Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Survey on Detecting Software Clones in Web Pages for Code Privacy

[1]G. Mohan, [2]Bharathi Anbarasan
[1]Dr.N.G.P Arts and Science College Coimbatore India
[2]Assistant Professor, Head Department of Computer Application, Dr.N.G.P Arts and Science College Coimbatore India

Abstract: ***The survey focus on describing our method for logical clone detection for detecting code theft in web applications with the main aim is to assess the effectiveness and efficiency of the approach, and measure the extent code theft detection opportunities. The research tries web applications from the public domain, for which it did not have expectations about how much duplication and bugs exists, and one web application from the research domain for which it was known that there were many duplicated tag functions. The initial survey focus on static web application, is a basic auction application that can be integrated into other web sites to add simple auctions features. The later research focus on dynamic web applications and third is dynamic web application with different programming language applications, respectively, are both web-based.***

Keywords: Code Theft Detection, Code Clone, Webpages, Duplication, bugs.

I. INTRODUCTION

Code bugs are similar program structures of considerable size and significant similarity. Several studies suggest that as much as 20-50 percent of large software systems consist of bug code. Knowing the location of bugs helps in program understanding and maintenance. Some bugs can be removed with refactoring, by replacing them with function calls or macros, or we can use unconventional Meta level techniques such as Aspect-Oriented Programming or XVCL to avoid the harmful effects of bugs. Bug detection is an active area of research, with a multitude of bug detection techniques been proposed in the literature. One limitation of the current research on code bugs is that it is mostly focused on the fragments of duplicated code (we call them simple bugs), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure. We call these larger granularity similarities structural bugs. Locating structural bugs can help us see the forest from the trees, and have significant value for program understanding, evolution, reuse, and reengineering.

Bug detection tools produce an overwhelming volume of simple bugs' data that is difficult to analyze in order to find useful bugs. This problem prompted different solutions that are related to our idea of detecting structural bugs. Some bug detection approaches target large-granularity bugs such as similar files, without specifying the details of the low-level similarities contained inside them. For example, the authors consider a whole webpage as a "bug" of another page if the two pages are similar beyond a given threshold, computed as the Levenshtein distance. Without the details of the low-level similarities in the large-granularity bugs, it is not always straightforward to take remedial actions such as refactoring or creating generic representation, as these actions require a detailed analysis of low-level similarities. Moreover, Bug Miner goes a step ahead in bug analysis, by looking at the bigger similarity structures consisting of groups of such highly similar files.

II. PROBLEM DEFINITION

To find if automatically produced bug report summaries can help a developer with their work, the system conducted a task-based evaluation that considered the use of summaries for bug report duplicate detection tasks. system found that summaries helped the study participants save time, that there was no evidence that accuracy degraded when summaries were used and that most participants preferred working with summaries to working with original bug reports. Many existing text summarizing approaches exist that could be used to generate summaries of bug reports. Given the strong similarity between bug reports and other conversational data

III. OBJECTIVE

We propose an approach to automatically testing and refactoring modern web applications and detect duplicated pages in dynamic Web sites and on the analysis of both the page structure, implemented by specific sequences of HTML tags, and the displayed content. In addition, for each pair of dynamic pages we also consider the similarity degree of their scripting code. The similarity degree of two pages is computed using different similarity metrics for the different parts of a web page based on the code duplication string edit distance. We have implemented a prototype to automate the clone detection process on web applications developed using technology and used it to validate our approach

IV. METHODOLOGY

IV.1 Reusable Mechanism and Code Consistency

A code bug is a code portion in source files that is identical or similar to another. It is common opinion that code bugs make the source files very hard to modify consistently. Bugs are introduced for various reasons such as lack of a good design, fuzzy requirements, undisciplined maintenance and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste. Thus, code bug detection can effectively support the improvement of the quality of a software system during software maintenance and evolution. The Internet and World Wide Web diffusion are producing a substantial increase in the demand of web sites and web applications. The very short time-to-market of a web application, and the lack of method for developing it, promote an incremental development fashion where new pages are usually obtained reusing (i.e. "cloning") pieces of existing pages without adequate documentation about these code duplications and redundancies. The presence of bugs increase system complexity and the effort to testing and refactoring, maintain and evolve web systems, thus the identification of bugs may reduce the effort devoted to these activities as well as to facilitate the migration to different architectures. This project proposes an approach for detecting bugs in web sites and web applications, obtained tailoring the existing methods to detect bugs in traditional software systems. The approach has been assessed performing analysis on several web sites and web applications.

IV.2 Software Environment and Maintenance

Maintaining software systems is getting more complex and difficult task, as the scale becomes larger. It is generally said that code bug is one of the factors that make software maintenance difficult. This project also develops a maintenance support environment, which visualizes the code bug information and also overcomes the limitation of existing tools.

IV.3 Collaborative Structures and Message Passing

One limitation of the current research on code bugs is that it is mostly focused on the fragments of duplicated code (we call them simple bugs), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure. We call these larger granularity similarities structural bugs. Locating structural bugs can help us see the forest from the trees, and have significant value for program understanding, evolution, reuse, and reengineering. The samples are abstracted from bugs found in Project Collaboration portals developed in industry using ASP and JEE and a PHP-based portal developed in our lab study . Structural bugs are often induced by the application domain design technique or mental templates used by programmers. Similar design solutions are repeatedly applied to solve similar problems

IV.4 Detection granularity structural bugs

Reuse only what is similar, knowing bugs helps in reengineering of legacy systems for reuse. Detection of large-granularity structural bugs becomes particularly useful in the reuse context . While the knowledge of structural bugs is usually evident at the time of their creation, we lack formal means to make the presence of structural bugs visible in software, other than using external documentation or naming conventions. The knowledge of differences among structural bug instances is implicit too, and can be easily lost during subsequent software development and evolution. The limitation of considering only simple bugs is known in the field . The main problem is the huge number of simple bugs typically reported by bug detection tools. There have been a number of attempts to move beyond the raw data of simple bugs. It has been proposed to apply classification, filtering, visualization, and navigation to help the user make sense of the cloning information. Another way is to detect bugs of larger granularity than code fragments. For example, some bug detectors can detect bugd files, while others target detecting purely conceptual similarities using information retrieval methods rather than detecting simple bugs.

IV.5 Template Extraction and Code Stealing

Generally speaking, templates, as a common model for all pages, occur quite fixed as opposed to data values which vary across pages. Finding such a common template requires multiple pages or a single page containing multiple records as input. When multiple pages are given, the extraction target aims at page-wide information. When single pages are given, the extraction target is usually constrained to record wide information, which involves the addition issue of record-boundary detection. Page-level extraction tasks, although do not involve the addition problem of boundary detection, are much more complicated than record-level extraction tasks since more data are concerned. A common technique that is used to find template is alignment: either string or tree alignment. As for the problem of distinguishing template and data, most approaches assume that HTML tags are part of the template, while EXALG considers a general model where word tokens can also be part of the template and tag tokens can also be data. However, EXALG's approach, without explicit use of alignment, produces many accidental equivalent classes, making the reconstruction of the schema not complete.

IV.6 Traditional Classification, Filtering, Visualization

Detection of large-granularity structural bugs becomes particularly useful in the reuse context. While the knowledge of structural bugs is usually evident at the time of their creation, we lack formal means to make the presence of structural bugs visible in software, other than using external documentation or naming conventions. The knowledge of differences among structural bug instances is implicit too, and can be easily lost during subsequent software development and evolution. The limitation of considering only simple bugs is known in the field. The main problem is the huge number of simple bugs typically reported by bug detection tools. There have been a number of attempts to move beyond the raw

data of simple bugs. It has been proposed to apply classification, filtering, visualization, and navigation to help the user make sense of the cloning information. Another way is to detect bugs of larger granularity than code fragments. For example, some bug detectors can detect bugd files, while others target detecting purely conceptual similarities using information retrieval methods rather than detecting simple bugs. The approach described in this paper is also based on the idea of applying a follow-up analysis to simple bugs' data. We observed that at the core of the structural bugs, often there are simple bugs that coexist and relate to each other in certain ways. This observation formed the basis of our work on defining and detecting structural bugs. From this observation, we proposed a technique to detect some specific types of structural bugs from the repeated combinations of collocated simple bugs.

IV.7 Extracting Structured Data from Web Pages

Many web sites contain large sets of pages generated using a common template or layout. For example, Amazon lays out the author, title, comments, etc. in the same way in all its book pages. The values used to generate the pages (e.g., the author, title,...) typically come from a database. In this paper, we study the problem of automatically extracting the database values from such template generated web pages without any learning examples or other similar human input. We formally define a template, and propose a model that describes how values are encoded into pages using a template. We present an algorithm that takes, as input, a set of template-generated pages, deduces the unknown template used to generate the pages, and extracts, as output, the values encoded in the pages. Experimental evaluation on a large number of real input page collections indicates that our algorithm correctly extracts data in most cases.

V. THE RETRIEVAL FRAMEWORK

The W WW distributions are generating a considerable boost in the order of web sites and web applications. A code similarity is a code portion in source files that is matching or similar to another. It is general view that code clones make the source files very hard to modify constantly. Clones are launched for various reasons such as lack of a good design, fuzzy requirements, disorderly protection and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste. Thus, code clone detection can effectively support the improvement of the quality of a software system during software preservation and growth.

The very short time-to-scope of a web application, and the need of method for developing it, support an increase al expansion fashion where new pages are usually obtained reusing (i.e. "cloning") pieces of existing pages without sufficient documentation about these code duplications and redundancies. The presence of clones increase system difficulty and the effort to test, maintain and change web systems, thus the identification of clones may reduce the effort devoted to these activities as well as to facilitate the migration to different architectures.

VI. CONCLUSIONS

Our survey analyses the page structure, implemented by specific sequences of HTML tags, and the content displayed for both dynamic and static pages. Moreover, for a pair of web pages we also consider the similarity degree of their java source. The similarity degree can be adapted and tuned in a simple way for different web applications. We have reported the results of applying our approach and tool in a case study. The results have confirmed that the lack of analysis and design of the Web application has effect on the duplication of the pages. In particular, these results allowed us to identify some common features for the web pages that could be integrated, by deleting the duplications and code clones. Moreover, the clone analysis and bug detection of the pages enabled to acquire information to improve the general quality and conceptual/design of the database of the web application. Indeed, we plan to exploit the results of the code clone analysis method to support web application reengineering activities

REFERENCES

[1] J. Anvik, L. Hiew, and G.C. Murhy, "Coping with an Open Bug Repository," Proc. OOPSLA Workshop Eclipse Technology eXchange, 2005.

[2] J. Anvik, L. Hiew, and G.C. Murph, "Who Should Fix This Bug?" Proc. 28th Int'l Conf. Software Eng. (ICSE '06), 2006.

[3] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate Bug Reports Considered Harmful; Really?" Proc. IEEE 24th Int'l Conf. Software Maintenance (ICSM '08), 2008.

[4] J. Davidson, N. Mohan, and C. Jensen, "Coping with Duplicate Bug Reports in Free/Open Source Software Projects," Proc. IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC '11), 2011.

[5] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," Proc. 29th Int'l Conf. Software Eng. 2007.

[6] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information," Proc. 30th Int'l Conf. Software Eng. (ICSE '08), 2008.

[7] A.J. Ko, B.A. Myers, and D.H. Chau, "A Linguistic Analysis of How People Describe Software Problems," Proc. IEEE Symp. Visual Languages and Human-Centric Computing (VL-HCC '06), 2006.

[8] N. Bettenburg, S. Just, A. Schr€oter, C. Weiss, R. Premraj, and T. Zimmermann, "What Makes a Good Bug Report?" Proc. 16th Int'l Symp. Foundations of Software Eng. (FSE '08), 2008.

[9] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information Needs in Bug Reports: Improving Cooperation between Developers and Users," Proc. ACM Conf. Computer Supported Cooperative Work (CSCW '10), 2010.

[10] R.J. Sandusky and L. Gasser, "Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management," Proc. Int'l ACM SIGGROUP Conf. Supporting Group Work (GROUP '05), 2005.

[11] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams," Proc. ACM Conf. Computer Supported Cooperative Work (CSCW '10), 2010.

[12] R. Lotufo, Z.Malik, andK. Czarnecki, "Modelling the 'Hurried' Bug Report Reading Process to Summarize Bug Reports," Proc. IEEE 28th Int'l Conf. Software Maintenance (ICSM'12), 2012.

[13] S. Mani, R. Catherine, V.S. Sinha, and A. Dubey, "AUSUM: Approach for Unsupervised Bug Report Summarization," Proc. ACM SIGSOFT 20th Int'l Symp. the Foundations of Software Eng. (FSE '12), article 11, 2012.

[14] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the Use of Automated Text Summarization Techniques for Summarizing Source Code," Proc. 17th Working Conf. Reverse Eng. (WCRE '10), pp. 35-44, 2010.

[15] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay Shanker, "Towards Automatically Generating Summary Comments for Java Methods," Proc. 25th Int'l Conf. Automated Software Eng. (ASE '10), pp. 43-52, 2010.

