Impact of Collaboration on Structural Software Quality

ABSTRACT

The structural quality of a codebase is a key determining factor in the software's total cost of ownership, yet it is notoriously difficult to measure or predict. In this doctoral research we leverage the power of open source repositories to understand the factors that influence structural quality (and by extension fault-proneness) in the context of the patterns of collaborative behaviour exhibited by contributors. The objective is to further our understanding of how such behaviour impacts structural quality with the end goal being to inform management decision making across the industry in the pursuit of better software engineering practices.

1. INTRODUCTION

A software metric is the quantitative measure of the degree to which a component, system, or process possesses a given characteristic or attribute. Software metrics embody an empirical approach to software engineering and are primarily designed to assist in making assessments of software artifacts and development processes, in the process guiding engineers and project managers in their decision making. If used appropriately, software metrics can lead to a significant reduction in costs of the overall implementation and maintenance of the final software product.

The research community is highly active in the field of software metrics, awash with publications covering topics as diverse as the usage of existing 'classical' metrics, the formulation of new metrics, through to various metric-based process models.

Despite this, metrics adoption has remained on the margins of software engineering [1]. We propose that we can effectively leverage the wealth of empirical models and raw data to establishing 'simple truths' (for example '*more authors do not impact structural quality*') to inform management decision making and bridge the gap between the excellent academic work in this field and the software engineering industry.

2. RELATED WORK

2.1 Software metrics

The study and application of software metrics dates back to the mid-1960's when the primitive Lines of Code metric was routinely used as the basis for measuring software development productivity (developer LoC per month) and quality (defects per KLoC). In 1971 Akiyama proposed the use of metrics for software quality prediction proposing a regression-based model for module defect density (number of defects per line of code) where line of code was used as a crude indicator of complexity – an early attempt to extract an objective measure of software quality through the analysis of observables of the system.

With the increasing adoption of Object-Oriented (OO) programming languages in the nineties, research in software metrics took a significant step forward. Chidamber and Kemerer argued that Object-Orientation, as the most prominent advance in software development, and with yet to be established practices, necessitated measures that could guide organizations to its successful adoption. This fact, coupled with criticisms of existing metrics suites, saw the development of the Chidamber and Kemerer (CK) metrics suite [2].

2.2 Predication Models

In 1993 Li and Henry performed a statistical analysis on two commercial software systems using a regression model to prove that metrics (including the CK metrics suite) can be used as a predictor of maintainability (as defined as the number of changed lines in a class in its maintenance history). [3]

Basili et al, motivated by the desire to leverage software metrics to provide guidance to the areas of a system where testing efforts are best spent, built on Li and Henry's research to establish the utility of the Chidamber and Kemerer software metrics suite to as a predicator of fault prone software classes. This was achieved through the diligent assembling of eight software development teams and a thorough regression analysis to establish relationships between OO metrics and observed defects. [4]. Many more such studies were to follow.

2.3 Metrics – Collaborations and other Impacting factors

One of the elements to the software development methodology is its approach to collaboration. There is a dearth of solid empirical studies assessing the impact of collaborative software development on structural code quality, there have been some attempts to do just that - albeit, again with contradicting results [5] [6].

Assessment of the impact on pair programming, for example, is fairly disjointed, often contradictory, and in many cases rely heavily on anecdotal evidence. Williams et al find that the collaborative pair programming approach yields favourable results when compared with solo development - as manifested through lower defect counts [7]. Conversely, Hulkko and Abrahamsson find that pair programming results in lower adherence to coding standards and has no impact on defect densities [8].

3. RESEARCH PROBLEM

3.1 Collaborative Contributor Behaviour Across Open Source Repositories

We are interested in furthering our understanding around patterns of contributor participation within an Open Source community. In particular, we are looking to understand the proportion of projects with a single contributor versus multiple contributors, understanding how likely contributors are to encounter one another in future projects, and the factors that cause these partnerships to repeat (e.g. the same programming language or category of project).

This research differs from the vast majority of Open Source community studies by virtue of the fact that we wish to study traversal of users throughout an entire open source community. This means that data must be gathered for every single project within that community. This has implications for data storage format and speed of execution. Furthermore, different repository types (e.g. SubVersion, Mercurial, and GIT) introduces an additional level of complexity.

3.2 Effect of Collaboration on the Structural Quality of a Codebase

The second key area of interest in this research is the impact that collaboration can have on the structural quality of a codebase. Project teams usually comprise of several developers which contribute to a single set of source files – indeed it is not uncommon for several developers to collaborate on a single source file. In fact virtually all source control systems have been developed with conflict resolution features to facilitate multiple and simultaneous contributions to a single source file. Our interest lies in how multiple authors can affect the structural quality of a codebase.

Second, third, or fourth developers (or collaborators) modifying an existing class file may not be as fully acquainted as the original author with the current functionality in a class, its hierarchy, or common utility classes. This may lead to additional complexity over and above what may be necessary for the functional change required. Furthermore, additional developers may not possess the same level of ownership as the original author, again leading to unnecessary complexity and a consequent degradation in quality. Conversely collaborators also offer an opportunity for the project to benefit from additional experience and expertise which may cast a fresh perspective, or indeed a critical eye, on the codebase – refactoring where necessary and reducing complexity.

Without studying a statistically significant sample of projects, it is difficult to ascertain which of these competing influences would be dominant one affecting the progression of complexity metrics throughout the evolution of projects.

3.3 Predictive Models

The third key aspect of this research is intrinsically related to the first and second points – namely to derive, from our understanding of the effect of collaboration on the structural quality, a mathematical model which predicts aspects of structural quality based on key factors such as contributor count or the number of revisions in project.

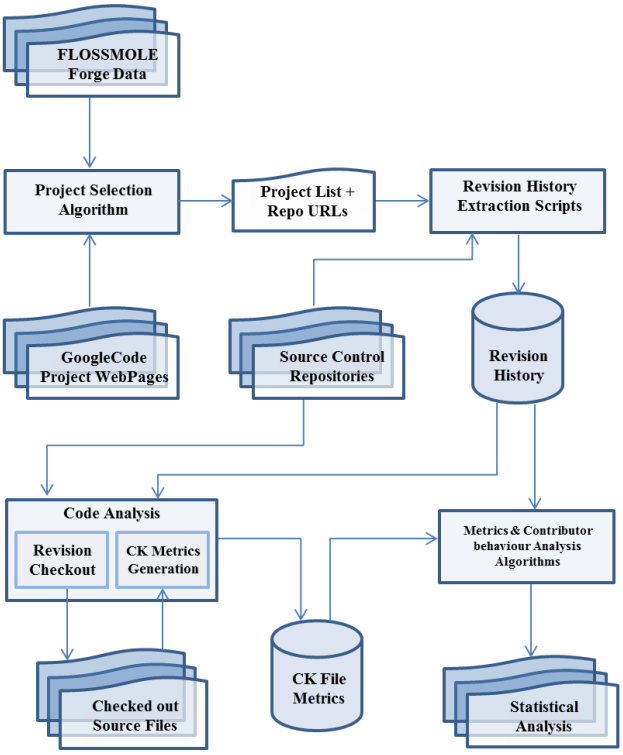
4. PROGRESS

At the outset of this research, it was necessary to select an open source community to be the focus of our studies. GoogleCode was selected for its popularity and high level of Java adoption rates. It was considered that Java projects would be the most preferable to study due to a number of reasons including highly OO nature, wealth of analysis tools and relevance to practitioners. The popular metrics suite proposed by Chidamber and Kemerer is both well understood and has a significant supporting body of research and was chosen to measure structural quality across codebases. A comprehensive tool-chain was developed, as outlined in figure 1, to extract and analyse the data pertinent to this resesarch.

5. EXPECTIONS OF CONFERENCE

A vast amount of data has been extracted and a significant amount of analysis has been conducted, leading to some very interesting observations that we believe further the understanding of the impact of collaborative software development on the structural quality of the codebase. We would be extremely keen on sharing that analysis with our peers and soliciting feedback on how to ensure that this research can make the most impact to the practitioners in the field.

Figure 1 Tool-chain to extract and analyse revision based metrics.



6. REFERENCES

[1] Fenton, N. E., & Neil, M. 2000. Software metrics: roadmap. In Proceedings of the Conference on the Future of Software Engineering (pp. 357-370). ACM

[2] Chidamber, S. R, & Kemerer, C. F. 1994. A Metrics Suite for Object Oriented Design, IEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994, pp. 476-493

[3] Li, W., Henry, S. 1993. Object-oriented metrics that predict maintainability. Journal of systems and software, 23(2), 111-122.

[4] Basili, V. R., & Briand, L. C., & Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. Software Engineering, IEEE Transactions on, 22(10), 751-761.

[5] Ciolkowski, M., & Schlemmer, M. (2002). Experiences with a case study on pair programming. In Workshop on Empirical Studies in Software Engineering.

[6] Madeyski, L. (2006). The impact of pair programming and test-driven development on package dependencies in object-oriented design—an experiment. In Product-Focused Software Process Improvement (pp. 278-289). Springer Berlin Heidelberg.

[7] Laurie Williams, Bob Kessler (2000). The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education, Proceedings of the 13th Conference on Software Engineering Education & Training, p.59, March 06-08

[8] Hanna, and Pekka Abrahamsson. "A multiple case study on the impact of pair programming on product quality." Proceedings of the 27th international conference on Software engineering. ACM, 2005

OpenSym '14 , Aug 27-29 2014, Berlin, Germany
ACM 978-1-4503-3016-9/14/08.
http://dx.doi.org/10.1145/2641580.26416