Web Systems 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 area, but not in the area of web systems/applications. 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 avoiding the problems in web development code before they happen, thus saving time and reducing cost. By means of longitudinal studies we plan to analyze the impact of these web smells in web systems maintainability and reliability. This will require developing a tool to detect the proposed web smells. For validation sake, we will also use surveys among web systems developers and peer reviewing in academic *fora*.

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

I. INTRODUCTION

THE work presented herein is within the scope of Web Engineering, a discipline that grew out of Software Engineering, due primarily to the multidisciplinary nature of web projects (e.g. encompassing web designers and marketers, in addition to the normal stakeholders of software development), specific application characteristics (e.g., navigation, authentication or handling of multimedia contents), somehow distinct development process (e.g., emphasis on design and shorter lead times), availability (always online), underlying structure (architecture and network), legal or ethical issues. Web Engineering encompasses the design, development, evolution, and quality evaluation of web applications [1-3].

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

Our research deals with the quality of web systems. Their quality attributes have been discussed for a while [4]. 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 [5]. This code was retrieved from online sites. The metrics collection tool was developed on top of a web crawler. We could also perform a validation of the metrics and the model, and compared it with the Google page rank, using statistical methods and we had success in achieving a validation.

Limitations of this study: 1) A complete quality model will have to have subjective metrics, and these at this time are difficult to retrieve automatically. However not impossible. 2) The code analyzed is only form the client side. This is the way to make the process automatic but from the perspective of the project and the code it falls short.

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 [6]. 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 documented ontology of web smells, can become an important teaching instrument, and also a challenge for the research community, since we will also look forward ways of automatically detecting web smells and refactoring web systems code to get rid of them, or at least mitigate their effect.

A. Software Evolution and Software Quality Evolution

Software evolution became an accepted research area in software engineering. Software evolution is a phenomenon worth studying because it poses serious problems to software projects, encompassing many dimensions and affecting, among others, all phases of the software process, managerial and economic aspects, and programming languages and environments [7].

Software quality evolution is an active research topic and published works often focus on the longitudinal observation of defects [8-10]. Indeed, monitoring software quality evolution for defects over time can help quality assurance teams to forecast and prioritize their efforts. Another research thread in this area is analyzing the trend of software design quality

978-1-5090-3581-6/16 $31.00 © 2016 IEEE
DOI 10.1109/QUATIC.2016.59

degradation due to long-term maintenance activities, such as functionality extension and bug fixing [11]. Again, monitoring the degradation trends of software design may provide useful feedback for evolution decisions.

Software evolution studies, where we have to consider a series of software system versions, has serious threats such as the fact that the environmental setup, including the human factor (development / maintenance team) most probably changed throughout the observation period.

B. Web Software Quality Evolution

We will address the quality of web systems in a longitudinal perspective. This requires choosing metrics that can serve as adequate surrogates for software quality. That choice is not consensual for various reasons [12]. For instance, in a previous study of ours, we found that web systems quality characteristics may depend on the application domain [13]. To mitigate this problem, several authors have chosen to study the evolution of code smells [14-19]. We plan to do that, but in the scope of web systems. In concrete, our prospects are to perform some 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.

This document is structured as follow: section II reviews the related work; section III identifies the research questions and provides an overview of the expected contributions; 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 [20], published as a chapter of the famous book *"Refactoring – Improving the Design of Existing Code"* [6]. Since then, the term gained popularity and there is a large number of studies about the subject, normally using the *Java* programming language, although the main ideas but 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. [14], as well as online in various sources [21-23].

The published work on *web systems smells* is considerably scarce. Some focus on usability [24-27], accessibility and navigation smells [28], but are outside the aforementioned declared scope of our work (reliability and maintainability). Other papers on web applications have focused on clone detection [29] (just one among many smells) and tools for code smells detection on client-side *JavaScript* [30, 31]. These studies mostly concern client-side programming issues and do not fully cover 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. [31] propose a list of 6 client side smells mainly concerning *JavaScript (JS)* and *CSS*: *JS in HTML (Separation of Concerns); CSS in JS (Separation of Concerns); CSS in HTML (Separation of Concerns); Scattered Sources (Software Modularity); Duplicate JS (Software Modularity); HTML Syntax Error (Coding Standards)*. 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. [30], a year later, proposed a set of *JavaScript* code smells, as follows: *Closure smell (Function), Coupling JS/HTML/CSS (File), Empty catch - Lines of code (Code block), Excessive global variables (Code block), Large object (Object), Lazy object (Object), Long message chain (Code block), Long method/function (Function), Long parameter list (Function), Nested callback (Function), Refused bequest (Object), Switch statement (Code block), Unused/dead code (Code block)*. This set of code smells is considerably based on the Fowler's catalog [20]. The authors developed a tool – *JNose* – to automate their collection. This tool uses a web crawler, so apparently it can only analyze the client side of a web application.

2) Server side

The most used server side programming language, *PHP*, accounts for over 80% of the server programing in the world [32]. *C#* in *Asp.Net* comes in second with around 17%, but some of the code will be proprietary, although the language itself is published as an ISO standard. *Java* is also used in around 3% of the web sites as the server programming language. A good choice for our study will then be *PHP*. Besides its representativeness, *PHP* projects are normally open source and we can analyze the code.

As shown in [33], most of Martin Fowler's code smells [6] 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 [34], a spin-off project of *"PHP Depend"* that aims to be a *PHP* equivalent of the *Java* tool *PMD*. *PHPMD* 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. These problems can be things like possible bugs, suboptimal code, overcomplicated expressions, unused parameters, methods and properties. Like in *PMD*, 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:

The Subjective evaluation of software evolvability using code smells: An empirical study [14] describes a study made

with a survey to developers. Authors suggest, as a conclusion, that organizations should make decisions regarding software evolvability improvement based on a combination of subjective evaluations and code metrics.

The evolution and impact of code smells: A case study of two open source systems [16] presents a study of the "*God Class*" and "*Shotgun Surgery*" code smells evolution. They claim to have identified different phases in the evolution of code smells during system development and that code smell infected components exhibit a different change behavior.

Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems [17] discusses the effect that these code smells have in the quality of those software systems. This study uses the information on bugs found, recorded on the *Bugzilla* issue tracking system. Instead of considering major releases, this study divides the schedule into chunks of 50 code revisions, i.e., code is analyzed every 50 versions. The authors conclude that the presence of the two target code smells *(God and Brain Classes)* is not necessarily harmful and such classes may be an efficient way of organizing code.

Improving multi-objective code-smells correction using development history [35] uses five medium and large-size open-source systems and four types of code-smells *(Blob, Spaghetti Code, Functional Decomposition, and Data Class)*. They use data mining techniques upon the change history data available on control versioning systems, notably CVS, and SVN. 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 refactoring semantically coherent when the change history is used.

III. RESEARCH QUESTIONS AND EXPECTED CONTRIBUTIONS

In our research on web systems quality evolution we will try to address the following research questions:

A. Research Questions

(RQ1) What are the most relevant web systems smells?

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 can be (should be) more than an experienced practitioner's exercise based on "gut feeling". As scientists, we should produce evidence that the presence of those web smells actually causes problems in web projects. For that purpose empirical experiments are required.

(RQ2) Can web systems smells location be detected automatically?

Collecting web systems smells location data manually is unfeasible. Depending on the answer to the previous question (i.e. depending on each concrete web smell) different detection techniques may be required [36]. 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.

(RQ3) Is web systems quality evolution, in terms of maintainability and reliability, influenced by the presence of web smells?

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. If the presence of smells stands as a catalyst for reduced reliability or maintainability, then detecting and removing those smells is expected to produce improvements in those quality characteristics.

(RQ4) How can we deal with uneven time-spaced software development data?

Open source projects, which are the ones amenable as targets for research purposes, usually evolve at an irregular pace, since they depend on the availability of their team members. The latter are, most times, unpaid volunteers that commit their contributions whenever they can, not as an obligation, like in commercial projects. As a consequence, major and minor software releases – the development cycles of web systems projects – are not evenly time-spaced [37, 38]. If we take data from committed releases, then the precondition for regular time intervals that is a precondition for the standard time-series analyses techniques that are usually used in longitudinal studies, like *ARMA* and *ARIMA*, is not met, therefore rendering conclusions useless. In fact, most of the basic theory for time series analysis was developed at a time when limitations in computing resources favored an analysis of equally spaced data, since in this case efficient linear algebra routines can be used and many problems have an explicit solution.

B. Expected Contributions

In our quest for answering the aforementioned research questions, we expect to produce the following contributions:

(EC1) Web smells catalog

This catalog will have two parts: one for the client side and another 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 have recently produced a candidate catalog composed of 22 web client smells and 28 server smells. Their description cannot be included here due to space restrictions. Some of them, especially on the client side, were taken from the literature (see section II.A).

To validate our catalog and hopefully obtain an initial consensus on the relevance of each web smell contained in it, we are now preparing a large scale survey on practitioners'

communities such as *Web Professionals Connect*[1], *Stack Overflow*[2], *Web Developer Forum*[3], or the *PHP Freaks Forum*[4]. Our survey strategy will encompass several rounds. In each round we will improve the catalog, before offering it for evaluation on the next round. The results of this survey are expected to help answering RQ1.

(EC2) Collection tools for the proposed catalog

For feasibility sake, the detection of web systems smells is expected to be as much automated as possible. For the server side, we plan to extend the open-source *PHPMD* tool. As for the client side languages, we have not yet identified an appropriate open-source tool for extension, but regarding its architecture, we will adopt a plugin architecture for extensibility purposes, based on our previous experience of developing the *SmellChecker* tool [39]. We plan to give free access to this tool online, as a service, or alternatively as a downloadable application.

The most important handicap pointed out for the detection of Fowler's code smells [20] is related to their informal definition that induces ambiguity [40]. Our exercise of developing an automated support for web smells collection will provide us a better insight on this problem. In the end, it will allow us to refine web smells definition to mitigate that handicap. In other words, besides answering RQ2, we expect that this contribution will also help improving EC1.

(EC3) Descriptive studies on web systems development

Descriptive statistics on process and product data will help understanding the web systems development phenomenon. Our sample of web systems will be taken from open-source repositories such as *GitHub*[5] or *SourceForge*[6]. We will mine those repositories, namely their issue tracking systems, for obtaining data on failures and defects. As for the web smells and release dates, we will mine their configuration management / version control systems.

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 (as graded by practitioners), along with its relative frequency (as detected by our tool), 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 tracking system, we will also be able to relate/predict the relative frequency of a particular kind of defect to a particular code smell (or group of). Besides analyzing the aforesaid aspects, these descriptive studies will also allow us to assess the validity of the code smells collection tools (EC2).

[1] www.linkedin.com/groups/3002424/

[2] stackoverflow.com/

[3] www.webdeveloper.com/forum/

[4] forums.phpfreaks.com/

[5] github.com

[6] sourceforge.net

(EC4) Systematic literature review (SLR) on longitudinal studies in software engineering

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. During our preliminary research on the related work, we could not find any secondary study (systematic review or mapping study) focusing on empirical longitudinal studies in software engineering. Producing such an SLR will be an opportunity for obtaining a deep understating of the potential, limitations and pitfalls of software evolution experiments, before defining the experimental designs required for answering *RQ3* and *RQ4*. To delimit bias in our SLR, we will follow the guidelines provided in [41].

(EC5) Longitudinal studies on web systems quality

These software evolution studies will be aimed at observing how web systems 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. In other words, *EC5* is expected to answer *RQ3*.

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.

As aforementioned, longitudinal studies in software engineering have a major drawback: software systems (and web systems are not distinct in this facet) 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, and consequently answering *RQ4*, we plan to use irregular time series techniques that have been used, for instance, to predict the stock market volatility [42, 43] and in electronic commerce research [44]. This is an active research area in statistics and new algorithms are being proposed [45, 46]. We firmly believe that its application in the context of software evolution studies is innovative.

IV. CONCLUSION

We have identified a set of research questions that have recently started to be addressed in the scope of the PhD research work of the first author, that basically entail find out (RQ1) what are the most relevant web systems smells, (RQ2) if web systems smells location can be detected automatically, (RQ3) if web systems quality evolution, in terms of maintainability and reliability, is influenced by the presence of web smells, and (RQ4) how can we deal with uneven time-spaced software development data. Those questions were raised based upon a preliminary survey of the state-of-the-art (discussed in section II) and the previous experience on related

research work on our research group. A set of expected contributions were then identified: (EC1) a web smells catalog, (EC2) tools to collect the web smells in that catalog, (EC3) descriptive studies on web systems development, (EC4) a SLR on longitudinal studies in software engineering and, (EC5) longitudinal studies on web systems quality. The rationale and the interdependence of those questions and contributions was described on section III. Table I summarizes the role of each expected contribution, particularly by providing answers to the research questions.

TABLE I. ROLE OF EXPECTED CONTRIBUTIONS

CONTRIBUTION	ROLE
EC1 – Web smells catalog	Answers RQ1
EC2 – **C**ollection tools for the proposed catalog	Answers RQ2 Refines EC1
EC3 – Descriptive studies on web systems development	Validates EC2
EC4 – SLR of longitudinal studies in software engineering	Clarifies RQ3 Clarifies RQ4
EC5 – Longitudinal studies of web systems quality	Answers RQ3 Answers RQ4

We expect that our triangulation-based validation strategy, combining industry surveys, quasi-experiments and peer reviewing in academic *fora* will help us to refine and distill our contributions, so that they can become useful inputs for the Web Engineering community.

REFERENCES

[1] Kappel, G., et al., *Web engineering*. 2006: John Wiley & Sons.

[2] Rossi, G., et al., *Web engineering: modelling and implementing web applications*. 2007: Springer Science & Business Media.

[3] Calero, C., *Handbook of Research on Web Information Systems Quality*. 2008: IGI Global.

[4] Offutt, J., *Quality attributes of web software applications.* IEEE software, 2002. **19**(2): p. 25.

[5] Rio, A., *Modelo Automático de Qualidade para Sítios Web (MSc Thesis)*. 2010.

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

[7] Mens, T., Demeyer, Serge (Eds.), *Software evolution*. 2008: Springer.

[8] Murgia, A., et al. *Empirical study of software quality evolution in open source projects using agile practices*. in *Proc. of the 1st International Symposium on Emerging Trends in Software Metrics*. 2009.

[9] Zhang, H. and S. Kim, *Monitoring software quality evolution for defects.* IEEE software, 2010(4): p. 58-64.

[10] Yu, L. and A. Mishra, *An empirical study of lehman's law on software quality evolution.* Int J Software Informatics, 2013. **7**(3): p. 469-481.

[11] Zhu, T., et al. *Monitoring software quality evolution by analyzing deviation trends of modularity views*. in *Reverse Engineering (WCRE), 2011 18th Working Conference on*. 2011. IEEE.

[12] 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.

[13] 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.

[14] 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.

[15] Bakota, T., R. Ferenc, and T. Gyimothy. *Clone smells in software evolution*. in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. 2007. IEEE.

[16] Olbrich, S., et al., *The evolution and impact of code smells: A case study of two open source systems*, in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. 2009, IEEE Computer Society. p. 390-400.

[17] 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.

[18] Chatzigeorgiou, A. and A. Manakos. *Investigating the evolution of bad smells in object-oriented code*. in *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*. 2010. IEEE.

[19] Peters, R. and A. Zaidman. *Evaluating the lifespan of code smells using software repository mining*. in *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*. 2012. IEEE.

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

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

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

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

[24] Grigera, J., A. Garrido, and J. Rivero, *A Tool for Detecting Bad Usability Smells in an Automatic Way*, in *Web Engineering*, S. Casteleyn, G. Rossi, and M. Winckler, Editors. 2014, Springer International Publishing. p. 490-493.

[25] Distante, D., et al., *Business processes refactoring to improve usability in E-commerce applications.* Electronic Commerce Research, 2014. **14**(4): p. 497-529.

[26] Grigera, J., et al., *Assessing refactorings for usability in e-commerce applications.* Empirical Software Engineering, 2015: p. 1-48.

[27] Almeida, D., et al. *Towards a catalog of usability smells*. in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2015. ACM.

[28] Garrido, A., et al., *Improving accessibility of Web interfaces: refactoring to the rescue.* Universal Access in the Information Society, 2014. **13**(4): p. 387-399.

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

[30] 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.

[31] 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.

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

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

[34] *PHP Mess Detector*. [accessed on 2015-07-31]; Available from: http://phpmd.org/.

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

[36] 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.

[37] Wu, J. and R.C. Holt. *Linker-based program extraction and its uses in studying software evolution*. in *Proceedings of the International Workshop on Foundations of Unanticipated Software Evolution*. 2004.

[38] Wu, J., et al. *Evolution spectrographs: Visualizing punctuated change in software evolution*. in *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of*. 2004. IEEE.

[39] Pessoa, T., Brito e Abreu, F., Pessoa Monteiro, M. and S. Bryton. *An Eclipse Plugin to Support Code Smells Detection*. in *INFORUM'2011*. 2011. Coimbra.
[40] Bryton, S., Brito e Abreu, F. and M. P. Monteiro. *Reducing subjectivity in code smells detection: experimenting with the long method*. in *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*. 2010. IEEE.
[41] Kitchenham, B., *Procedures for performing systematic reviews*. 2004.
[42] Dionisio, A., R. Menezes, and D.A. Mendes, *Mutual information: a measure of dependency for nonlinear time series.* Physica A: Statistical Mechanics and its Applications, 2004. **344**(1): p. 326-329.
[43] 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.
[44] Jank, W. and G. Shmueli, *Functional data analysis in electronic commerce research.* Statistical Science, 2006. **21**(2): p. 155-166.
[45] Eckner, A., *A framework for the analysis of unevenly-spaced time series data.* Preprint. Available at: http://www.eckner.com/papers/unevenly_spaced_time_series_analysis, 2012.
[46] Eckner, A., *Algorithms for unevenly-spaced time series: Moving averages and other rolling operators*. 2012, Working Paper.