Proceedings of the Federated Conference on Computer Science and Information Systems pp. 1207–1210
DOI: 10.15439/2017F295
ISSN 2300-5963 ACSIS, Vol. 11

Measuring dimensions of Software Engineering projects' success in Academic context

Rafał Włodarski
Lodz University of Technology
ul.Wolczanska 215, 90-924 Lodz, Poland
Email: r.wlodarski89@gmail.com

Aneta Poniszewska-Marańda
Lodz University of Technology
ul.Wolczanska 215, 90-924 Lodz, Poland
Email: aneta.poniszewska-maranda@p.lodz.pl

***Abstract*—The notion of success is unsubstantial, complex and domain-specific. Software companies have been exploring its different aspects and aiming to put forward measures to capture and evaluate them. In this paper three main dimensions of success have been elicited based on previous industrial studies: project quality, project efficiency along with social factors and stakeholder's satisfaction. By investigation of their assessment criteria in commercial context a set of metrics and measures was determined and adapted to provide a structured evaluation approach for projects developed in academic setting. Professionalizing teaching and assessment process is an attempt to close a gap between workforce's expectations towards new graduates and the outcomes of their university education.**

I. Introduction

GRADING system of students' work was born at the most prestigious universities in the world and dates back to eighteenth century: the first grades were issued at Yale in 1785 [1] while the concept of grading students' work quantitatively was implemented by the University of Cambridge in 1792 [2]. These grading systems were straightforward and measured on a given scale the learning outcomes and extent of assimilation of knowledge. Although a shift to project-based learning has been made ever since, no wide-known elaborate framework that would privilege assessing different dimensions of success of students' projects has been published.

While much research has addressed the definition and measurement of success in industrial software engineering undertakings [3], [4], [5], so far little attention has been paid to the same problem in academic context.

This article explores how assessment criteria used for IT deliverables can be translated into an academic grading context and aims to introduce a set of metrics, and measures for evaluation of projects developed by students.

A thorough literature review reveals three main criteria of project success [3], [5], [6], [7], [9]:

- project quality – source code and product quality,
- project efficiency – resources utilization and productivity of the team,
- social factors and stakeholders' satisfaction – team cohesion, morale; students' satisfaction and learning outcomes.

The presented paper is structured as follows: applicability of the proposed evaluation framework is discussed in section two, while the three dimensions and their measurement methods are detailed in sections three, four and five respectively.

II. Academic setting

While commercial projects are carried out according to the rules of a certain software development approach – ranging from plan-driven (Waterfall), through evolutionary (Spiral) to iterative and agile (Scrum, XP), academic projects do not always adhere to any formal processes. They could however, follow certain phases of development as software projects do:

- *requirements engineering*, usually in the form of functionality imposed by the lecturer that needs to be analyzed by students,
- *design*, when architecture and the technology stack of the project are defined,
- *implementation*, in form of coding that involves the most effort relative to the rest of the project,
- *testing*, which allows students to internally verify adherence to functional requirements and encompasses tutor's evaluation.

The project life cycle in a commercial context acts as a structure that allows for coordination and management, efficient allocation of resources, risk assessment and mitigation and a common and shared vocabulary [6], [7]. While the evaluation framework provided in this article is not tailor made for a particular development approach, it requires application of distinct project phases.

III. Project quality

Paul Ralph and Paul Kelly [3] investigate the dimensions of software engineering success, yield 11 different themes with project being the most important and central concept. In this article two types of quality are explored:

Internal quality aspects vary depending on the chosen software development methodology and include [8]:

- requirements documentation – in the form of user stories for Agile approaches and written communication for traditional ones,
- detailed design documents – applicable in traditional models,
- the code and adherence to continuous integration practice – also comprised in Agile models.

External quality aspects are observable outside the development team and are assessed by client in commercial projects and professor in academic ones. They encompass both:

- documentation materials – presentation, user manual, installation guide [8] and
- software's characteristics – adherence to functional requirements, user-friendliness, robustness and reliability.

These software quality attributes account for four out of six areas covered by the quality model proposed by ISO/IEC 9126.

A. Internal quality

As mentioned, there are two major factors that influence the internal quality of a project: source code and extent of adherence to continuous integration practices. Studies have shown that more complex code, or "spaghetti code", produced by undergraduate students in particular, is difficult to understand, more prone to produce errors than a well-designed and coded module [12].

1) Code complexity and size measure: Cyclomatic complexity (CC) is a measure of complexity of a program and is determined by counting the number of decisions (linearly independent paths) made in a given source code. It is a commonly used metric in the industry and according to McCabe Software Company [11] it meets three qualities of a good complexity measure. It is:

- descriptive, as it objectively measures something – decision logic in the case of CC,
- predictive, as it correlates with something important – errors and maintenance effort,
- prescriptive as it guides risk reduction – testing and improvement.

While Cyclomatic Complexity proves to be a good indication of quality, the Software Assurance Technology Center (SATC) at NASA [13] found that the most effective evaluation is a combination of size and complexity. Source code of significant size and high complexity bears very low reliability. Likewise, software with low size and high complexity, as it tends to be written in a very terse fashion, renders the source code difficult to change and maintain [14].

An additional success criterion of students' projects is its quality of Object Oriented Design. This is a core of any computer science related course and a paradigm that is applied to a majority of students' future undertakings [36]. As suggested by SATC [13] a pertinent evaluation is the Weighted Methods per Class, introduced by Chidamber and Kemerer [15]. WMC is the sum of the complexity of the methods of a class and a predictor of how much time and effort is required to develop and maintain the class [13], which is particularly important for students as they share the code with other team members.

2) Continuous integration: Continuous Integration is a concept first introduced by Booch [17] whose aim was to avoid pitfalls while merging code from different programmers and thus reduce the work and time effort required for the project. Though initially employed only in a commercial setting, CI has gained popularity in the students' undertakings, as effective teamwork requires use of a version control system on regular basis. In a Technical University of Munich study [9], continuous integration was perceived as beneficial according to 63% of a sample of 122 students and only 13% did not agree with that statement.

To measure adherence to this practice in a large-scale agile transformation in multiple companies, Olszewska et al. [18] propose a metric called *Pacemaker: Commit pulse* by counting the average number of days between commits and aiming at keeping it as low as possible. Evenly distributed workload and regular merges reduce the complexity of integration and decrease the pressure related to issues of meeting a deadline, which is a frequent challenge in students' undertakings.

The metric is applicable to both plan-driven and agile settings as data can be collected and evaluated with respect to any significant time frame, e.g. sprint, month, or semester.

The three aforementioned metrics can be easily elicited from source code written in any major programming language [10], [16] and give meaningful insight into a product's internal quality.

The calculation method of *Cyclomatic Complexity, CC* is given by:

$$CC = \sum_{i=1}^{n} E_i - \sum_{j=1}^{nm} N_j + 1,$$

where: *N*: number of nodes – logic branch point, such as *if*, *while*, *do*, case statements in *switch*, *E*: number of edges – an edge represents a line between nodes.

The calculation method of *Weighted Method per Class, WMC* is given by:

$$WMC = \sum_{i=1}^{n} c_i,$$

where: *C*: given class, *M*: methods defined in a class, c: complexity of a method.

The calculation method of *Pacemaker Commit Pulse, PCP* is given by:

$$PCP = \sum_{i=1,j=i+1}^{n,m=n-1} (C_j - C_i)/N$$

where: C_i: timestamp of a commit, C_j: timestamp of the following commit, *N*: total number of commits in a given period.

B. External quality

While in commercial environment the testing phase is an elaborate process carried out by dedicated personnel and lasting weeks, in the academic setting, it involves mostly a rudimentary test campaign performed by students to ensure that the requested functionality is in place before assignment completion. The tutor performs additional ad-hoc tests to evaluate functional conformance and judge his overall satisfaction level with the results. What additionally differs between the two contexts is that the latter often lacks a framework for defects categorization and tracking. In this article a minimal formal process for testing is proposed so that the external quality factor can be incorporated into the project success evaluation.

1) Orthogonal Defect Classification: Businesses provide considerable effort to analyse and improve their software development life cycle process; one of early adopters of a formalized approach was IBM. Chillarege et al. [32] introduce an Orthogonal Defect Classification (ODC), a conceptual framework using semantic information from defects to extract cause-effect relationships in the development process. It involves classifying defects according to different attributes at two points in time:

- once by a submitter, who evaluates the functional correctness of software,
- once the defect has been fixed or responded to by a technical team member, who identifies the type of problem origin.

In order to keep the classification simple and the overhead added minimal, only the first phase is retained as part of the proposed framework. When a defect is detected, it needs to be classified according to three attributes [33]:

- *Activity*, which refers to the actual process step (code inspection, function test etc.) that was being performed at the time the defect was discovered.
- *Trigger*, which describes the environment or condition that had to exist to expose the defect.
- *Impact*, which refers to either perceived or actual impact on the customer.

IV. Project Efficiency

From a classical project-management point of view the underpinning of a process's success is respect of underlying budget and time constraints. Indeed, a systematic literature review of 148 papers published between 1991 and 2008 [27] revealed that Effort and Productivity were defined as success indicators in 63% of studies of process improvement initiatives. In the simplest terms, effort is the time spent by the team during the development process and productivity is its output size in terms of KLOC (*kilo lines of code*) [28]. In this paper, more nuanced ways of evaluating project efficiency and team productivity are investigated as motivation to produce significant amounts of code can be detrimental to the quality and is not representative of delivered software value.

A. Defining measurements units

An antagonistic approach of measuring functional size was introduced by Middleton et al. [30], who invented a method called *Function Point Analysis (FPA)*. Ever since, multiple recognized standards and public specifications were defined based on the notion of *Functional Points*. In parallel, other size-based estimation models emerged, such as *Use Case Points* [31] or story-based estimation in Agile techniques. While complex frameworks might yield precise results, they require experience that students lack and are frequently time-consuming.

Function Point will thus be understood as an informed high-level estimation of an underlying piece of functionality (known as *Early Function Point Analysis*). Professors are encouraged to provide the estimates along with the specification of projects or assist and share their knowledge with students if the estimation process is within the assignment scope.

In order to adopt a reference unit, students need to track the time spent on the project. Abrahamsson [29] suggests collecting effort for each defined task with a precision of 1 minute using paper/pen and predefined excel-sheet as the primary collection tools. While feasible in a commercial setting, students are required to estimate their effort with a precision of 15 minutes. This number is more suitable to a working environment that is characterized by irregular efforts and little attention to one's own time tracking.

B. Productivity and efficiency metrics

The definition of effort and reference unit sets the ground for the evaluation of project efficiency through the application of multiple metrics. The first one, suggested by Olszewska et al. [18] is *Hustle Metric: Functionality/Time spent*, which measures how much functionality can be delivered with respect to a certain work effort. It is calculated by dividing function points of a task, module or even an entire project by total amount of implementation time spent by the students.

The calculation of *Hustle Metric: Functionality/Time spent, HM* is given by:

$$HM = \sum_{i=1}^{n} F_{pi} / \sum_{i=1}^{n} T_i,$$

where: F_{pi}: number of functional points of an artefact (task, module etc.) considered, T_i: overall time spent by the team implementing the considered functionality.

The evaluated efficiency facet of this metric is overall global productivity of the team.

Processing interval is calculated as a subtraction of a timestamp when the feature is fully implemented and uploaded to a repository (*Tship*) and a timestamp when the feature is accepted for implementation (*Tacc*). This metric mirrors the efficiency of the implementation process as one can monitor the technological or functional cumbersomeness of a certain feature and team's capability to tackle encountered problems.

The calculation of *Processing Interval: Lead-time per feature, PI* is given by:

$$PI = T_{ship} - T_{acc},$$

where: T_{ship}: timestamp when the feature is fully implemented and uploaded to a repository, T_{acc}: timestamp when the feature is accepted for implementation.

A related metric was tracked at Timberline Inc. [29] – *Work In Progress (WIP)* – defined as a sum of function points of features that are currently under development. As observed in the study, large amounts of work in progress can translate into many unidentified defects, which would be discovered eventually. It can also mitigate a potential risk of another harmful phenomenon: cherry-picking features that are most interesting to the team or perceived as the simplest ones.

The calculation of *Work In Progress, WIP* is given by:

$$WIP = \sum_{i=1}^{n} F_{pi},$$

where: F_{pi}: function points of a task currently in progress.

V. Social factors and stakeholders' satisfaction

In their study, Hoegl and Gemuenden [19] express three principle factors that influence the success of innovative projects: team performance, teamwork quality, personal success.

Teamwork quality is a measure of conditions of collaboration in teams; according to Hoegl and Gemuenden [19] it consists of six facets: communication, coordination, balance of member contributions, mutual support, effort and cohesion.

Team cohesion is defined as the "shared bond that drives team members to stay together and to want to work together" [22]. As stated in [20] cohesion is highly correlated with project success, critical for team effectiveness [21], and leads to increased communication and knowledge sharing [22].

A final dimension of project success is satisfaction and personal accomplishment of its participants. Although it might not be apparent to students, it is their learning outcomes and improved skills that are of paramount importance in that subject matter. Employers emphasize that both technical and soft skills are essentials for implementation of successful software projects. A study by Begel et al. [34] on struggles of new college graduates in their first development job at Microsoft finds that they have difficulties in communication, collaboration and cognition areas; Brechner [35] suggests they should participate in dedicated courses in Design Analysis and Quality Code as part of their education.

VI. Conclusions and future work

This paper provides professors and faculty members a framework for evaluation of Software Engineering projects' success. Its different dimensions are elicited and further divided into sub facets so that they can be addressed with a specific metric or measure. Exploring assessment criteria used for IT deliverables in commercial setting helps professionalize computer engineers' university education and more aptly prepare the graduates to join today's workforce.

Future work will consist of application of the framework to University courses so that students' perception can be taken into consideration and possibly some of the measures adjusted. By employing the proposed evaluation scheme, roadblocks can be identified along with supporting tools to minimize the potential overhead.

References

[1] G. Pierson, "C. Undergraduate Studies: Yale College", Yale Book of Numbers. Historical Statistics of the College and University 1701-1976, New Haven: Yale Office of Institutional Research, 1983

[2] N. Postman, "Technopoly The Surrender of Culture to Technology", New York: Alfred A. Knopf, 1992

[3] P. Ralph, and P. Kelly, "The Dimensions of Software Engineering Success", 2014

[4] E. Kupiainen and M. V. MaÌĹntylaÌĹ and J. Itkonen, "Using metrics in Agile and Lean Software Development âĂŞ A systematic literature review of industrial studies", 2015

[5] M. Unterkalmsteiner and T. Gorschek and A. K. M. Moinul Islam, "Evaluation and Measurement of Software Process Improvement – A Systematic Literature Review", 2011

[6] D. Dalcher and O. Benediktsson and H. Thorbergsson, "Development Life Cycle Management: A Multiproject Experiment", 2005

[7] D. Dalcher, "Life Cycle Design and Management", 2002

[8] F. Macias and M. Holcombe and M. Gheorghe, "A Formal Experiment Comparing Extreme Programming with Traditional Software Construction, 2003

[9] B. Bruegge and S. Krusche and L. Alperowitz, "Software Engineering Project Courses with Industrial Clients", 2015

[10] Z. Naboulsi, "Code Metrics – Cyclomatic Complexity", MSDN Ultimate Visual Studio Tips and Tricks Blog, https://blogs.msdn.microsoft.com/zainnab/2011/05/17/code-metrics-cyclomatic-complexity, 2017

[11] McCabe Associates, "Integrated Quality" as part of CS699 Professional Seminar in Computer Science, 1999

[12] B. Kitchenham and S. L. Pfleeger, "Software Quality: The Elusive Target', IEEE Software, 1996

[13] L. Rosenberg and T. Hammer, "Software metrics and reliability", NASA GSFC, 1998

[14] L. Rosenberg and T. Hammer, "Metrics for Quality Assurance and Risk Assessment", Proceedings of 11th International Software Quality Week, USA, 1998

[15] C. F. Kemerer and S. R. Chidamber, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering 1994

[16] Java Code Geeks, "Java Tools: Source Code Optimization and Analysis", https://www.javacodegeeks.com/2011/07/java-tools-source-code-optimization-and.html, 2017

[17] G. Booch, "Object Oriented Design: With Applications", 1991

[18] M. Olszewska and J. Heidenberg and M. Weijola, "Quantitatively measuring a large-scale agile transformation", Journal of Systems and Software, 2016

[19] M. Hoegl and H. G. Gemuenden, "Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence", Organization Science, vol. 12, 2001

[20] A. Carron and L. Brawley, "Cohesion: Conceptual and Measurement Issues", Small Group Research 31, 2000

[21] E. Salas and R. Grossman, "Measuring Team Cohesion: Observations from the Science", Human Factors, vol. 57, 2015

[22] M. Casey-Campbell and M. L. Martens, "Sticking it all together: A critical assessment of the group cohesion-performance literature", 2008

[23] C. A. Wellington and T. Briggs, "Comparison of Student Experiences with Plan-Driven and Agile Methodologies", 35th ASEE/IEEE Frontiers in Education Conference, 2015

[24] A. V. Carron, L. R. Brawley, "G.E.Q. The Group Environment Questionnaire Test Manual", Fitness Information Technology, Inc., 2002

[25] F. van Boxmeer, C. Verwijs, "A direct measure of Morale in the Netherlands Armed Forces Morale Survey: 'theoretical puzzle, empirical testing and validation", Presented at Internation Military Testing Association Symposium (IMTA), 2007

[26] C. Verwijs, "Agile Teams: Don't use happiness metrics, measure Team Morale", Agilistic blog, retrieved 05.2017

[27] M. Unterkalmsteiner and T. Gorschek, "Evaluation and Measurement of Software Process Improvement – A Systematic Literature Review", IEEE Transactions on Software Engineering, 2012

[28] S. Ilieva and P. Ivanov and E. Stefanova, "Analyses of an agile methodology implementation", Proceedings of 30th EUROMICRO Conference, 2004

[29] P. Abrahamsson, "Extreme Programming: First Results from a Controlled Case Study", Proceedings of 29th EUROMICRO Conference, 2003

[30] P. Middleton and P. S. Taylor, "Lean principles and techniques for improving the quality and productivity of software development projects: a case study", International Journal of Productivity and Quality Management, 2007

[31] M. Ochodek and J. Nawrocki, "Simplifying effort estimation based on Use Case Points", Information and Software Technology, 2011

[32] R. Chillarege and I. S. Bhandari, "Orthogonal defect classification-a concept for in-process measurements", IEEE Transactions on Software Engineering, 1992

[33] M. Butcher and H. Munro, "Improving Software Testing via ODC: Three Case Studies", IBM Systems Journal, 2002

[34] A. Begel and B. Simon, "Struggles of New College Graduates in their First Software Development Job", Proceedings of 39th SIGCSE, 2008

[35] E. Brechner, "Things They Would Not Teach Me of in College: What Microsoft Developers Learn Later",2003

[36] M. Weuisfeld, "The Importance of Object-Oriented Programming in the Era of Mobile Development", InformtIT, Pearson, http://www.informit.com/articles/article.aspx?p=2036576, 2013, retrieved 05.2017