Code Smells Survival Analysis in Web Apps

Américo Rio
ISTAR-IUL, ISCTE-IUL, Lisbon, Portugal; NOVAIMS, UNL, Lisbon, Portugal

Fernando Brito e Abreu
ISTAR-IUL, ISCTE-IUL, Lisbon, Portugal

This is the Author Peer Reviewed version of the following conference paper published by Springer:

Rio, A., & Abreu, F. B. E. (2019). Code Smells Survival Analysis in Web Apps. In M. Piattini, P. R. D. Cunha, I. García Rodríguez de Guzmán, & R. Pérez-Castillo (Eds.), Quality of Information and Communications Technology : 12th International Conference, QUATIC 2019, Ciudad Real, Spain, September 11–13, 2019, Proceedings (pp. 263-271). (Communications in Computer and Information Science; Vol. 1010). Springer. https://doi.org/10.1007/978-3-030-29238-6_19

Code Smells Survival Analysis in Web Apps

Américo Rio[1,2] and Fernando Brito e Abreu[1]

[1] ISTAR-IUL, ISCTE-IUL, Portugal
{jaasr,fba}@iscte-iul.pt
[2] NOVAIMS, UNL, Portugal
americo.rio@novaims.unl.pt

Abstract. Web applications are heterogeneous, both in their target platform (split across client and server sides) and on the formalisms they are built with, usually a mixture of programming and formatting languages. This heterogeneity is perhaps an explanation why software evolution of web applications (apps) is a poorly addressed topic in the literature. In this paper we focus on web apps built with PHP, the most widely used server-side programming language.

We analyzed the evolution of 6 code smells in 4 web applications, using the survival analysis technique. Since code smells are symptoms of poor design, it is relevant to study their survival, that is, how long did it take from their introduction to their removal. It is obviously desirable to minimize their survival.

In our analysis we split code smells in two categories: scattered smells and localized smells, since we expect the former to be more harmful than the latter. Our results provide some evidence that the survival of PHP code smells depends on their spreadness.

We have also analyzed whether the survival curve varies in the long term, for the same web application. Due to the increasing awareness on the potential harmfulness of code smells, we expected to observe a reduction in the survival rate in the long term. The results show that there is indeed a change, for all applications except one, which lead us to consider that other factors should be analyzed in the future, to explain the phenomenon.

Keywords: code smells, PHP, software evolution, survival analysis, web apps.

1 Motivation

This study is in the crossroads of Software Evolution [1, 2] and Web Engineering [3, 4] and is a follow-up of our preliminary work on the evolution of web systems/applications, regarding maintainability and reliability problems [5, 6]. Web applications (web apps, for short) encompass a heterogeneity of target platforms, since they run both on a browser and a server, and a mix of programming and content formatting languages. That mix makes web apps a more complex target for software quality studies, either synchronic or diachronic (longitudinal), because those studies are often based on static analysis of source code, as is the case with code smells research. Indeed, for each language used (e.g. *JavaScript, html5, CSS, PHP*) a different parser is required, what may explain why software evolution of web apps is a poorly addressed topic in the literature,

as we will see later.

Several perspectives can be adopted in longitudinal studies, were the evolution of software products or processes are analyzed, focusing on aspects such as software metrics, teams' activity, defects identification and correction, or time to release [7, 8]. This paper addresses the survival of code smells in web apps using PHP, the main server-side programming language, currently used in 79% of web apps[1]. Since code smells occurrences are symptoms of poor design and implementation choices, the Software Engineering community has been thriving to reduce their survival, i.e., how long does it take from when they were introduced to when they are removed, by proposing new detection techniques and tools [9, 10]. Despite this interest, there is a shortage of evolution studies on code smells in web apps, especially with PHP [11, 12].

The research described herein covers two factors: scope and period. Code smells effect can vary widely in spreadness. In localized ones, the scope is a method or a class (e.g. *Long Method, Long Parameter List, God Class*), while the influence of others may be scattered across large portions of a software system (e.g. *Shotgun Surgery, Deep Inheritance Hierarchies or Coupling Between Classes*). Since widespread code smells can cause more damage than localized ones, we expect their survival rates to be shorter. The other factor we are concerned with regards a superordinate temporal analysis. Since the topic of code smells has been addressed by researchers, taught at universities and discussed by practitioners over the last two decades, we want to investigate whether this had an impact on their survival. We expect that, in a long term, an increased awareness has caused a more proactive attitude towards code smells detection and removal (through refactoring actions), thus leading to shorter survival rates. Summing up, we aim at testing the following two null research hypotheses:

- *H_0x: Survival does not depend of the code smells scope.*
- *H_0y: Survival of a given code smell does not change over time period.*

To test these hypotheses, we performed a longitudinal study encompassing 4 web apps, and 6 code smells, as surrogates of more scattered or localized scopes.

This paper is structured as follows: section 2 introduces the study design; section 3 describes the results of our data analysis; section 4 overviews the related work and section 5 outlines the major conclusions and identifies required future work.

2 Study design

2.1 Aim of the study

The aim of this work is to study the evolution/survival of code smells in web apps built with PHP in the server-side. We selected a set of applications from different domains, a set of 6 code smells, and collected data across all their stable development versions.

[1] https://w3techs.com/technologies/overview/programming_language/all (accessed: June 2019)

2.2 Applications sampling

Inclusion criteria: (i) fully blown web apps taken from the GitHub top listings, (ii) programmed with object-oriented style[2], (iii) covering a diversity of application areas.
Exclusion criteria: (i) libraries, (ii) frameworks, (iii) web apps built using a framework.

Table 1. Characterization of the target web apps

Web app	Purpose	Versions	Period	LOC	Classes
phpmyadmin	Database manager	158	Aug 2010 - Jun 2018	89788	374
dukuwiki	Wiki solution	40	July 2005 - Jan 2019	271514	402
opencart	Shopping cart solution	28	April 2013 - April 2019	99052	760
phpbb	Forum/bulletin board solution	50	April 2012 - Jan 2018	101556	846

2.3 Code smells sampling

We used PHPMD [13], the only open source tool, we are aware of, that is capable of detecting scattered and localized code smells in PHP. It supports 3 scattered code smells, so we chose the same number of localized ones, although more were available.

Table 2. Characterization of the target code smells

Code Smell	Characterization	Type
ExcessiveMethodLength	The method does too much	Localized
ExcessiveClassLength	The class does too much	Localized
ExcessiveParameterList	The method has too many parameters	Localized
DepthOfInheritance	The class is too deep in the inheritance tree	Scattered
CouplingBetweenObjects	The class has too many dependencies	Scattered
NumberOfChildren	The class has too many descendants	Scattered

2.4 Design of the study

The workflow of our study (see **Fig. 1**) was fully automated by means of batch and PHP scripts. First, all versions of the selected web apps were downloaded from *GitHub*, except the alpha, beta, release candidates and corrections for old versions. Then, using the PHPMD tool, we extracted the location of the code smells from all versions and stored it in XML file format. After some format manipulation, the information on those XML files was stored in a database, to make it amenable for survival analysis. That includes the date when each code smell was first detected and, if that was the case, when it disappeared, either due to refactoring, or because the code where it was detected was (at least apparently) removed. Survival analysis took the input data from the database

[2] Note: PHP can be used in a pure procedural way.

and stored the results in it too. Finally, a data completion step was performed, where results were exported to csv format, for interoperability with visualization tools.

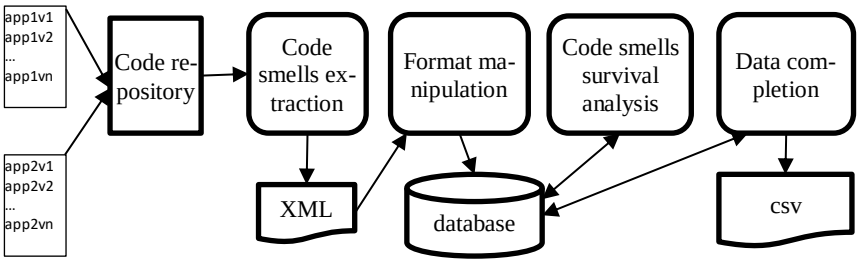

Fig. 1. Workflow of the study

2.5 Survival analysis

Survival analysis encompasses a set of statistical approaches that investigate the time of interest for an event to occur [14]. The questions of interest can be the average survival time, and the probability of survival at a certain point in time. Also, we can calculate the Hazard function, the probability of the event to occur.

The Kaplan-Meier estimator is a non-parametric statistic that allows us to estimate the survival function and it gives the probability that a given code smell will subsist/exist past a time t. The log-rank test is used to compare survival curves of two groups, in our case two types of code smells. It tests the null hypothesis that survival curves of two populations do not differ by computing a p-value (statistical significance). In our analysis we will consider a confidence interval of 95%, corresponding to an observed significance level (known as p-value) of 0.05 for the test hypotheses.

Data extraction and processing, and format for survival analysis.
To apply survival data analysis, we transformed the collection of detected code smells instances for each version of each web app, to a table with the "life" of each instance, including the date of its first appearance, removal date (if occurred) and a censoring value, with the following meaning:

Censoring=1 $\Rightarrow$ the smell disappeared, usually due to a refactoring event;
Censoring=0 $\Rightarrow$ the code smell is still present at the end of the observation period.

For replication purposes, the collected dataset is made available for other researchers[3].

3 Results and Data analysis

To test the hypotheses, we used the R tool and the log-rank test [15]. We performed two kinds of studies, using the log-rank test and 2 different co-variables.

[3] https://github.com/americorio/articledata/

3.1 Comparing the survival curves for different types of code smells

For this study we divided the smells in “Localized Smells” and “Scattered Smells”. We then fitted the Kaplan-Meier curves and performed the log-rank to compute the p-value.

Table 3. Code smells found, removed and survival in days (median and mean), by type.

Web app	Type	Found	Removed	Median(d)	Mean(d)
phpmyadmin	Localized Smells	1067	846	707	744
	Scattered Smells	34	23	324	424
dokuwiki	Localized Smells	159	112	1381	2169
	Scattered Smells	6	2	620	2779
opencart	Localized Smells	798	393	1189	1281
	Scattered Smells	12	0	NA	2172
phpbb	Localized Smells	747	393	2512	2255
	Scattered Smells	20	2	NA	2537

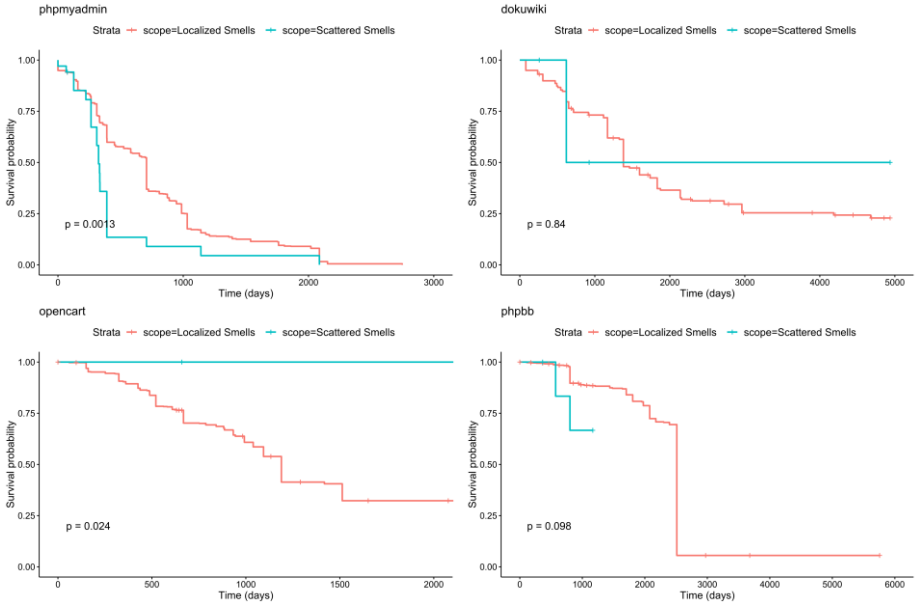

Fig. 2. Survival curves of localized and scattered code smells, by application.

For *phpMyAdmin* and *opencart*, the survival curves of scattered smells differ significantly from those of localized smells (see **Fig. 2**). For *phpbb*, the curves differ, but not with statistical significance (given the considered confidence interval). For *dokuwiki*, the study is inconclusive, since only 2 scattered code smells are removed. It is worth noticing that localized code smells are much more frequent targets for change than scattered code smells. This may be due to the lack of refactoring tools that support the automated removal of scattered code smells in PHP.

3.2 Comparing code smells survival curves for different time frames

Here we used the log-rank test, and created a co-variate "timeframe", with two values 1 and 2, 1 being the first half of historic data, and 2 the second half. For the first half, we truncated the variables of the study as if it was a sub-study ending in this period. In other words, we considered two independent observation periods.

Table 4. Code smells found, removed and survival in days (median and mean) by timeframe.

Web app	Timeframe	Found	Removed	Median(d)	Mean(d)
phpmyadmin	1 (< 2014-07-01)	498	393	946	787
	2 (>=2014-07-01)	603	371	451	473
dokuwiki	1 (< 2012-04-03)	94	53	2139	1582
	2 (>=2012-04-03)	71	52	1381	1352
opencart	1 (< 2016-03-31)	496	276	1016	794
	2 (>=2016-03-31)	314	35	NA	966
phpbb	1 (< 2015-02-15)	685	79	NA	986
	2 (>=2015-02-15)	82	1	NA	952

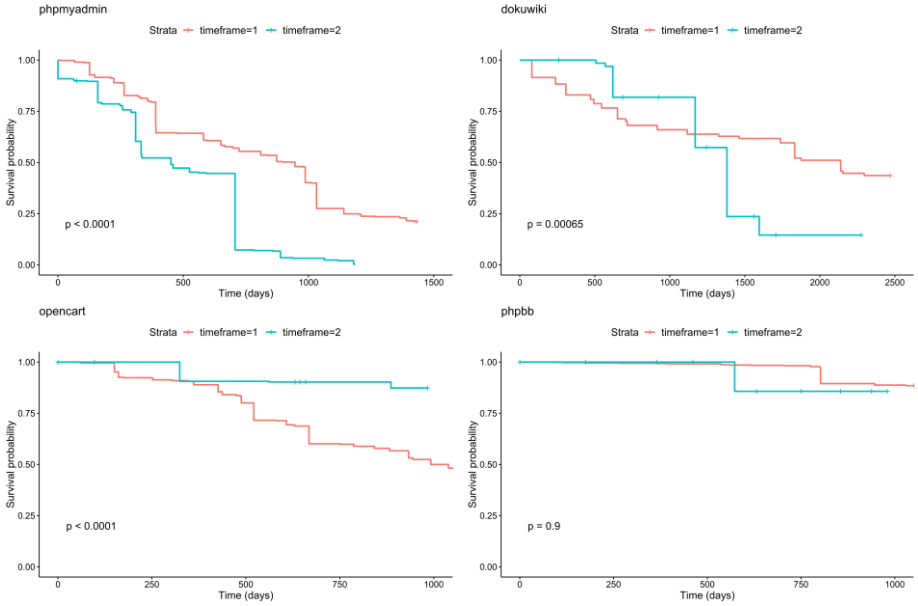

Fig. 3. Code smells survival curves in two consecutive timeframes (1st and 2nd half)

For all projects except *phpbb*, the survival curves of the 1st timeframe differ significantly from the 2nd one (see **Fig. 3**). In *phpMyAdmin* and *dokuwiki* the area under the code smells survival curves is smaller in the 2nd timeframe, what seems to corroborates our expectation that, due to the increasing awareness on the potential harmfulness of code smells, we would observe a reduction in the code smells survival rate in the long term. However, for *opencart* and *phpbb*, we cannot draw the same conclusion, probably because the number of detected and removed code smells is much smaller.

4 Related work

In [11], the authors study five web apps in PHP, the aspects of their history, unused code, the removal of functions, the use of libraries, the stability of interfaces, migration to object-orientation and the evolution of complexity. They found these systems undergo systematic maintenance.

In [12], the authors analyze 30 PHP projects for their metrics, to examine if the Lehman's laws of evolution are maintained in an web app, and found that not all of them stand true for this kind of projects.

In [16], the authors analyze two Java open source systems and four code smells, in a longitudinal study with versions. They study the evolution of code smells, including their persistence, and do survival analysis to find the average time of persistence.

In [17], after a survey to developers, the authors analyze when test smells occur in source code, what their survivability is, and whether their presence is associated with the presence of design problems in production code (code smells). They found, among other conclusions, relationships between test and code smells. They extracted data from a Git repository from 3 ecosystems making a total of 152 projects. They employ, among other techniques, survival analysis to study the longevity of test smells.

In [18] the authors address the faults in the releases of five JavaScript projects (1 framework, 2 libraries and 2 command line programs) and try to relate them to 12 types of code smells. They employ survival analysis for the faults.

The techniques used to extract the code for further analysis are divided in mining software repositories and getting full versions or mining Git repositories for the changes. However, to compute scattered code smells, we had to deal with the full code. For smells based only on metrics for the file, the Git extraction is simpler to automate.

5 Conclusions

As far as we know, this is the first study analyzing code smells in PHP web apps. We studied the evolution of 6 groups of code smells, in 4 web applications built with PHP. For the first hypothesis we found that the code smells survival curves indeed vary with the type of code smell, localized or scattered. We also found that the insertion and removal events are much lower for the scattered code smells. Nevertheless, for PHP web apps developed with object-oriented paradigm, it is not enough to study localized smells in the file scope, but we also must address the smells that are scattered across the system, since the latter are potentially more harmful.

For the second hypothesis results, in 3 of the applications the survival curve varies between timeframes, but for one (*phpbb*) we do not observe this variance. This can be explained by the low relative removal of code smells compared to the other applications. We cannot fully sustain that the long-term reduction in survival rate of code smells in PHP web apps is due to the awareness on potential code smells harmfulness. Other factors should be analyzed in the future, to explain the phenomenon.

The biggest validity threats are: there should be a bigger number of apps and smells studied; perceive if applications that implemented OOP gradually, impact the study.

In Memoriam Acknowledgment

We are grateful to the late Professor Rui Menezes (deceased 14 May 2019), whose contribution to this work was of great significance. He encouraged and supported us on the usage of survival analysis techniques and inspired us with his enthusiasm.

References

1. Mens, T., Demeyer, S. eds: Software Evolution. Springer (2006)
2. Rossi, G., Pastor, O., Schwabe, D., Olsina, L.: Web engineering: modelling and implementing web applications. Springer Science & Business Media (2007)
3. Mendes, E., Mosley, N.: Web engineering. Springer Science & Business Media (2006)
4. Rio, A., E Abreu, F.B.: Analyzing web applications quality evolution. In: Iberian Conference on Information Systems and Technologies, CISTI. pp. 1–4. IEEE (2017)
5. Rio, A., Brito e Abreu, F.: Web Systems Quality Evolution. In: QUATIC. pp. 248–253. IEEE (2016)
6. Herraiz, I., Rodriguez, D., Robles, G., Gonzalez-Barahona, J.M.: The evolution of the laws of software evolution: A discussion based on a systematic literature review. ACM Comput. Surv. 46, 28 (2013)
7. Radjenović, D., Heričko, M., Torkar, R., Živkovič, A.: Software fault prediction metrics: A systematic literature review. Inf. Softw. Technol. 55, 1397–1418 (2013)
8. w3techs: Usage Statistics and Market Share of Server-side Programming Languages for Websites, May 2019, https://w3techs.com/technologies/overview/programming_language/all
9. Zhang, M., Hall, T., Baddoo, N.: Code bad smells: a review of current knowledge. J. Softw. Maint. Evol. Res. Pract. 23, 179–202 (2011)
10. Fernandes, E., Oliveira, J., Vale, G., Paiva, T., Figueiredo, E.: A review-based comparative study of bad smell detection tools. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. p. 18. ACM (2016)
11. Kyriakakis, P., Chatzigeorgiou, A.: Maintenance Patterns of Large-Scale PHP Web Applications. 2014 IEEE Int. Conf. Softw. Maint. Evol. 381–390 (2014). doi:10.1109/ICSME.2014.60
12. Amanatidis, T., Chatzigeorgiou, A.: Studying the evolution of {PHP} web applications . Inf. Softw. Technol. . 72, 48–67 (2016). doi:http://dx.doi.org/10.1016/j.infsof.2015.11.009
13. Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Penta, M. Di, Lucia, A. De, Poshyvanyk, D.: When and Why Your Code Starts to Smell Bad.
14. Clark, T.G., Bradburn, M.J., Love, S.B., Altman, D.G.: Survival analysis part I: basic concepts and first analyses. Br. J. Cancer. 89, 232 (2003)
15. Daniel Schuette: Survival Analysis in R Tutorial (article) - DataCamp, https://www.datacamp.com/community/tutorials/survival-analysis-R
16. Chatzigeorgiou, A., Manakos, A.: Investigating the evolution of bad smells in object-oriented code. In: Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the. pp. 106–115. IEEE (2010)
17. Tufano, M., Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., Poshyvanyk, D.: An empirical investigation into the nature of test smells. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 4–15 (2016)
18. Saboury, A., Musavi, P., Khomh, F., Antoniol, G.: An empirical study of code smells in JavaScript projects. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 294–305 (2017)