Analyzing Web Applications Quality Evolution

Américo Rio
Fernando Brito e Abreu
Instituto Universitário de Lisboa (ISCTE-IUL)
ISTAR-IUL
Lisboa, Portugal
{jaasr, fba}@iscte-iul.pt

Abstract — **Software evolution is a well-established research topic, but not in the web applications area. Web projects are normally more complex than other software development projects because they have both server and client code, encompass a variety of programming languages, and are multidisciplinary.**

We aim to produce a catalog of web smells to help mitigating quality problems in web apps implementation, thus saving time and reducing cost. By means of longitudinal studies, we plan to analyze the impact of these web smells in web apps maintainability and reliability. This paper describes several particularities of the proposed research work, as well as introduce procedures and techniques to be used.

Keywords—software evolution, web code smells, web engineering, software quality, longitudinal studies, irregular time series

I. INTRODUCTION

The main goal of this research is to reduce time and cost spent in the initial development and maintenance of a web project, and to improve its global quality. A web project, like any software development project, evolves with time, due to changes in the environment and requirements, and it is essential to mitigate the problems and malformations in code that cause delays in releases and bugs.

Web systems or applications are normally more complex than other software development projects, because they have the server part (that can run in a remote server or cloud server) and the client part (that is displayed and runs in a browser). They are also more complex because they encompass a mix of programming languages (e.g. *PHP*, *C#*, *JavaScript* and *Java*), and formatting and content languages (e.g. *HTML* and *CSS*).

We previously proposed an automatic model for the evaluation of web sites supported by a tool that extracted around 60 metrics from the client code [1], retrieved automatically from online sites. We could also perform with success a validation of the metrics and the model, and compared it with the Google page rank, using statistical methods. The limitations of that study were that: i) despite our quality model was comprehensive, only objective metrics were used, leaving out subjective aspects that only humans could assess and ii) the code analyzed was only from the client side.

We will deal now with the complete code, both at the client and server sides of a web application. We will address the quality of web systems in a longitudinal perspective, as an evolution study. This requires choosing metrics that can serve as adequate surrogates for software quality. That choice is not consensual for various reasons [2]. For instance, in a previous study of ours, we found that some web systems quality characteristics may depend on the application domain [3]. To mitigate this problem, several authors have chosen to study the evolution of code smells [4-6]. We plan to do that, but in the scope of web systems. In concrete, our prospects are to perform various longitudinal studies on open-source web systems to find evidence on the relation between the existence of the web smells – the explanatory variables – and maintainability and reliability problems, such as release delays, failures occurrence and faults (aka defects or bugs) density, which will be the outcome variables.

Thus, we now aim at researching if maintainability and reliability problems found in web projects may be due to the violation of fundamental software design and coding principles. *Code smells* (aka *bad smells*), typically indicate those violations, require refactoring and, if spotted early, can improve quality and save time [7]. Code smells are not bugs, since they do not prevent a program from functioning, but rather symptoms of software maintainability and reliability problems.

Web smells are code smells in the specific context of web systems. Since awareness is fundamental for prevention purposes, we need a *web smells catalog*. Such a documented ontology of web smells can become an important learning instrument, and a good starting point for producing web smells detection and refactoring tools.

We already introduced our research proposal in a previous paper [8]. In this one we will provide more details on the procedures and techniques to be used.

This document is structured as follows: section II reviews the related work; section III describes the experimental procedures to be applied; finally, in section IV, we present some preliminary conclusions.

II. STATE-OF-THE-ART

A. Web Systems Smells

"*Bad Smells in Code"* was an essay by Kent Beck and Martin Fowler [9], published as a chapter of the famous book *"Refactoring – Improving the Design of Existing Code"* [7]. Since then, the term gained popularity and there are many studies about the subject, normally using *Java*, although the main ideas can be applied to any object-oriented programming language. A comprehensive list of code smells can be found in

a paper by Mantila et al. [4], as well as online in various sources [10-12].

The published work on *web systems smells* is considerably scarce. Some papers on web applications have focused on clone detection [13] (just one among many smells) and tools for code smells detection on client-side *JavaScript* [14, 15]. These studies mostly concern client-side programming issues, therefore not covering all the spectrum of relevant issues. As for web systems smells on the server-side, published work is even scarcer. In the next two sections, we will review and comment published works on both sides.

1) Client side

Hung Viet Nguyen et al. [15] propose a list of 6 client side smells, mainly concerning *JavaScript (JS)* and *CSS*: They claim that *WebScent* is a tool for detecting embedded code smells in server code, but detected smells lie only on the client side (i.e. the smells are only in the part of code that runs in the browser – *HTML, CSS, JavaScript*). However, to detect code duplication, one must examine the server code, because a lot of code will be perceived as duplicated in the browser, but it is really a server side include and not repeated (i.e. a false clone).

Fard et al. [14], a year later, proposed a set of 13 *JavaScript* code smells. This set of code smells is considerably based on the Fowler's catalog [9]. The authors developed a tool – *JNose* – to automate their collection. This tool uses a web crawler, so it can only analyze the client side.

2) Server side

The most used server side programming language, *PHP*, accounts for over 80% of the server programing in the world [16]. A good choice for our study will then be *PHP*. Besides its representativeness, *PHP* projects are normally open source and therefore we can analyze its code.

As shown in [17], most of Martin Fowler's code smells [7] still make sense for *PHP*. However, we could not find any published work on the corresponding detection algorithms. The closest we could find was the *PHPMD* tool [18]. The latter is a rule-based static analyzer of *PHP* source code base and looks for several potential problems within that source that can be code smells, and one can define a ruleset and use it accordingly.

B. Longitudinal Studies in Software Development

To obtain evidence that web systems smells may in fact cause problems, we need to perform longitudinal studies. We did not find such studies in the web engineering area. Nevertheless, there are a few important studies with code smells that, albeit outside the web systems scope, are worth mentioning, as follows:

Mäntylä et al. [4] describe a survey they performed on developers, where they concluded that organizations should make decisions regarding software evolvability improvement, based on a combination of subjective evaluations and code metrics.

Olbrich et al. [5] discuss the effect that two code smells *(God and Brain Classes)* have in the quality of software systems. This study uses the information on bugs found, and instead of considering major releases, this study divides the schedule into chunks of 50 code revisions. The authors concluded that the presence of the aforementioned code smells is not necessarily harmful and such classes may be an efficient way of organizing code.

Ouni et al. [6] studied five medium and large-size open-source systems and four types of code smells. They used data mining techniques upon the change history data available on control versioning systems. Their experimental results show the effectiveness of the approach, compared to three different state-of-the-art approaches, with more than 85% of code smells fixed and 86% of suggested refactorings semantically coherent when the change history is used.

III. EXPERIMENTAL PROCEDURES

In our previous paper on this PhD research topic [8], we outlined some of the research questions, and expected contributions. In this paper, we present how we are going to answer the questions, i.e. the experimental procedure. The latter will include four steps: systematic literature review, building of web smells catalog, web smells detection and evolution studies, as described in this section.

A. Systematic literature review (SLR)

SLRs are secondary studies that provide researchers and practitioners a vehicle to gain access to distilled evidence synthesized from results of multiple original studies (aka primary studies). SLRs thus substantially reduce the time and expertise it would take to locate and subsequently appraise and synthesize primary studies. To delimit bias in our SLR, we follow the guidelines provided in [19].

We are interested in longitudinal studies, or evolution studies, that analyze the quality of applications according to time or versions, and deal with empirical data. This led to the questions asked in this SLR, which are:

i) What are the attributes that have been used to describe web software quality in longitudinal studies?

ii) What are the factors that have been found to influence the evolution of web software quality?

iii) Which techniques have been used to deal with the unevenly time-spaced nature of development data in web software evolution studies? In other words, how can we model the web software evolution phenomenon to allow forecasting?

Our search string, to apply on several databases, crosses the definition of web application /system / software with evolution studies in various ways:

(("web application" OR "web-based application" OR "web system" OR "web software") AND ("longitudinal study" OR "time series" OR "software evolution"))

We planned to use six databases, however we had to take Springer out because it was not possible to use our search string in it – it always searches for every term. In the hope that

Springer articles are indexed in Scopus or ISI, we used the results from five databases: Scopus, IEEE, ACM, ISI - Web of knowledge and Science Direct.

The search produced a result of 308 articles. After a first filtering pass, we identified 62 articles with a related subject, marked YES and MAYBE, and we are currently in the deep analysis process. In this second pass, some of the articles will be taken out. We already identified some longitudinal studies, but none deals with web code smells. This is a work in progress, to be published soon.

B. Building of Web Smells Catalog

Some preliminary attempts have been made to define web smells, as described in section II, but they have limited coverage and their validation was almost exclusively done through peer review in the corresponding publication fora. To the best of our knowledge, there is no comprehensive web smells catalog that addresses both client and server sides.

Setting up a web smells catalog should be more than an experienced practitioner's exercise based on "gut feeling". To validate our catalog and hopefully obtain an initial consensus on the relevance of each web smell contained in it, we are preparing a collaborative web platform to support a large-scale survey on practitioners, both from academia and industry. In this platform, it is possible to propose new web smells and vote in the existing ones. This will serve two purposes: to reduce the amount of subjectivity in the catalog proposal; to increase the external validity of each proposed web smell, through peer assessment.

The addition of new web smells and the issuing of votes are logged operations to avoid tampering with the catalog construction. We hope that this collaborative construction of the catalog yields better results than its elaboration by just one or two researchers.

The metadata on each web smell includes, but is not limited to, the following attributes:

TABLE I. WEB SMELL ATTRIBUTES

Attribute Name	***Description***
Name	Web smell name
Short description	Brief description / abstract
Long description	Detailed characterization of this web smell
Detection algorithm	Algorithm in pseudocode
Author name	Author of the proposed smell
Author affiliation	Author affiliation
Proposal date	Date when the web smell was proposed
Votes	Number of votes the smell gets

All proposals and votes are logged to allow recording its author/issuer and corresponding affiliation.

This catalog will include parts for the client side and for the server side. The server part will cover *PHP* that accounts for over 80% of the market (see section II). A problem to be faced here is that *PHP* can be used in a procedural or object-oriented manner. The client part will cover *JavaScript*, *HTML* and *CSS*.

We started to produce a candidate catalog composed of 22 web client smells and 28 server smells. Some of them, especially on the client side, were taken from the literature (see section II.A). We will continue this preliminary web smells identification task as a jump-start in the platform.

C. Web Smells Detection

Depending on each concrete web smell type, different detection techniques may be required [20]. Although there is a considerable amount of research on code smells detection techniques, their application on the context of web systems is a largely uncovered topic.

Collecting web smells location data manually is unfeasible. We have already identified two possible approaches for automating their collection:

The *direct approach* is the one based on lexical analysis and typically implies developing algorithms based on abstract syntax tree (AST) manipulation. For the server side, we plan to extend the open-source *PHPMD* tool. As for the client side languages, we can build upon some open-source lexical analyzers for *JavaScript*, *HTML* and *CSS*, since developing those analyzers from scratch would take too long. The downside of this approach is that the implementation of the algorithm for detecting the same web smell will be different for each target language, since it will be dependent on the corresponding AST.

In the *reverse engineering approach*, we transform the code into an instance of a higher abstraction model (a metamodel) and then detect the web smells by operating upon that metamodel. We are currently experimenting with OMG's Knowledge Discovery Metamodel (KDM). This KDM-based approach can be done with a generalized tool, Modisco[1], using a "PHP discoverer", or with a specialized parser. The Modisco approach is preferred because it can be extended to other server languages and purposes.

In both approaches we will explore and compare the results of different detection algorithms, either using thresholds, traditional statistics or machine learning techniques [21].

D. Evolution Studies

Based on our preliminary literature review, we could conclude that the evolution of web systems quality is mostly an unknown phenomenon, since we could not find published works on this topic.

We will first perform the descriptive statistics of the variables involved, and then the evolution studies. The first obvious aspect we will address is the relative distribution of web smells. If that distribution varies throughout time, it is worth understanding why. If there is a co-occurrence of two smells, are they assessing the same aspect, or is there some causality effect? The relevance of each code smell, along with its relative frequency, will be an interesting decision factor for refactoring. In other words, relevant web smells that occur more often are the ones that should be considered as first candidates for refactoring. If defects are classified in the issue

[1] https://www.eclipse.org/gmt/modisco/infrastructure/KDM/

tracking system, we will also assess if it is possible to forecast certain types of defects based upon the occurrences of one or more code smells. Generically speaking, we will observe how web smells manifest themselves in large open-source web systems, namely if they have some impact on maintainability and reliability problems, such as release delays, failures occurrence and faults density.

The expected outcome of these quasi-experimental studies will hopefully help increasing the awareness on the importance of detecting web systems smells as early as possible. Removing them is expected to reduce the failure potential, as well as the time spent developing new features, in other words, improving web systems reliability and maintainability.

Statistical longitudinal studies in software engineering have a major drawback: web software systems are released at unequally spaced time intervals. Traditional time series techniques (e.g. *ARMA* and *ARIMA*) are therefore not appropriate, since they assume that data is collected at a constant pace. To mitigate this issue, we plan to use irregular time series techniques that have been used, for instance, to predict the stock market volatility [22] and in electronic commerce research [23]. This is an active research area and new algorithms have been recently proposed [24]. For detecting anomalies in those longitudinal time series, we plan to use an existing library, either in R [25] or in Python [26].

IV. Conclusion

In this paper, we presented some details and progress from the previous paper. We showed some preliminary results in the SLR, that we hope to publish soon. We briefly detailed the process we are using to build a web smells catalog, namely the prospects for a collaborative platform to mitigate subjectivity in that catalog. We also introduced our current work in web smells detection in the server side, where we are comparing an AST based approach to a metamodel-based approach. Last, but not the least, we hope our research efforts will contribute to the Web Engineering community, by providing a systematic approach for analyzing web applications quality evolution, where irregular time series techniques will be used. We expect our results will help increasing the awareness for the use of web smells detection techniques, with the hopefully increase in maintainability and overall quality of web applications.

References

[1] Rio, A., Modelo Automático de Qualidade para Sítios Web (MSc Thesis). 2010, FCT/UNL.

[2] Drouin, N., M. Badri, and F. Touré. Metrics and Software Quality Evolution: A Case Study on Open Source Software. in Proceedings of the 5th International Conference on Computer Science and Information Technology, Hong Kong. 2012.

[3] Rio, A. and F. Brito e Abreu. Websites Quality: Does It Depend on the Application Domain? in Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the. 2010.

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

[5] Olbrich, S.M., D.S. Cruzes, and D.I.K. Sjoberg. Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. in Software Maintenance (ICSM), 2010 IEEE International Conference on. 2010.

[6] Ouni, A., et al., Improving multi-objective code-smells correction using development history. Journal of Systems and Software, 2015. 105: p. 18-39.

[7] Fowler, M., Refactoring: improving the design of existing code. 1999: Addison-Wesley Longman Publishing Co., Inc. 464.

[8] Rio, A. and F.B. e Abreu. Web Systems Quality Evolution. in Quality of Information and Communications Technology (QUATIC), 2016 10th International Conference on the. 2016. IEEE.

[9] Beck, K., M. Fowler, and G. Beck, Bad smells in code. Refactoring: Improving the design of existing code, 1999: p. 75-88.

[10] A Taxonomy for "Bad Code Smells" 2015-07-30]; Available from: http://mikamantyla.eu/BadCodeSmellsTaxonomy.html.

[11] Atwood, J. Code Smells on Code Horror Blog. 2015-07-30]; Available from: http://blog.codinghorror.com/code-smells/.

[12] Fowler, K.B.a.M. Code smells on sourcemaking.com. 2015-07-30]; Available from: https://sourcemaking.com/refactoring/bad-smells-in-code.

[13] Lanubile, F. and T. Mallardo. Finding function clones in web applications. in Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on. 2003. IEEE.

[14] Fard, A.M. and A. Mesbah. JSNOSE: Detecting JavaScript Code Smells. in Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on. 2013.

[15] Hung Viet, N., et al. Detection of embedded code smells in dynamic web applications. in Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on. 2012.

[16] W3Techs. Usage of server-side programming languages for websites. 1-4-2016]; Available from: http://w3techs.com/technologies/overview/programming_language/all.

[17] Reiersøl, D., Code smells in PHP, in International PHP Conference 2009, 15-18 Nov. 2009: Karlsruhe Congress Center.

[18] PHP Mess Detector. 2015-07-31]; Available from: http://phpmd.org/.

[19] Kitchenham, B., Procedures for performing systematic reviews. 2004.

[20] Fontana, F.A., P. Braione, and M. Zanoni, Automatic detection of bad smells in code: An experimental assessment. Journal of Object Technology, 2012. 11(2): p. 5:1-38.

[21] Fontana, F.A., et al., Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, 2016. 21(3): p. 1143-1191.

[22] Dionisio, A., R. Menezes, and D.A. Mendes, An econophysics approach to analyse uncertainty in financial markets: an application to the Portuguese stock market. The European Physical Journal B - Condensed Matter and Complex Systems, 2006. 50(1-2): p. 161-164.

[23] Jank, W. and G. Shmueli, Functional data analysis in electronic commerce research. Statistical Science, 2006. 21(2): p. 155-166.

[24] Eckner, A., Algorithms for unevenly-spaced time series: Moving averages and other rolling operators. 2012, Working Paper.

[25] Twitter AnomalyDetection R package. 2015; Available from: https://github.com/twitter/AnomalyDetection.

[26] A Python port of Twitter's AnomalyDetection R Package. 2016.