Clone analysis: Just another quality validation process

Angela Lozano
U. Catholique de Louvain, LLN, Belgium

Minhaz F. Zibran
U. of Saskatchewan Saskatoon, Canada

Michael W. Godfrey
U. of Waterloo, Canada / CWI, Netherlands

Werner Teppe
Amadeus GmbH, Bad Homburg, Germany

***Abstract*— This paper summarizes some discussions at Dagstuhl Seminar #12071 *Software Clone Management Towards Industrial Application* on alternatives envisioned for clone management within the software development process.**

***Keywords*-clone management; software development process; quality assurance**

I. Introduction

Although research on software clones began in the 1990s, most of the results have concentrated on the detection and analysis of clones within source code [11]. Recently, the focus has begun to shift towards problems of clone management [1] and to the study of clones in artifacts beyond source code, such as requirements documentation [2], models [3], bug reports [4, 5], and between libraries and the applications that use them [6]. This paper proposes alternatives envisioned for clone management within the software development process. These alternatives are based on the observation that cloning of various kinds of development artifacts could have demonstrable effects of the system's quality, and so the detection, analysis, assessment, and management of software artifact clones should be a standard part of QA at each step of the development process.

II. Requirements

By their nature, requirements documents are often full of near duplication of descriptions of desired functionality, as similar ideas are described in slightly different situations, and often in great detail. How best to model near-duplicates in use-cases, for example, is a well-known problem without a clear solution. Clones in requirements documents may lead to duplicate implementation of very similar features if, for example, similar use cases are given to different development teams to implement. In turn, this may result in semantic code clones, or even code segments with very similar structure. Therefore, the detection of clones at requirements could help avoiding clones in code, or to identify semantic clones, which in turn could help to differentiate features from sub-features.

III. Design

Clones in design models, especially in model-driven development, may also lead to clones in the source code. Thus, the upfront detection of clones in design models might help to reconsider architectural choices, and result in a leaner, more abstract and essential design with fewer code clones. And when there might not be an appropriate abstraction to encapsulate the commonalities, the mere identification of the clones can help to promote awareness and provide traceability for the conceptual links.

IV. Implementation

The best-known software artifact clones are those found in source code. There are two common ways that clone detection is used during the development: management of clones themselves to minimize their negative impact, and using code clones as a kind of code oracle.

A. Minimizing clones' negative impact

The success of clone management depends on a timely detection of clones and effective timely feedback. An alternative to prevent copy and paste behavior can be allowing the IDE to automatically log keyboard combinations used for copy and paste [7], and to track the source and target snippets. However, to minimize the number of cloned snippets it is also necessary to integrate code similarity-based clone detection to the Quality Assurance (QA) process, given that not all copy & paste operations result in cloned code and that there might be accidental clones, which are not created by deliberate copy-paste activities.

Clone detection capability can be integrated at the central code repository such as SVN or CVS. Then clone detection can be performed as a part of periodic code review process to inform QA, and developers can take necessary step for clone removal, refactoring, or documentation. The result of clone detection can be reported in parallel to other QA reports such as test coverage. A proactive approach could initiate incremental clone detection at the time of the developer's commit (or before commit, if possible), and if the checked-in code is found to be a clone, an immediate feedback can be provided to the developer. A more proactive technique may be the incorporation of a clone search facility in the client-side IDEs [8], which can enable the developer search for clones of a particular code segment of interest, and take necessary actions upfront before attempting to check-in to the central repository.

The feedback from clone detection should be actionable, which might suggest for clone refactoring or documentation[1] [2] of the rationale for leaving such clones. This documentation is crucial for the improvement of the clone management process because it states which clones cannot be removed/refactored (and why), and which clones are known design exceptions [9], as well as the context to consider one

[1] We suggest tagging be done as metadata embedded in the code.
[2] VisualStudio and Eclipse now support plug-in tools that support basic clone tagging.

or the other (e.g., platform differences or domain concepts might be relevant if the clones are in a product line architecture). It is important to improve the level of feedback from the clone detection tools so that they can support deciding easily to what extent a refactoring is feasible. Simple heuristics that consider the number of instances, length of the clone, distance among clone instances, differences across clone instances, and types of those differences could facilitate the refactoring choice.

Although immediate feedback seem to be a good idea to minimize code cloning, it may raise usability issues by popping too many notifications that might hinder the developer's normal flow of work ("MS-Clippy"-like behavior). Therefore, care must be taken to keep the notifications passive enough, while still making them visible to the developers.

B. Maximizing clones' positive impact: clones as an oracle

Cloned code can also be used as an oracle. For instance, clone detection can be applied to identify crosscutting concerns [10, 11], or to locate exemplars of API usage [12]. Another possible use of clones as oracle consists on using fragments of code that the developer is writing to find places in the code that implement a similar functionality: this could not only provide exemplars and ideas for refactoring, but also can reduce the effort by avoiding duplication of similar functionality.

V. TESTING

It is important to check to what extent clone refactoring can be trusted, in other words, should the code that has had clone refactoring be tested again? Indeed, the expectation of complete automation in the refactoring of code clones can be too ambitious due to the diversity, dispersion, and dependencies of those clones with respect to different portions of the code base. Therefore, automated test case generation or adaptation may be necessary to validate that the manual or semi-automated refactoring of clones really did not alter the program behavior [13].

Another open issue to consider is the relation between cloned code and their tests, i.e. to what extent having a clone shared between two source code entities imply that their corresponding tests should be cloned? Does it worth to detect clones inside test code, or can clones help to understand the dependencies between tests, and therefore across their corresponding source code entities?

VI. DEPLOYMENT

At the deployment phase, clone detection can be applied to analyze code leakage: where the code is going, not only for intellectual property management but also to analyze how other people are modifying the code. In that sense it would be interesting to see research in transitive code search based on code snippets and analysis of the evolution of such.

VII. MAINTENANCE

It would also be important to trace and document as soon as a clone group requires consistent changes because they are likely to cause bugs. Therefore, Quality Assurance should validate the changes in code hosting those clones.

VIII. SUMMARY

Clone detection of source code is a mature technology that can complement the Quality Assurance process [14]. However, it is worthwhile to further investigate the costs of avoiding clones, of detecting and managing clones vs. costs of double development, and of fixing bugs inside vs. outside clones. And we believe that since clones can, in principle, be detected within other development artifacts and at other stages of development, it is also worthwhile to consider detecting, analyzing, and managing these clones as well.

ACKNOWLEDGMENTS

Angela Lozano is funded on an FNRS-FRFC project.

REFERENCES

[1] J. Harder and N. Göde. 2010. Quo vadis, clone management?. In Proc. of the Int'l Workshop on Software Clones (IWSC '10), 85-86.

[2] E. Juergens, F. Deissenboeck, M. Feilkas, B. Hummel, B. Schaetz, S. Wagner, C. Domann, and J. Streit. 2010. Can clone detection support quality assessments of requirements specifications?. In Proc. of the Int'l Conf. on Softw. Engg. (ICSE '10), 79-88.

[3] F. Deissenboeck, B. Hummel, E. Jürgens, B. Schätz, S. Wagner, J.F. Girard, and S. Teuchert. 2008. Clone detection in automotive model-based development. In Proc. of the Int'l Conf. on Softw. Engg. (ICSE '08), 603-612.

[4] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In Proc. of the Int'l Conf. on Softw. Engg. (ICSE '08), 461-470.

[5] Y. Song, X. Wang, T. Xie, L. Zhang, and H. Mei. 2010. JDF: detecting duplicate bug reports in Jazz. In Proc. of the Int'l Conf. on Softw. Engg. (ICSE '10), 315-316.

[6] C. Sun, S.-C. Khoo, and S. J. Zhang. 2011. Graph-based detection of library API imitations. . In Proc. of the Int'l Conf. on Softw. Engg. (ICSM '11), 183-192.

[7] F. Jacob, D. Hou, and P. Jablonski. 2010. Actively comparing clones inside the code editor. In Proc. of the Int'l Workshop on Softw. Clones (IWSC '10), 9-16.

[8] M. F. Zibran and C. K. Roy. 2012. IDE-based Real-time Focused Search for Near-miss Clones. In proc. of the Symposium on Applied Computing, 08 pages (to appear).

[9] C. J. Kapser and M. W. Godfrey, 2008, "Cloning considered harmful" considered harmful: Patterns of cloning in software, *Empirical Software Engineering*, vol. 13, no. 6.

[10] D. Shepherd, E. Gibson, and L.L. Pollock. 2004. Design and Evaluation of an Automated Aspect Mining Tool. In Proc. Int'l Conf. Softw. Eng. Research and Practice (SERP '04), 601-607.

[11] M. Bruntink, A. v. Deursen, R. v. Engelen, and Tom Tourwe. 2005. On the Use of Clone Detection for Identifying Crosscutting Concern Code. IEEE Trans. Softw. Eng. 31, 10, 804-818.

[12] R. Cottrell, R. J. Walker, and J. Denzinger. 2008. Semi-automating small-scale source code reuse via structural correspondence. In Proc. of the Int'l Symposium on Foundations of Softw. Engg. (SIGSOFT '08/FSE-16), 214-225.

[13] M. F. Zibran and C. K. Roy. 2011. Towards Flexible Code Clone Detection, Management and Refactoring in IDE. In proc. of the Int'l Workshop on Software Clones (IWSC'11), 75-76.

[14] M. F. Zibran and C. K. Roy. 2012. The Road to Software Clone Management: A Survey. Technical Report 2012-03, Department of Computer Science, University of Saskatchewan, Canada, 1-62.