Proactive Software Complexity Assessment

Vard Antinyan

Presentation:
7th of November 2017, 13:00
Room Omega, Jupiter Building
Hörselgången 5, 417 56, Gothenburg
University of Gothenburg

UNIVERSITY OF GOTHENBURG

Main adviser: Prof. Miroslaw Staron
Second advisor: Assoc. Prof. Anna Sandberg
Examiner: Prof. Jörgen Hansson

Opponent: Prof. Tracy Hall

Committee:
Prof. Luigi Lavazza
Prof. Dag Søberg
Prof. Daniel Sundmark

Abstract

Large software development companies primarily deliver value to their customers by continuously enhancing the functionality of their products. Continuously developing software for customers insures the enduring success of a company. In continuous development, however, software complexity tends to increase gradually, the consequence being deteriorating maintainability over time. During short periods of time, the gradual complexity increase is insignificant, but over longer periods of time, complexity can develop to an unconceivable extent, such that maintenance is no longer profitable. Thus, proactive complexity assessment methods are required to prevent the gradual growth of complexity and instead build quality into developed software.

Many studies have been conducted to delineate methods for complexity assessment. These focus on three main areas: 1) the landscape of complexity, i.e., the source of the complexity; 2) the options for complexity assessment, i.e., how complexity can be measured and whether the results of assessment reflects reality; and 3) the practicality of using complexity assessment methods, i.e., the successful integration and use of assessment methods in continuous software development.

Partial successes were achieved in all three areas. Firstly, it is clear that complexity is understood in terms of its consequences, such as spent time or resources, rather than in terms of its structure per se, such as software characteristics. Consequently, current complexity measures only assess isolated aspects of complexity and fail to capture its entirety. Finally, it is also clear that existing complexity assessment methods are used for isolated activities (e.g., defect and maintainability predictions) and not for integrated decision support (e.g., continuous maintainability enhancement and defect prevention).

This thesis presents 14 new findings across these three areas. The key findings are that: 1) Complexity increases maintenance time multifold when software size is constant. This consequential effect is mostly due to a few software characteristics, and whilst other software characteristics are essential for software development, they have an insignificant effect on complexity growth; 2) Two methods are proposed for complexity assessment. The first is for source code, which represents a combination of existing complexity measures to indicate deteriorating areas of code. The second is for textual requirements, which represents new complexity measures that can indicate the inflow of poorly specified requirements; 3) Both methods were developed based on two critical factors: (i) the accuracy of assessment, and (ii) the simplicity of interpretation. The methods were integrated into practitioners' working environments to allow proactive complexity assessment and prevention of deteriorating maintainability.

In addition, several additional key observations were made, primarily that the focus should be in creating more sophisticated software complexity measures based on empirical data indicative of the code characteristics that most influence complexity. It is desirable, therefore, to integrate such complexity assessment measures into the practitioners' working environments to ensure that complexity is assessed and managed proactively. This would allow quality to be built into the product rather than having to conduct separate, post-release refactoring activities.

Keywords: complexity, metric, measure, code, requirement, software quality, technical risk, technical debt, continuous integration, agile development

Technical Report No 143D
ISBN 978-91-982237-2-9