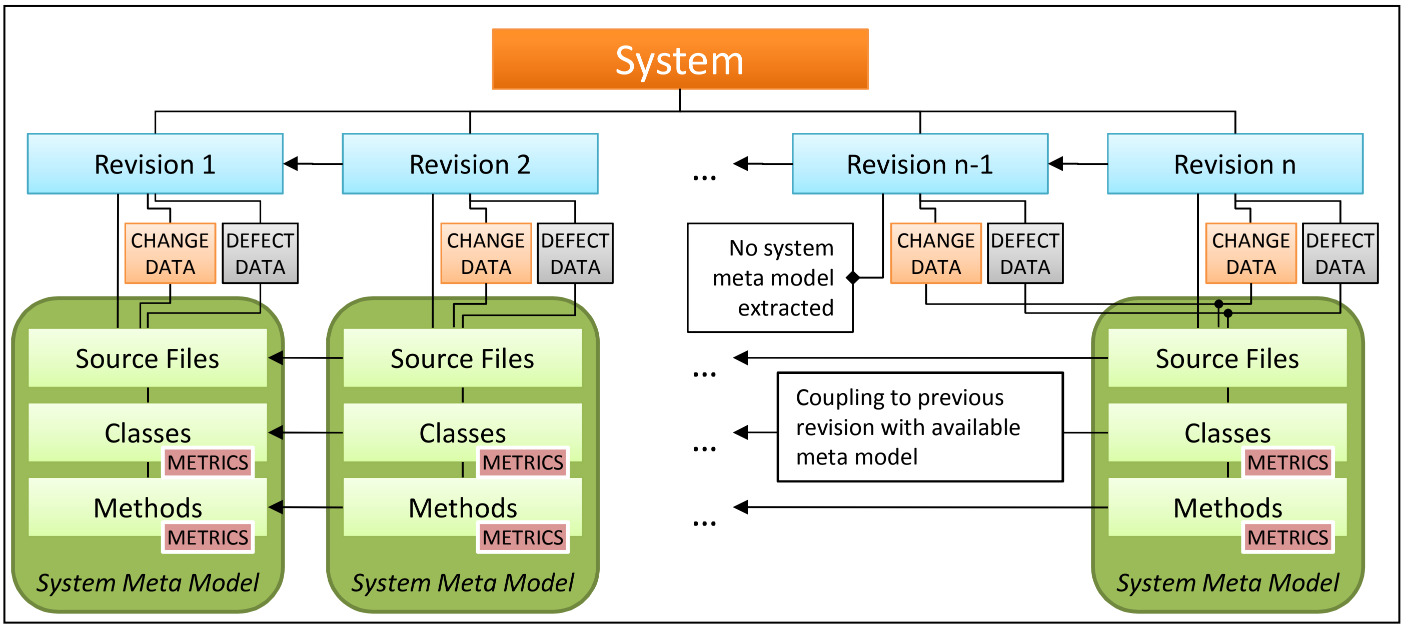

Figure 2. System-Evolution-Meta-Model.

C. Systems

We investigated the change and defect history over multiple years of the development of three popular open source systems: Apache Lucene, Apache Xerces 2 Java and Log4j. Lucene is a high-performance, full-featured text-search engine library written in Java that is suitable for applications that require full text search, especially cross-platform.[9] Apache Xerces 2 Java is "a library for parsing, validating and manipulating XML documents"[10], and Log4j is a "Java-based logging utility that is used primarily as a debugging tool."[11] The characteristics of the selected systems are summarized in Table II.

TABLE II. OVERVIEW OF ANALYZED SYSTEMS

Characteristics/Systems	*Lucene*	*Xerces*	*Log4j*
Bug-Tracking:	JIRA	JIRA	BugZilla
#Revisions	4521	5288	2642
Analyzed Timeframe:	2001-2010	1999-2009	2000-2007
#Revisions with extracted meta-model	67	92	41
#Classes (last Version)	651 (Revision 920499, 03/08/10)	712 (Revision 888921, 09/12/09)	337 (Revision 500752, 01/28/07)
#Defects (tracked)	695	175	342
#Authors	38	32	16

The three systems are well-known in the open-source and software-engineering-research communities, and, as such, are frequently chosen as subjects of analysis for various empirical studies. These particular systems were also selected because they provide (1) sufficient data about the evolution of the system (measured by the number of revisions and years), and (2) a system size that is sufficient to obtain adequate sample sizes (measured by the number of classes).

[9] Apache Lucene <http://lucene.apache.org>
[10] Apache Xerces <http://xerces.apache.org>
[11] Log4j <http://logging.apache.org/log4j>

D. Procedures for Data Analysis

As experimental data, we used the complete change and defect data of the described evolution span of the three systems. Due to the fact that extracting the system meta-model of each single revision would be too time-consuming and that the data collected would yield too little extra value, this extraction was only performed for each 50th revision of the system. As an exception to this rule, both the first and second revisions that were committed were extracted. The reason for that is that a standard initial SVN revision is only used for the creation of the repository and does not contain any files. As shown in Figure 3, each source file that is present in revision *n* is enriched with an aggregation of the changes that were made and defects that occurred between its current revision and revision *n-1* (the latest, previous revision with the extracted system meta-model). For the analysis, this information on source file level is adopted by the class that is represented in a source file whereas cases of multiple classes per source file are excluded to assure a correct mapping of the effects.

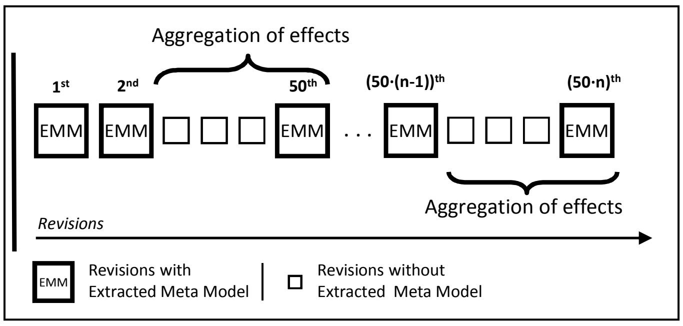

Figure 3. Aggregation of effect data as performed by EvAn.

As described above, the extraction of the meta-model requires a compilable version of the system. This constraint is not always met for systems in development. In response, if a revision was not compilable due to missing libraries, we used the EvAn functionalities to add missing libraries manually. If the revision did not compile for other reasons, we skipped it and looked for later revisions. Therefore, the system meta-model could not always be extracted for each 50th revision. These exceptions to the general procedure of extracting a meta-model for each 50th revision were recognized and handled by EvAn and the applied measures. The number of system meta-models that were extracted for each system is presented in Table II, row "#Revisions with extracted meta-model".

To test our hypotheses, the change frequency (CF and CF/LOC), change size (CS and CS/LOC), and defect rate (WDR and WDR/LOC) were calculated for God, Brain, and other classes in each revision for which a meta-model was extracted. The value for each effect factor using normalization (i.e., the extent of change and number of defects per LOC), is given by equations 7, 8, and 9. The values for non-normalized effect factors were given by similar equations, which differed only in excluding the LOC.

$$CF(C_t)/LOC(C_t) = \frac{(NC(C_t)/PDIST(C_t)*100)}{LOC(C_t)} \quad (7)$$

$$CS(C_t)/LOC(C_t) = \frac{(CSIZE(C_t)/PDIST(C_t)*100)}{LOC(C_t)} \quad (8)$$

$$WDR(C_t)/LOC(C_t) = \frac{(SUMWD(C_t)/PDIST(C_t)*100)}{LOC(C_t)} \quad (9)$$

Where:

NC(C_t) returns the number of changes that were made in class *C* between revision *n* and revision *n-1* (at time *t*),

CSIZE(C_t) returns the sum of the code churns on class C between revision *n* and revision *n-1*,

SUMWD(C_t) returns the sum of the (weighted) number of defects that occurred in class *C* between revision *n* and revision *n-1*,

PDIST(C_t) returns the distance in number of revisions between class *C* in revision *n* and its counterpart in revision *n-1*,

LOC(C_t) returns the number of lines of code in class *C* in revision *n*. In our case, this number is the sum of lines of code of all methods within the class (including comments and blank lines), and

all values are multiplied by *100* to avoid numbers that are too small and to avoid problems with rounding numbers when calculating (i.e., machine ε issues).

The analysis adheres to the following constraints:

We only analyzed the effects on classes that had a stable detection status between two analyzed revisions. That is, the class in the current revision and its counterpart in the previous revision were either detected as a God/Brain Class or neither. This was done so that we could add the measured effects clearly to (a) the sample of classes with smell and (b) the sample of classes without smell [23].

Interfaces were excluded in both samples. Given the definition of the detection strategies, an Interface can never contain a God or Brain Class.

Abstract and Enum classes were included. Abstract and Enum classes can be God or Brain Classes.

EvAn produced the necessary output data as CSV files[12]. The data for the impact analysis of God/Brain Classes contains the following information:

The revision number at time *t* of the measured class.

An identifier of the class at time *t* (Class name and source file name).

The God/Brain Class detection result at time *t* (in accordance with the constraints identical to the detection status at time *t-1*).

The measured impact data:

- *CF(C_t)/LOC(C_t) and CF(C_t)*
- *CS(C_t)/LOC(C_t)* and *CS(C_t)*
- *WDR(C_t)/LOC(C_t) and WDR(C_t).*

We also performed a descriptive analysis of the evolution of the density of God and Brain Classes within the systems. The density at time *t* was defined as the proportion of God and Brain Classes relative to the total number of classes in the system at time *t*. The data that was acquired contains the following information:

The revision number at time *t* of the measured system version.

The revision date.

The total number of classes at time *t*.

The number of God Classes at time *t*.

The number of Brain Classes at time *t*.

VI. Results

A. Descriptive Information about the Systems

We now report the descriptive statistics for the three systems. We monitored the evolution of the system size (#classes) and density of God and Brain Classes. Figures 5, 6, and 7 show the evolution of the three systems within the measured timeframe (the common legend is shown in Figure 4). In the graphs, the x-axis represents the time line of the measured evolution span, the y-axis on the left side represents the density in percentage relative to the system size, and the y-axis on the right represents the system size measured by the number of classes.

Xerces has the most God Classes. In the most recent years, almost 8% of the classes were God Classes, and the number of such classes has increased slowly but steadily since the first years. Almost 4% of the classes in Lucene are God Classes, but

[12] The XLS files with the collected data can be found on our web space: http://www.idi.ntnu.no/~dcruzes/ICSM2010/.

the number of these has decreased slowly in recent years. Almost 3% of the classes in Log4j are God Classes, but in the last few years this proportion has fallen to 1.2%.

Brain Classes are less frequent in the systems. Lucene had no Brain Classes to begin with and Xerces had only one. At any one time during the entire sampling period, at most 0.5% of the Lucene classes were Brain Classes and at most 1.9% of the Xerces classes. Xerces contains eight classes that are both a God and Brain Class; Lucerne contains one; and Log4j does not contain any.

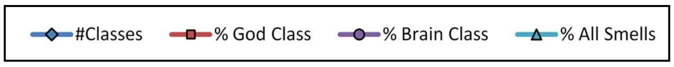

Figure 4. Legend for Figures 5-7.

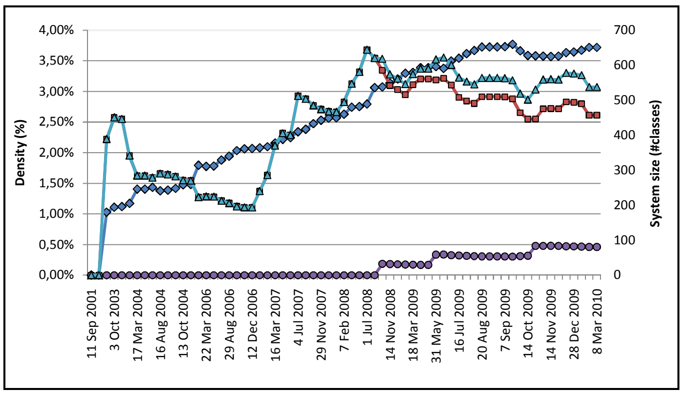

Figure 5. Density of God and Brain Classes in Lucene.

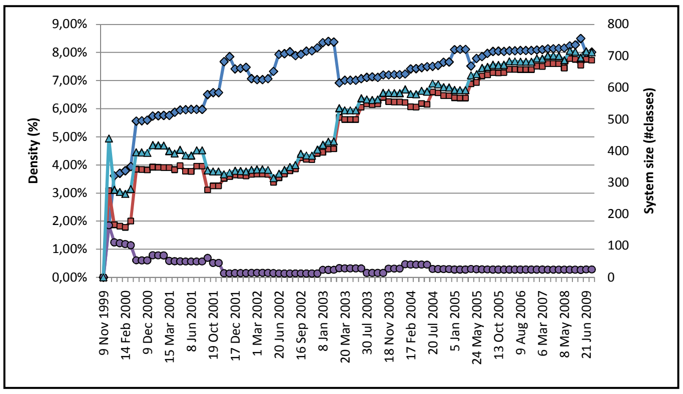

Figure 6. Density of God and Brain Classes in Xerces.

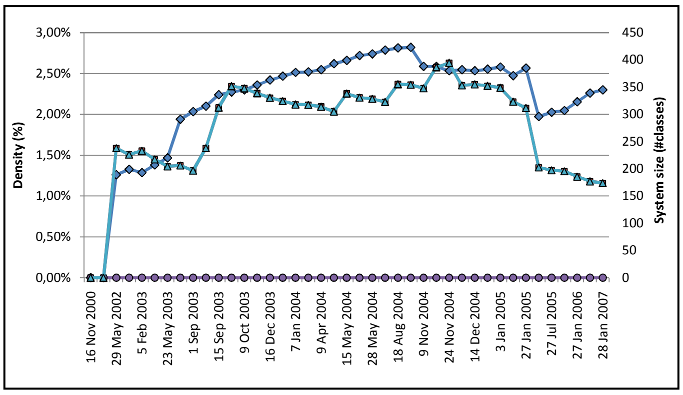

Figure 7. Density of God and Brain Classes in Log4j.

Within the systems, there is a large, positive correlation between the system size (#classes) and the number of God Classes (0.94 for Lucene, 0.77 for Xerces, and 0.89 for Log4j). For Brain Classes, there is the same positive correlation in Lucene (0.79), but a negative correlation in Xerces (-0.65). Log4j contains no Brain Classes at all.

The definition of God and Brain Classes indicates that they are larger than other classes and thus cluster or centralize functionality [8] [19]. Table III and IV show that God and Brain Classes in the systems under consideration are about 6 to 10 times larger than other classes.

TABLE III. CLASS SIZE COMPARISON GOD CLASS

		Lucene		*Xerces*		*Log4j*	
		GC	¬GC	GC	¬GC	GC	¬GC
	n :	714	26214	2526	33690	256	10166
LOC	$\overline{X}$:	**881.3908**	93.6075	**768.0867**	111.2898	**677.4453**	62.7319
	s :	808.2458	144.6259	593.6356	212.7042	478.0222	82.0332

TABLE IV. CLASS SIZE COMPARISON BRAIN CLASS

		Lucene		*Xerces*	
		BC	¬BC	BC	¬BC
	n :	57	26225	177	33695
LOC	$\overline{X}$:	**939.0**	93.7139	**655.4972**	111.3175
	s :	359.5675	144.9027	261.8377	212.7329

B. *Hypothesis Tests*

Table V summarizes the results of the tests of the alternative hypotheses. We used one-sided Mann–Whitney U-tests for the not-normalized measures and two-sided ones for the normalized measures. We used a standard non-parametric test because the collected data did not always show a normal distribution.

TABLE V. RESULTS OF TESTING THE ALTERNATIVE HYPOTHESES

Factor	*H#*	*God Class*	*Brain Class*
Not-normalized			
CF	H1	Supported	Supported
CS	H3	Supported	Not supported for Lucene, Supported for Xerces
WDR	H5	Supported for Lucene and Xerces, Not for Log4j	Not supported
Normalized			
CF/LOC	H2	Supported	Not supported for Lucene, Supported for Xerces
CS/LOC	H4	Supported	Not Supported for Lucene, Supported for Xerces
WDR/LOC	H6	Supported for Lucene and Xerces, Not for Log4j	Not supported

Tables VI–IX show the results of the hypothesis tests. In all the tables, a grey-shaded *p-value* row indicates that the null hypothesis is rejected and hence that the alternative hypothesis (higher value in God and Brain Classes) is supported. A *p-value* row shaded with diagonal strokes indicates that the null hypothesis is not rejected. The higher mean value of the two samples is emphasized in bold letters. For all the tests, an alpha value of 0.05 was used. In the result tables presented below, *n* represents the sample sizes, $\overline{X}$ the mean, *s* the standard deviation, and *U* the test's calculated u-value for the samples.

TABLE VI. H1, H3, H5: NON-NORMALIZED GOD CLASS EFFECTS

		Lucene		*Xerces*		*Log4j*	
		GC	¬GC	GC	¬GC	GC	¬GC
	n :	714	26214	2526	33690	256	10166
CF	$\overline{X}$:	**2.2669**	0.5341	**1.5599**	0.3651	**2.6055**	0.4611
	s :	3.4126	1.1815	3.2168	1.0333	4.6876	1.3303
	p:	**< 0.0001**		**< 0.0001**		**< 0.0001**	
	U:	6156503		31793884.5		923024	
CS	$\overline{X}$:	**98.2568**	8.8320	**66.0564**	8.2975	**188.9023**	14.5583
	s :	362.9928	43.7211	359.4753	61.9957	708.0995	115.4257
	p:	**< 0.0001**		**< 0.0001**		**< 0.0001**	
	U:	6109831		31876982		926404	
WDR	$\overline{X}$:	**2.0436**	0.2824	**0.2536**	0.0265	**0.2031**	0.1078
	s :	5.5677	1.8393	1.8908	0.8062	2.1305	1.1588
	p:	**< 0.0001**		**0.0179**		**0.4661**	
	U:	7806803		41493800.5		1297388	

TABLE VII. H2, H4, H6: NORMALIZED GOD CLASS EFFECTS

		Lucene		*Xerces*		*Log4j*	
		GC	¬GC	GC	¬GC	GC	¬GC
	n :	714	26214	2526	33690	256	10166
CF/LOC	$\overline{X}$:	0.0032	**0.0163**	0.0022	**0.0148**	0.0040	**0.0205**
	s :	0.0051	0.0823	0.0039	0.0922	0.0070	0.1379
	p:	**< 0.0001**		**< 0.0001**		**< 0.0001**	
	U:	7399961		34508326.5		1028154	
CS/LOC	$\overline{X}$:	0.1071	**0.2148**	0.0679	**0.2420**	0.2483	**0.5732**
	s :	0.3336	2.2655	0.2545	2.8626	0.7868	10.324
	p:	**< 0.0001**		**< 0.0001**		**< 0.0001**	
	U:	6878362.5		33602003		985512.5	
WDR/LOC	$\overline{X}$:	0.0027	**0.0084**	0.0004	**0.0004**[a]	0.0007	**0.0037**
	s :	0.0090	0.1584	0.0034	0.0172	0.0076	0.0540
	p:	**< 0.0001**		**0.0362**		**0,9363**	
	U:	7876078		41497052.5		1297621	

a. WDR mean for ¬GC is 0.00042 and for GC 0.00038

TABLE VIII. H1, H3,H5: NON-NORMALIZED BRAIN CLASS EFFECTS

		Lucene		*Xerces*	
		BC	¬BC	BC	¬BC
	n :	57	26225	177	33695
CF	$\overline{X}$:	**0.8772**	0.5349	**1.0197**	0.3655
	s :	1.6044	1.1821	2.2487	1.0335
	p:	**0.0364**		**< 0.0001**	
	U:	676795		2542405	
CS	$\overline{X}$:	**9.7544**	8.9902	**17.5324**	8.3916
	s :	33.3661	45.2429	51.2649	62.6868
	p:	**0.0538**		**< 0.0001**	
	U:	683809		2551628	
WDR	$\overline{X}$:	0.1404	**0.2823**	**0.2961**	0.0265
	s :	1.0596	1.8389	2.0165	0.8061
	p:	**0.6069**		**0.2398**	
	U:	761942		2890775	

TABLE IX. H2, H4, H6: NORMALIZED BRAIN CLASS EFFECTS

		Lucene		*Xerces*	
		BC	¬BC	BC	¬BC
	n :	57	26225	177	33695
CF/LOC	$\overline{X}$:	0.0013	**0.0163**	0.0015	**0.0147**
	s :	0.0028	0.0823	0.0029	0.0922
	p:	**0.6201**		**0.0021**	
	U:	727992.5		2669926	
CS/LOC	$\overline{X}$:	0.0144	**0.2198**	0.0256	**0.2424**
	s :	0.0448	2.3925	0.0708	2.8627
	p:	**0.4012**		**0.0009**	
	U:	714231		2643592	
WDR/LOC	$\overline{X}$:	0.0001	**0.0084**	0.0004	**0.0004**[a]
	s :	0.0009	0.1584	0.0026	0.0172
	p:	**0.7777**		**0,4804**	
	U:	762538		2890953	

a. WDR/LOC mean for ¬BC is 0.00042 and for BC 0.00039

H1$_A$: *The CF of God and Brain Classes is higher than the CF of other classes.*

The hypothesis that God and Brain Classes undergo changes more frequently than other classes is supported for all three systems (Tables VI and VIII). On average, God Classes are changed 4-6 times and Brain Classes 1.5-3 times more often than other classes.

H2$_A$: *The CF per LOC of God and Brain Classes is different from the CF per LOC of other classes.*

The hypothesis is supported for God Classes in all the systems. For Brain Classes, it is supported for Xerces but not for Lucene. Tables VII and IX show that the means of CF/LOC for God and Brain Classes in all three systems are noticeably (5 to 10 times) *lower* than for other classes. This shows that each line of code in a God or Brain Class, is, on average in the analyzed systems, changed *less* often than in other classes.

H3$_A$: *The CS of God and Brain Classes is higher than the CS of other classes.*

The hypothesis is supported for God Classes in all the systems and for Brain Classes in Xerces. Tables VI and VIII show that the results for change size are similar to those for change frequency in the three systems. Although the hypothesis was not supported for Brain Classes in Lucene, the average sizes of changes for all the systems are higher for God and Brain Classes than for other classes.

H4$_A$: *The CS of God per LOC and Brain Classes is different from the CS per LOC of other classes.*

Tables VII and IX show that the size of changes performed per line of code in the investigated systems is, on average, *smaller* for God and Brain Classes than for other classes. For example, in Lucene, the change size for non-God Classes is on average 15 times higher than for God Classes. The hypothesis is supported in all three systems for God Classes but only in Xerces for Brain Classes.

$H5_A$: *The WDR of God and Brain Classes is higher than the WDR of other classes.*

Tables VI and VIII show that, on average, God and Brain Classes have a higher weighted defect rate than other classes (up to 10 times greater for Brain Classes in Xerces). Only the Brain Classes in Lucene have a lower weighted defect rate than other classes. However, the results are only significant (and hence, the hypothesis is only supported) for God Classes in Lucene and Xerces.

$H6_A$: *The WDR per LOC of God and Brain Classes is different from the WDR per LOC of other classes.*

With respect to change frequency and change size, Tables VII and IX show that the means of the weighted defect rate per LOC for God and Brain Classes are *lower* than for other classes. However, there are great variations in the values, which range from a really small difference in Xerces to up to 84 times higher for Brain Classes than for other classes in Lucene. The hypothesis was supported for God Classes in Lucene and Xerces but not for Log4j. It was not supported for Brain Classes in either of the two systems with Brain Classes.

VII. Discussion

We now discuss the implications of the results for research and practice. Then, we state and assess the limitations of the study.

A. *Implications for Research and Practice*

The results show that without taking the class size into account there is a higher change frequency, change size, and defects rate in God and Brain Classes than in other classes. This is in line with the findings of our previous study [23] and other independent work [10], [11], [13]. However, when the God and Brain Classes in our study were normalized with respect to size, they had smaller values for change frequency, change size, and weighted defect rate than had other classes.

The implication of these results for the research community is that there is a need for further investigation of the effects of normalizing classes and methods with respect to size when considering the effects of code smells on a system's maintainability. A good beginning to such an endeavour may be to analyze the studies already reported in the literature: if the God and Brain Classes in those studies are normalized with respect to size, what will the results be?

For practice, our results indicate that it may be tolerable and even good for a system to have some God and Brain Classes, provided that these classes are constructed intentionally. This observation is consistent with the results of a large experiment on object-oriented control styles: more subjects, in particular the juniors, performed better on the version of a system with a centralized control style than those that worked on a version with a delegated control style, i.e., the most "object-oriented" version of the system [1]. Nevertheless, there is certainly a limit to how large God and Brain Classes can be, and how great a proportion of them there can be, before they become harmful to the system. More work is needed to identify such thresholds in various contexts.

In addition to the foregoing, this work advances the state of the art regarding the analysis of code. We developed a tool that enables the analysis of evolution data stored in source code repositories. It automatically extracts data on change frequency and change size. Such automatic extraction enables studies to be done that could not be done before. For example, it would simply not have been feasible to perform *manually* the analyses that we conducted on the large-scale systems that we investigated.

B. *Threats to Validity*

Internal Validity: The data was collected from the subversion repositories (in each case its development trunk) and bug-tracking systems of each particular system. In the data extraction, we excluded all folders that were not directly related to the functionality of the system itself (test folders, contribution folders, demo folders, etc.). We did not consider whether the source files contained in the actual system folders were actually part of the "real" system or not (e.g., obsolete or discarded code that is still within the repository). Furthermore, we do not claim that the defect data collected from the bug tracking systems is complete, in that it is possible that not all of the defects that occurred during the development of the analyzed systems were tracked during the lifetime of the systems.

External Validity: The reported results stem from the analysis of three open-source systems, each of which represents a different problem domain. However, we do not recommend generalizing our results to other systems without analyzing them to see how similar they are to those that we investigated.

Construct Validity: For the detection of God and Brain Classes, we used the code metric and threshold-based detection strategies defined by Marinescu [19]. These metrics and thresholds have been evaluated for their efficacy in a number of previous studies. However, it has not been evaluated whether their use is appropriate in all contexts. Hence, the precise metrics and thresholds that it is appropriate to use may depend on the context. We did not evaluate their efficacy for use in our study. Hence, it may well be that different metrics and values would have been more appropriate. If different metrics and thresholds had been used, the results may well have been different.

VIII. Conclusion

The study reported herein provides empirical evidence on the issue of how the code smells God Class and Brain Class affect the quality of software systems. We investigated the effects of both smells in three well-known, large-scale open-source systems. Without normalization with respect to size, the analysis shows that God and Brain Classes have a negative effect measured in terms of change frequency, change size, and number of weighted defects. However, when the God and Brain Classes were normalized with respect to size, the results were opposite; these classes were less subject to change and had fewer defects than had other classes. Hence, our results lead us to conclude that as long as the size of the God and Brain Classes is not extreme, and as long as there is not many of them, the presence of God and Brain Classes may be beneficial to a software system. Nevertheless, the extent to which the findings in our study can be generalized to other systems needs to be investigated further.

The results also show that there is a large, positive correlation between system size and the number of God Classes in all three systems. For Brain Classes, there is also a large, positive correlation in Lucene but a negative correlation in Xerces. Hence, the correlation between system size and the number of Brain Classes needs further investigation. Furthermore, we observed that the larger systems have a larger proportion of God and Brain Classes.

In future work, we want to assess the effect on systems of code smells other than those analyzed in this study. Such studies would require an extension of the measures and methodologies applied in this study. We would also like to study other open- and closed-source systems than the ones upon which we report herein, with respect to the effect of both God and Brain Classes, and other smells.

After creating a foundation for analysing the effect of various smells, it would be interesting to take into account the lifecycle of system components that are involved in code smells. Ratiu et al. [26] and Vaucher et al. [27] have taken the first steps in this direction. We believe that future work in this area will result in the generation of useful advice on how to refactor software systems and avoid harmful code smells.

Acknowledgment

The authors thank Prof. Frank-Thomas Nürnberg of the University of Applied Sciences Mannheim for his support with the statistical analysis of the collected data.

References

[1] Arisholm, E., & Sjøberg, D. I. K. (2004). Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software. *IEEE TSE,* 30 (8), 521-534.

[2] Bieman, J. M., & Kang, B.-K. (1995). Cohesion and Reuse in an Object-Oriented System. *Proc. Int'l Symp. Software Reusability*, (259-262).

[3] Chidamber, S. R., & Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. *IEEE TSE*, 20 (6), 476-493.

[4] Chillarege, R., Bhandar, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., et al. (1992). Orthogonal Defect Classification—A Concept for In-Process Measurement. *IEEE TSE,* 18 (11), 943–956.

[5] Deligiannis, I. S., Shepperd, M. J., Roumeliotis, M., & Stamelos, I. (2003). An empirical investigation of an object-oriented design heuristic for maintainability. *Journal of Systems and Software*, 65 (2), 127-139.

[6] Emden, E. V. & Moonen, L. (2002). Java Quality Assurance by Detecting Code Smells. WCRE, 97-108.

[7] Fenton, N., & Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47 (2-3), 149 - 157.

[8] Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). Refactoring: Improving the Design of Existing Code. Addison Wesley.

[9] Moha, N., Guéhéneuc, Y.-G., Duchien, L., & Meur, A.-F. L. (2010). DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE TSE*, 36 (1), 20-36.

[10] Khomh, F., Penta, M. D., & Guéhéneuc, Y.-G. (2009). An Exploratory Study of the Impact of Code Smells on Software Change-proneness. WCRE (75-84). IEEE Computer Society.

[11] Li, W., & Shatnawi, R. (2007). An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. *Journal of Systems and Software*, 80 (7), 1120-1128.

[12] Lorenz, M., & Kidd, J. (1994). *Object-Oriented Software Metrics: A Practical Approach.* Prentice-Hall.

[13] Lozano, A., Wermelinger, M., & Nuseibeh, B. (2007). Evaluating the harmfulness of cloning: a change based experiment. *MSR, p18.*.

[14] Mäntylä, M., & Lassenius, C. (2006). Subjective evaluation of software evolvability using code smells: An empirical study. *Empirical Software Engineering,* 11 (3), 395-431.

[15] Mäntylä, M., Vanhanen, J., & Lassenius, C. (2004). Bad Smells - Humans as Code Critics. ICSM (399-408). IEEE Computer Society.

[16] Marinescu, R. (2001). Detecting Design Flaws via Metrics in Object-Oriented Systems. TOOLS (39) (173-182). IEEE Computer Society.

[17] Marinescu, R. (2002). *Measurement and Quality in Object-Oriented Design (PhD Thesis).* Timisora: "Politehenica" University of Timisora.

[18] Marinescu, R. (2004). Detection Strategies: Metrics-Based Rules for Detecting Design Flaws. ICSM, (350-359).

[19] Lanza, M., & Marinescu, R., (2006). Object-Oriented Metrics in Practice. Springer.

[20] McCabe, T. J. (1976). A Software Complexity Measure. *IEEE Transactions on Software Engineering*, 2 (4), 308-320.

[21] Munro, M. J. (2005). Product Metrics for Automatic Identification of "Bad Smell" Design Problems in Java Source-Code. METRICS '05, p15.

[22] Munson, J. C., & Elbaum, S. G. (1998). Code Churn: A Measure for Estimating the Impact of Code Change. ICSM (24-31). IEEE Press.

[23] Olbrich, S., Cruzes, D., Basili, V. R., & Zazworka, N. (2009). The evolution and impact of code smells: A case study of two open source systems. ESEM 2009, (390-400).

[24] Opdyke, W. F. (1992). Refactoring object-oriented frameworks (PhD Thesis). Urbana-Champaign: University of Illinois.

[25] Rahman, F., Bird, C., & Devanbu, P. (2010). Clones: What is that Smell? MSR (72-81), Cape Town, South Africa, 2010.

[26] Ratiu, D., Ducasse, S., Gîrba, T., & Marinescu, R. (2004). Using History Information to Improve Design Flaws Detection. CSMR (223-232). IEEE Computer Society.

[27] Vaucher, S., Khomh, F., Moha, N., & Guéhéneuc, Y.-G. (2009). Tracking Design Smells: Lessons from a Study of God Classes. WCRE (145-154). IEEE Computer Society.

[28] Zhou, Y., & Leung, H. (2009). Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness. *IEEE TSE*, 35 (5), 607-623.