An Assessment Scheme for Student Development Projects with Software Industry Experience

Rafał Włodarski[1], Aneta Poniszewska-Marańda[1][0000−0001−7596−0813], and Anna Śniegula[1]

Institute of Information Technology, Lodz University of Technology, Lodz, Poland
rafal.wlodarski@edu.p.lodz.pl, aneta.poniszewska-maranda@p.lodz.pl, anna.sniegula@edu.p.lodz.pl

Abstract. Numerous attempts were taken to define criteria against which to evaluate and measure software project success. The complexity that surrounds it has been recognized by companies, however the scholarly world is yet to follow. In this article, three dimensions of success have been elicited basing on prior industrial studies: project quality, project efficiency as well as social factors (teamwork quality and learning outcomes).

Keywords: software projects evaluation · project success · project quality · project efficiency · capstone project · academic context

1 Introduction

Over the course of history, the approach to evaluation of students' work has greatly evolved. It begun as straightforward grading systems [1, 2] that assessed level of assimilation of knowledge and shifted to project-based pedagogy that favours active exploration of real-world challenges and privileges soft skills development. While project success is now regarded as a multidimensional construct, no framework that evaluates its different facets in case of students' work has been published.

Although studies devoted to the definition and measurement of success of industrial software development proliferate , in an academic context the same subject attracts little attention and focused on the source code by means of mining the repository data.

In this paper we refine the earlier established metrics, categorize them along the proposed success dimensions, provide necessary adaptations for an academic setting and generalize them so that they can be applied to a broad spectrum of student software undertakings.

The paper is structured as follows: the three dimensions and their measurement methods are detailed in sections two, three and four respectively. Threats to validity and a discussion part conclude the article in section five.

2 Project quality

A research of Paul Ralph and Paul Kelly [3] that examines the dimensions of success in software engineering, yields 11 themes with project being the most relevant and central concept. A study executed at Microsoft [4] confirms the finding that data on quality is the most important to the stakeholders of a software development project. The proposed evaluation approach makes a distinction between internal and external quality and proposes dedicated measures for both.

2.1 Internal quality

Researchers in computer science and commercial institutions have been prolific in defining software quality metrics. Its general notion encompasses an array of aspects the taxonomy of which was defined in the ISO/IEC 25010 international standard on software product quality [19]; it embraces facets pertaining to both functionality and technical traits. The focus of the upcoming section is on the latter, which is grounded in the source code base. A version control system, employed to a large extent in computer science related courses, can be used as an insightful source for calculation of structural code-level metrics and tracking data. Source code and the compliance with continuous integration practice are regarded as two major factors influencing internal project quality in the proposed evaluation scheme.

Complex code structures are proven to be difficult to understand and more likely to generate errors as compared to a well-designed module [7]. Complexity has a direct impact on the quality of a product, its maintainability and ease of troubleshooting. A common measure used in the industry is Cyclomatic Complexity as it proved to reveal insight on internal code quality.

Code complexity and size measure *Cyclomatic complexity (CC)* determines complexity of a program by counting the number of decisions (linearly independent paths) made in a given source code. It is a standard metric in the industry and as reported by McGabe Software Company [6], CC meets the three qualities of a good complexity measure, namely:

- it is descriptive, objectively measuring some quality – decision logic in the case of CC,
- predictive, correlating with some aspect – errors and maintenance effort,
- prescriptive, as it guides risk reduction – testing and improvement.

Further findings of the Software Assurance Technology Center (SATC) at NASA [8] concluded that it is a combination of the code size as well as the complexity that proves to most effectively evaluate quality. 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.

Maintainability ranking SIG, a software management consulting company, in collaboration with TV Informationstechnik have developed a measurement model that maps a collection of source code metrics to all the maintainability sub-facets as defined by the ISO 25010 standard: *analyzability, modifiability, testability, modularity and reusability*. The technical quality model involves analysis of the following metrics [22]:

- *Lines of Code (LOC)*, as increasing volume of the codebase implies more information to be taken into account and entails more maintenance effort,
- *duplicated LOC*, as repeated, redundant code requires rework of all of its instances in case of deficiency,
- *Cyclomatic Complexity*, as the simpler the code is the easier it is to comprehend and test,
- *parameter counts*, as unit interfaces with many attributes can indicate bad encapsulation [23],
- dependency counts, as tight coupling impedes modifications to the underlying code.

They are collected at different levels of the building blocks of the codebase: units (e.g. Java methods), modules (e.g. Java classes) or components (e.g. Java packages) and subsequently aggregated via grand total or quality profiles [23]. This operation allows to correlate its outcome to ratings of properties for the entire software project: volume, duplication, unit complexity, unit size, unit interfacing, module coupling, component balance and component independence. Property ratings are mapped to the sub-characteristics of maintainability as defined by ISO 25010 by calculating weighted averages according to dependencies designated in figure 1: every cross in a given column signifies contribution of a system property to the given sub-characteristic (row).

	Volume	Duplication	Unit size	Unit complexity	Unit interfacing	Module coupling	Component balance	Component independence
Analyzability	X	X	X				X	
Modifiability		X		X		X		
Testability	X			X				X
Modularity						X	X	X
Reusability			X		X			

Fig. 1. Property ratings mapped to the sub-characteristics of maintainability

Continuous integration First introduced by Booch [9], the concept of continuous integration aimed at avoiding pitfalls when merging code from different developers and in turn reduce the time and work efforts engendered by this

process. CI rapidly became the industry standard and is now employed in the academia as well, as effective teamwork in student projects requires regular use of a version control system. A long-term study by Technical University of Munich (TUM) [5] reports that CI was perceived as beneficial by 63% of 122 students, whereas a mere 13% was not in line with that statement.

Data can be collected and assessed with respect to any time frame, e.g. sprint, month as well an entire semester making this metric easily applicable to any student undertaking.

2.2 External quality

External product quality is quantified during the testing phase, by dynamic analysis of the product's behaviour as observed by its users[22]. Various kinds of software testing are essential to ensure software functional and technical product quality [23] however not all of them can be accommodated in a classroom. Whereas business projects comprise a dedicated testing phase, managed by a specialized team and running for several weeks, students usually perform only a rudimentary test campaign to make sure that the functionality is in place. That is followed by ad-hoc tests by the instructor at the end of the semester to evaluate his overall satisfaction level with the delivered product.

In order to address that gap, the authors of this paper propose to define metrics evaluating a subset of the product quality properties as defined by the ISO 25010 standard, prior to assignment implementation. That allows to incorporate it into the "Definition of done" criteria for Agile approaches or be used to guide a dedicated testing phase, when following a more traditional development lifecycle. In either case the metrics can serve as an evaluative measure of the project and ease the assessment of solutions for the instructor.

Moreover, students can be challenged to define the metrics themselves to infuse a real tester's thinking. Considering the array of possible projects implemented as part of Computer Science coursework and the information-rich taxonomy of product quality and quality in use, the measures can vary greatly depending on the type of the developed software system. For embedded systems projects or the ones based on CPU calculations, a particular focus can be put on performance efficiency in terms of time behaviour or resources utilization. Network Programming assignments could target the reliability feature with its availability and recoverability sub-characteristics.

In one of the authors' experiences, students whose objective was to implement an Artificial Conversational Entity supporting e-commerce catalogue browsing and product advisory, were asked to define a set of efficiency and usability metrics to assess the chatbots. They were required to:

- frame them according to the Goals-Signals-Metrics process,
- write test cases for evaluation,
- specify the method of metric procurement, the procedure of its collection and interpretation, and establish supporting tools used.

Once the teams documented their proposed metrics, a set of five most suitable measures to the context was chosen by the course instructors. Thereupon, metrics acquisition, calculation, storage and results representation method was specified and provided as a common reference for all the groups. That laid out a foundation for a "Jigsaw exercise" (Palacin-Silva et al. [21]) that was performed during one week of the testing phase when the students evaluated digital assistants developed by other teams according to a set of defined scenarios.

This approach enables assessment of the external quality aspect of software produced by students based on objective metrics and a large sample. The scores can be incorporated into the final grade for the projects, in turn facilitating solutions evaluation for the instructors.

3 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 [13] revealed that Effort and Productivity were defined as success indicators in 63% of studies of process improvement initiatives. Simply put, effort is reflected by the amount of time invested by the team and productivity by the output size in KLOC (kilo lines of code) [14] produced in the context of the development process. This paper proposes more comprehensive ways of evaluating a project in terms of efficiency and team productivity, as an incentive to produce large amount of code can have adverse effects on project quality (Fig. 2).

METRIC	CALCULATION METHOD	EVALUATED EFFICIENCY FACET
HUSTLE METRIC: FUNCTIONALITY/TIME SPENT	$HM = \sum_{i=1}^{n} Fp_i / \sum_{i=1}^{n} T_i$, where **Fpi**: number of functional points of an artifact (task, module etc.) considered **Ti**: overall time spent by the team implementing the considered functionality	overall productivity of the team
PROCESSING INTERVAL: LEAD-TIME PER FEATURE	$PI = T_{ship} - T_{acc}$, where $\mathbf{T_{ship}}$: timestamp when the feature is fully implemented and uploaded to a repository, $\mathbf{T_{acc}}$: timestamp when the feature is accepted for implementation	efficiency of implementation process and capability to tackle encountered problems
WORK IN PROGRESS	$WIP = \sum_{i=1}^{n} Fp_i$, where $\mathbf{Fp_i}$: function points of a task currently in progress	

Fig. 2. Project efficiency metrics, calculation method and aspect evaluated

4 Social factors

According to the study by Hoegl and Gemuenden [10] there are three leading factors that shape the success of innovative projects:

- team performance,
- teamwork quality,
- personal success.

While the efficiency aspect and its significance have already been covered, the following section focuses on teamwork as part of university classes and students' accomplishment. The objective of computer science higher education is to prepare corporate-ready graduates to be apt at programming and equipped with practical skills such as efficient collaboration and effective communication which can be developed while carrying out team projects. To assess growth in these areas, measurement methods of the teamwork quality as well as learning outcomes of students are proposed.

The evaluative tools described in this section can only be examined in a qualitative manner. The most straightforward and common way of doing so is through the use of opinion polls [5, 18, 20]. For the purpose of the proposed assessment scheme, it is suggested that all statements should be evaluated by students on a 4-degree Likert scale (as it leaves more space for nuance and omits a neutral answer): *strongly agree, agree, disagree, strongly disagree.*

4.1 Teamwork quality

Teamwork quality is a measure of conditions of collaboration in teams; according to Hoegl and Gemuenden [10] it consists of six facets: communication, coordination, balance of member contributions, mutual support, effort and cohesion. Capturing any of these properties of cooperation within a group is a baffling task, hence the most representative of these characteristics is examined – cohesion.

Team cohesion Team cohesion is defined as the "shared bond that drives team members to stay together and to want to work together" [12]. As stated in [11] cohesion is highly correlated with project success, critical for team effectiveness and leads to increased communication and knowledge sharing. As cohesion emerges over time and its perception among the group variates as the project progresses [11], its investigation needs to be carried out periodically. There are no restriction on the time intervals but they should be frequent enough for the assessment to stay pertinent, e.g. every sprint or month.

In the context of Software Engineering, two dimensions of cohesion can be distinguished: attachment within the team (*social cohesion S* [11]) and attachment to the project (*task cohesion T* [11]). It can be further categorized according to granularity: at project member level (*Individual Attractions to the Group ATG* [11]) and team level (*Group Integration GI* [11]). These two distinction levels yield the following aspects of cohesion:

– GI-T: The team's attachment to the task,
– GI-S: The team's social connection,
– ATG-T: Individual attachment to the task,
– ATG-S: Individual connection to the team.

Team morale A common observation is that successful teams are happy and as such, they are efficient and produce quality results [21]. Measuring happiness has been gaining popularity in Agile software development frameworks as they emphasize teamwork and recognize its human aspect. The most common practice measures a happiness index by employing a Niko-niko calendar, also known as smiley calendar, where team members systematically rate their mood with a smiling, straight-faced, or frowning smiley. As appealing as that might be to students, in order to evaluate teamwork quality and personal success at the same time one should investigate team morale instead. Its classical meaning reflects a sense of common purpose and the amount of confidence felt by a person or group of people. In a commercial setting it is linked to job satisfaction, outlook, and feelings of well-being an employee has within a workplace setting which also resonate with conditions a university course should provide.

4.2 Learning outcomes and skills

A final facet of project success is 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. [16] on struggles of new college graduates in their first development job at Microsoft finds that they have difficulties in teamwork and cognition areas. Brechner [17] suggests they should participate in dedicated courses in Design Analysis and Quality Code as part of their education in order to address the identified qualifications gap.

The proposed scheme evaluates student competencies divided into two categories, along with their their constituents:

1. *Software engineering skills*, that encompass: requirements elicitation, system design, data modelling, programming.
2. Non-technical skills, that encompass: communication, teamwork.

To increase the validity of the evaluation, the professor can contribute to the process by assessing the produced artefacts. Clark [20] has mapped learning outcomes and skills to corresponding assessment tasks as part of his study on student teams developing industry projects; its version enriched with evaluation tools described in this article is presented in table 1.

5 Conclusions

This paper provides professors and researchers an approach for the evaluation of computer science project success that encompasses three dimensions, further divided into sub facets and addressed with a specific metric, measure or pedagogic

Table 1. Learning outcomes and graduate skills – mapping of projects' outcomes, developed skills along with artefacts and proposed measures that can be used for their evaluation

Learning Outcomes	Graduate Skills	Assessment Artefacts
Working in a team, students will be able to question a client/professor to extract and analyse the software requirements and present the analysis in a written report.	Software engineering skills – requirements elicitation. Communication skills.	Requirements documentation (UML diagrams, Functional Requirements Document, User Interface Specification). User stories.
Having analysed the requirements, students will be able to prepare appropriate design documents while working in a team.	Software engineering skills: - system design, - data modelling	System architecture diagram, Detailed Design document, Database model, Data structure diagram, Entity-relationship model, Wireframes.
Having prepared design documents, students will be able to construct and integrate a significant software system while working in a team.	Software engineering skills: - programming, - version control. Team-working skills.	Developed software evaluated along its internal and external quality aspects Repository (Pacemaker: Commit pulse metric).
Having developed a software system, students will be able to produce written technical and instructive documentation on the implemented solution.	Communication skills	User manual. Installation guide.
Students will be able to formulate a schedule for a team of people and individually and collectively manage their time.	Team-working skills. Communication skills.	Issue and project tracking software (Processing Interval metric, Work In Progress metric). Time reports. Risks document.
Students will be able to work in a small team, planning effectively and be able to evaluate their own and peers performance at team and individual activities.	Team-working skills. Communication skills.	Issue and project tracking software. Team cohesion questionnaire.

tool. A description of the metrics and necessary adaptations for an academic context is provided, along with a discussion of the validity and appropriateness of the proposed measures for their intended use. The work described in this article has a number of implications.

The proposed evaluation approach contributes to the field of software engineering in terms of empirical research methods. First off, it can serve as an analytical framework for scientists investigating the applicability of software developments methodologies to a university setting. It determines areas of focus for evaluation, which can underpin given hypotheses and suggests dedicated metrics and measures that can be used for their verification. Moreover, as it provides a set of measures and pedagogic tools that are course organization-agnostic it can be applied to compare different development methodologies in terms of the project outcomes and process-related attributes.

References

1. G. Pierson, "C. Undergraduate Studies: Yale College", Yale Book of Numbers. Historical Statistics of the College and University 17011976, 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. R. Buse and T. Zimmermann, "Information needs for software development analytics", Proc. of 20th International Conference on Software Engineering, IEEE Press, pp. 987–996, 2012.
5. B. Bruegge and S. Krusche and L. Alperowitz, Software Engineering Project Courses with Industrial Clients", 2015.
6. McCabe Associates, "Integrated Quality" as part of CS699 Professional Seminar in Computer Science, 1999.
7. B. Kitchenham and S. Pfleeger, "Software Quality: The Elusive Target", IEEE Software, 1996.
8. L. Rosenberg and T. Hammer, "Software metrics and reliability, NASA GSFC, 1998.
9. G. Booch, "Object Oriented Design: With Applications", 1991.
10. M. Hoegl and H. Gemuenden, "Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence", Organization Science, vol. 12, 2001.
11. A. Carron and L. Brawley, "Cohesion: Conceptual and Measurement Issues", Small Group Research 31, 2000.
12. M. Casey-Campbell and M. Martens, "Sticking it all together: A critical assessment of the group cohesion-performance literature", 2008.
13. M. Unterkalmsteiner and T. Gorschek, "Evaluation and Measurement of Software Process Improvement – A Systematic Literature Review", IEEE Transactions on Software Engineering, 2012.
14. S. Ilieva and P. Ivanov and E. Stefanova, "Analyses of an agile methodology implementation", Proc. of 30th EUROMICRO Conference, 2004.
15. M. Ochodek and J. Nawrocki, "Simplifying effort estimation based on Use Case Points", Information and Software Technology, 2011.
16. A. Begel and B. Simon, "Struggles of New College Graduates in their First Software Development Job", Proc. of 39th SIGCSE technical symposium on Computer science education, 2008.
17. E. Brechner, "Things They Would Not Teach Me of in College: What Microsoft Developers Learn Later", ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2003.
18. G. Melnik and F. Maurer, "A Cross-Program Investigation of Students Perceptions of Agile Methods", International Conference on Software Engineering, 2005.
19. ISO/IEC25010: 2011 Systems and software engineering, 2011.
20. C. Clark, "Evaluating student teams developing unique industry projects", Proc. of 7th Australasian Conference on Computer Education, 2005.
21. M. Palacin-Silva and J. Khakurel and A. Happonen, "Infusing Design Thinking Into a Software Engineering Capstone Course". Proc. of 30th IEEE Conference on Software Engineering Education and Training (CSEE&T), 2017.
22. D. Bijlsma and M. Ferreira and B. Luijten and J. Visser, "Faster Issue Resolution with Higher Technical Quality of Software", Software Quality Journal, 20(2), pp. 265-285, 2012.
23. R. Baggen and J. Correia and K. Schill and J. Visser, "Standardized code quality benchmarking for improving software maintainability", Software Quality Journal, 20(2), pp. 287–307, 2012.
24. A. Albrecht, "Measuring Application Development Productivity", Proc. of Joint SHARE, GUIDE, and IBM Application Development Symposium, pp. 83-92, 1979.