Strengthening Web Based Learning through Software Quality Analysis

Juan Manuel Montero, Ruben San Segundo, Ricardo De Cordoba, Amparo Marin de la Barcena, and Alexander Zlotnik

Grupo de Tecnologia del Habla, DIE, ETSIT, Universidad Politécnica de Madrid, Avenida Complutense nº30, 28040 Madrid, Spain
juancho@die.upm.es, lapiz@die.upm.es, cordoba@die.upm.es, amarin.alumno.ie@gmail.com, alexander@zlotnik.net

Abstract. The Web is changing the way people access & exchange information. Specifically in the teaching & learning environment, we are witnessing that the traditional model of presence based magisterial classes is shifting towards Web Based Learning. This new model draws on remote access systems, knowledge sharing, and student mobility. In this context, pedagogical strategies are also changing, and for instance, Project- Based Learning (PBL) is seen as a potential driver for growth and development in this arena. This study is focused on a PBL oriented course with a Distributed Remote ACcess (DRAC) system. The objective is to analyze how quantitative methods can be leveraged to design and evaluate automatic diagnosis and feedback tools to assist students on quality-related pedagogical issues in DRAC enabled PBL courses. Main conclusions derived from this study are correlation-based and reveal that the development of automatic quality assessment and feedback requires further research.

Keywords: Project Based Learning, Software Quality Analysis, Remote Access.

1 Introduction

Although there is no-one accepted definition for PBL, a standard one defines PBL as a *systematic teaching method that engages students in learning knowledge and skills through an extended inquiry process structured around complex, authentic questions and carefully designed products and tasks* [1]. PBL is becoming an important educational approach to help faculty improve student outcomes and there are several examples of the PBL technique successfully applied in both pre-university [2] [3] and university courses [4] [5]. In university teaching it has been applied to an ample variety of disciplines including science, arts, business & entrepreneurship education, law, medicine [6] [7] [8]; but most applications have been in technical and engineering courses [9] [10] [11] [12].

In PBL oriented courses, students' and instructors' focus shifts to cover not only functional but also non-functional quality aspects. Over the last few years, there have been several works towards developing automatic tools for supervising and evaluating

M. Spaniol et al. (Eds.): ICWL 2009, LNCS 5686, pp. 277–286, 2009.

student work as well as facilitating feedback [13] [14] [15] [16]. It is expected that as instructors measure students' performance and provide them with mid-course feedback, supported by automatic diagnosis and supervision tools, students will improve their non-functional skills (e.g. developing high quality software). Moreover, student performance measurement and making feedback available to remote access students will also contribute positively to the development of web based learning.

2 Description and Context of the Course

Analyzed data proceed from and relate to the LSED course (Laboratory of Digital Electronic Systems), which is taught by the Department of Electronics Engineering at the Telecommunications Engineering School of the Polytechnic University of Madrid (UPM). The course's main objective is to serve as a practical approach to the key phases involved in the development of a digital electronic system prototype (including Hardware (HW) and Software (SW) based on a MC5272 ColdFire microcontroller. In modern engineering education, remote access is becoming increasingly important to cope with students' demand for more web based access tools that allow them to use an integrated development environment from their homes instead of having to attend physically to the laboratory.

The implementation of a PBL oriented remote on-line based laboratory entails several difficulties and might require expensive and/ or specialized equipment. Therefore, the approach towards enabling remote access in the LSED is a Distributed Remote ACcess (DRAC) system mainly consisting on a web-based portal that provides simultaneous access to software and hardware resources for several students. [17]. The system was designed to be applicable to certain subjects related to microcontroller programming and digital electronic design with a great emphasis on multidisciplinary interactive applications. A cost-effective mashup approach was followed through the use of several open source technologies. These technologies are not designed for interoperability and are combined in a single system using the best of their individual features. The main implementation problem was the classical one in this kind of system: glue logic. [18]

LSED is a laboratory with a high students-to-faculty ratio and attended by ~400 students every year, but only a minority is aware of the DRAC's capabilities. Students, grouped in couples, have to design, build, test and document a complete microprocessor-based system (both HW and SW) and, at the end of the course, they are evaluated based on a written final report, and an oral examination, which mainly serves to verify that the prototype meets the specifications, check the quality of the software and determine students' ability to explain the obtained results. There are detailed evaluation forms which are filled by the instructors, and in the end students obtain a grading score ranging in a 0-100 scale.

Instructors, at the laboratory sessions, teach students not only the microprocessor's capabilities and some practical implementation issues, but also a systemic point of view, involving multi-disciplinary knowledge: communications, control, user interfaces, etc. This provides a competitive advantage to students who physically attend the laboratory over remote access students, and thereby the interest on software quality analysis to develop web based tools that contribute to reduce that gap.

3 PBL, a New Pedagogical Approach

Assessment, student centered, collaboration, real world connection, extended time frame and multimedia are considered key levers of the PBL approach.

It is a common trend that students and instructors, when dealing with a project which combines hardware and software, focus on the functional aspects setting aside non-functional skills such as developing high-quality software. Nevertheless instructors have started to realize the importance of these non-functional skills and the contribution of PBL techniques in this sense. These allow, not only to grade in a more accurate and comprehensive way by devoting importance to a broad set of components, but also to measure software quality in order to provide students with mid-course feedback on how to improve their software.

Although it is not easy to provide a precise definition of software quality, experts classify software programs in terms of quality based on two non-functional aspects:

- a better code structure and documentation, which makes programmers more lean and agile to undertake complex projects, at a lower effort and including more functionalities.
- an efficient and smart use of data structures, which adds flexibility to the solutions, whilst leading to more elegant algorithms.

Our proposed approach consists on evaluating software quality through quantitative methods based on a two-step process:

- Feature extraction: to quantify those features that could be related to high-quality software.
- Feature analysis: to assess the relevance of the features used in terms of impact on the final grade studying the correlation and mathematical patterns involved.

The outcome of the feature extraction and analysis applied to data from a given academic year could be taken as a reference to set the target objectives in the following years.

4 Analysis of Software Quality Features

This study focuses on a 65-student sample space. For each of them we know the final grade which considers some objective criteria (fulfillment of technical specifications) and a subjective component based on non-functional parameters such as flexibility, tolerance or intelligibility, among others. The objective is to identify code-quality parameters with significant influence on LSED students' final grades as a mean to develop a grade/ performance predicting tool.

A Feature extraction

16 variables have been defined, analyzed and grouped in two categories (Table 1)

- Code structure and documentation: including number of subroutines and their average length, average number of exit points per subroutine, number of commented lines, etc.

- The use of data structures: this includes the use of arrays, global variables, constants, tables, messages, etc.

As students have several degrees of freedom in the design phase, some of the variables have been normalized by the number of code lines with the aim of making them more comparable and avoid favoring longer programs. Variables were collected through a C code analyzer and afterwards introduced in a spreadsheet application for the purpose of analyzing the collected data.

Table 1. Variables' categorization and main values

Categor	Variable	Comments	Rel. mean	Rel Pear -son	Abs. mean	Abs. Pear -son
Code structure & docume-ntation	Number of code lines	Relevant; Apparently, high values signal good students	1,00	N.A.	484,09	0,38
	Number of subroutines	Relevant even after normalizing; high values apparently signal good students	0,06	0,30	30,69	0,43
	Number of exit points per subroutine	Low relevance and negative contribution; can be reduced	0,01	-0,08	1,64	-0,08
	Mean subroutine length	Negative impact; can be reduced (area for improvement)	0,07	-0,23	22,40	-0,22
	Length of the longest subroutine	Irrelevant; can be reduced	0,19	-0,30	67,29	0,04
	% commented lines	Negative after normalizing; should significantly increase	0,22	-0,10	68,53	0,16
Data structures	Number of Loops	Relevant even after normalizing; indirect complexity indicator	0,05	0,20	21,92	0,50
	Number of IFs	Almost relevant after normalizing; indirect complexity indicator	0,11	0,13	49,11	0,36
	Number of lines with [ ]	Negative after normalizing; apparently high	0,23	-0,10	110,71	0,22
	Number of strings / messages	Relevant even after normalizing; apparently high	0,13	0,23	58,22	0,52
	Number of GOTOs	Fortunately nobody has used them	0,00	0,00	0,00	0,00
	Number of STRUCTs	Nobody has used them; they should be more used	0,00	0,00	0,00	0,00
	Number of MACROS	Scarcely used; should be more used	0,00	0,22	0,28	0,25
	Number of INCLUDEs	Relevant even after normalizing; should be more used	0,02	0,32	7,83	0,45
	Number of DEFINEs	Relevant even after normalizing; should significantly increase	0,05	0,30	21,12	0,51
	Number of TYPEDEFs	Low number of cases; should be more used	0,00	0,21	0,32	0,19

B Feature analysis

For the target variables included in our study, the average (MEAN function), standard deviation, maximum and minimum values and the Pearson correlation coefficient were calculated. Given the final grades for each student and their correlation with the

Table 2. Criteria to assess features' relevance on grades predicting

Correlation	Negative	Positive
Small	$-0.3 \leq \rho_{x,y} \leq -0.1$	$0.1 \leq \rho_{x,y} \leq 0.3$
Medium	$-0.5 \leq \rho_{x,y} \leq -0.3$	$0.3 \leq \rho_{x,y} \leq -0.5$
Large	$-1.0 \leq \rho_{x,y} \leq -0.5$	$0.5 \leq \rho_{x,y} \leq 1.0$

selected quality variables, the goal is to assess the relevance of each feature on the final grade.

The correlation coefficient is a value ranging between [-1, 1]. Two variables are said to be non-correlated when their correlation coefficient is null; for the purpose of this study, the relevance of the features analyzed for grades' predicting is subject to specific criteria (Table 2.) Additionally, when the absolute value of the correlation is close to 0, it is assumed that the feature is not relevant to predict students' grades.

A first comprehensive analysis of the selected variables' Pearson coefficient was conducted, considering the overall students sample; this revealed that none of the selected variables' Pearson was high enough to classify it as decisive to predict students' grades. Most of the variables showed a small or medium correlation and only the use of strings/ messages, and the use of DEFINEs surpassed the threshold of large correlation variables. (Fig. 1.)

Nevertheless, results after normalizing showed a different picture, revealing that the use of optimized subroutine lengths was the most correlated normalized variable to students' grades. (Fig. 2.)

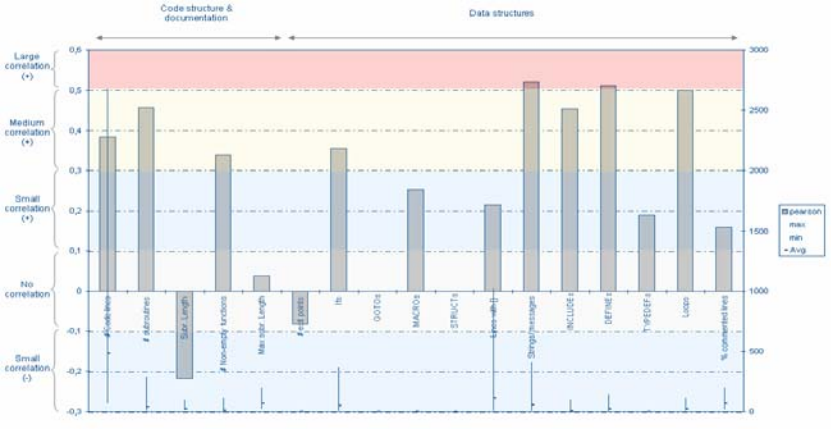

Fig. 1. Analysis of selected variables

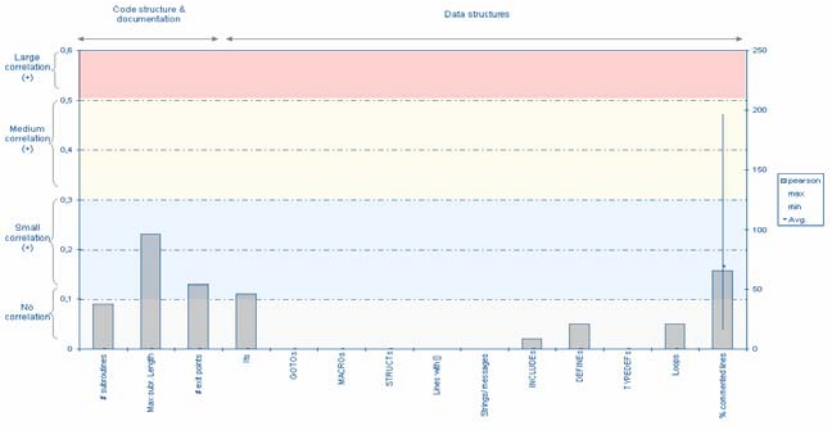

Fig. 2. Analysis of selected normalized variables

Table 3. Criteria to assess features' relevance on grades predicting

Segment	Description	% over total
Top performers	Students whose grade is ≥ 100.0	20%
Average	Students whose grade is ≥ 75.0 but < 100.0	35%
Low performers	Students whose grade is < 75.0	45%

As a next step, in order to get a deeper understanding of the results, the 65 students sample was segmented into 3 groups (Table 3.)

For each of the groups, the Pearson coefficient, mean, maximum and minimum values, and the standard deviation were analyzed. The outcome of this analysis confirmed that features related to data structures have a higher impact on students' grades than those related to the structure and code documentation, but as students get more experienced, these latter features become more important, increasing their weight from 26% in the group of lower performers to 42% in the group of top performers. (Fig. 3.)

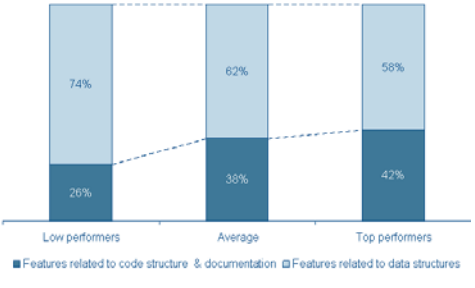

Fig. 3. Weight of the different variables per student segment

From the compilation of the results derived from the various analyses conducted the following highlights have arisen in the features analyzed:

Features related to the structure and code documentation

- Number of lines of code (avg. 484, Pearson 0.38): the volume of lines of code is apparently advantageous for the grade, as it enables to create more complete algorithms (Top performers (752 lines) vs. Low performers (319 lines)). However, it is observed that for those students who have achieved a reasonable advanced level, more code lines often result in more confusion and space to commit mistakes. Hence, advice aimed to students in this regard should be to look for code optimization, producing more compact and better readable algorithms.
- Number of subroutines (avg. 31, relative Pearson 0.3): the number of functions is related to the scope of the program's functionality; increasing number of functions enables increasing system's functionalities and avoids excessive length of code blocks, which difficult programs' tracking and reveal that there is a flaw in the design phase. (Top performers (58 subroutines) vs. Low performers (17 subroutines)).
- Number of exit points per subroutine: The structure of subroutines is also relevant: they should be non-interlaced (non-overlapped) and with just one exit point or return instruction per function. (Current average is around 1.6)
- Mean subroutine length: represents the average size of students programs' functions. Its Pearson coefficient in absolute terms is negative (-0.22) signaling that long functions are not a good programming practice to achieve a good grade. On the contrary, smaller and more specialized functions make better programs, easier to analyze and debug. (Students achieving the best grades use a below average number of lines per subroutine (17 vs. 23)
- Length of the longest subroutine: it is measured in number of code lines and after normalizing its Pearson coefficient is -0.30. However, taking into account the Pearson per student segment, it is worth highlighting that this parameter achieves a significant value (0.46) for the average students segment. Consequently we can infer that the length of the longest subroutine is positively correlated with the grade until students reach a certain knowledge level. At that point they understand the importance of code optimization and thus the Pearson coefficient turns negative.
- Number of commented lines (absolute Pearson 0.16): is considered in relative terms, as a % of total code lines (68.5% on avg.). Most commented programs are not necessarily the best graded, as students with the longest programs tend to focus on including more functionalities without paying the same degree of attention to keeping comments at such high level.

Features related to the use of data structures

- Number of loops and IFs: the use of unconditional and conditional jumps, which is related to loops and if-then-else structures, adds complexity to the programs; a high number of functions implemented is an indirect indicator of

complexity; the most complex programs have more but shorter local jumps, limited by the size of the subroutines. Underscore that the Pearson coefficient of loops after normalization reaches 0.20.

- Number of complex data structures: linked to the use of arrays which allow more compact and smarter algorithms (number of lines with []) and messages (warning, error) for a better user interface (number of strings/ messages). The relevant feature here is the number of strings, which after normalizing shows a Pearson coefficient of 0.23. On the other hand, the number of lines with [] is negatively correlated (-0.10) after normalizing this feature, which leads to infer that after a certain point, the use of additional complex data structures may difficult the reading of programs and thus contribute negatively to the grade.
- Number of GOTOs: often considered as a bad programming practice; fortunately no single use case has been described.
- Number of STRUCTs and MACROs: the use of these commands is an indicator of advanced knowledge programming level. Unfortunately, STRUCTS, which are elegant data structures, were not used by the students
- Number of INCLUDEs, DEFINEs and TYPEDEFs: ease the access to complex data structures such as tables or lists, which are indeed related to a more elegant programming style, revealing in many cases higher quality software. Specifically in the case of INCLUDEs, the normalized Pearson coefficient is in the range of [0.23 – 0.30]. Moreover, the use of DEFINEs has more weight as students acquire more advanced programming skills.

5 Results Assessment

From the data analyzed, several areas of improvement have been identified; in this regard, the proposition of specific targets and initiatives should help achieve students' software quality enhancement and new learning objectives in the following courses:

- In the assignment's introductory text, include both general and specific recommendations and examples of good coding practices
- Define quality rules of thumb to guide students:
    - The longest subroutine should not exceed 50 lines
    - Average number of subroutines should be above 25
    - Average subroutine length should be below 20 lines
    - 95% of the subroutines should have just one exit point
    - 30% of code lines should include comments
- Develop automatic web-based diagnosis tools to provide mid-course student feedback, early detection of bad programming habits and deter students from plagiarism.

In terms of the weight of quality features on students' grades, the number of subroutines, the use of strings / messages, and the use of complex structures has a

positive impact on students' grades showing positive correlations above 0.4. Additionally, as instructors emphasize on code optimization and program documentation aspects, it is expected that variables such as the mean subroutine length and the % of commented lines will become increasingly important.

The present analysis has also served instructors to identify concepts that students have not yet fully assimilated:

- Number of exit points per subroutine should be ideally 1: most top performers are close to attaining this, whereas the average is around 1.6
- Use of commands and complex data structures: (e.g. STRUCTs, MACROs, INCLUDEs) these can be very useful, but there is a low number of examples on students' programs.

Conclusions and findings can be used as the basis not only to orientate students on their performance compared to the rest of their classmates on that year, but also to provide students some advice and guidelines which have been helpful to others in similar circumstances; i.e. to build-up a continuous learning process based on past experiences.

6 Conclusions

This paper has presented an analysis of 16 selected code quality parameters in the context of a PBL remote access course. The objective was to identify key variables that could influence students' programming behavior and thus, to what extent these could have an impact on students' grades. Results derived from this analysis are based on a 65 student sample and therefore, should not be considered as statistically relevant, though useful to identify and understand certain trends. Main conclusions and findings reveal that none of the variables selected had a determinant impact on student's grades (with correlations < 0.55). For instance, although the number of strings/ messages, considering the whole students sample, showed the highest Pearson coefficient (0.52), if the sample was segmented based on students' performance, both the most correlated variable and the degree of correlation may vary per category (Top Performers, Number of Macros (-0.44); Average Performers, Number of Strings/ Messages (0.51); Low Performers, Number of IFs (-0.37)).

One key concern related to the use of a correlation-based analysis is that if features are correlated amongst them, and one feature shows a significant correlation, the remaining features will also show correlations in line. In this regard and to isolate this effect, we have conducted a recursive multi-phase analysis: For each iteration, the highest correlated variable was identified and, in the next phase, all variables were normalized with respect to the most correlated variable. For the purpose of this study after two iterations were completed, the number of strings / messages (1^{st} iteration) and the length of the longest subroutine (2^{nd} iteration), resulted the most correlated variables (Pearsons of 0.52 and -0.44 respectively).

References

1. Project Based Learning Handbook, pp. 3–10. Buck Institute of Education (2007)
2. Katz, L.G., Chard, S.C.: Engaging Children's Minds: The Project Approach. Ablex, Norwood (1989)
3. Chard, S.C.: The Project Approach: A Practical Guide for Teachers. University of Alberta Printing Services, Edmonton (1992)
4. Barrows, H.S.: Problem–based learning in medicine and beyond: A brief overview in Bringing problem–based learning to higher education: Theory and practice, pp. 3–12. Josey-Bass, San Francisco (1996)
5. Solomon, G.: Project–Based Learning: A Primer. Technology and Learning 23, 20–27 (2003)
6. Albanese, M.A., Mitchell, S.: Problem–based learning: A review of literature on its outcomes and implementation issues. Academic Medicine 68, 52–81 (1993)
7. Vernon, D.T.A., Blake, R.L.: Does problem-based learning work? A meta-analysis of evaluation research. Academic Medicine 68, 550–563 (1993)
8. D'Intino, R.S., Weaver, K.M., Schoen, E.J.: Integrating a Project Based Learning Model into the Entrepreneurial University. Rowan University, Glassboro (2007)
9. Macías-Guarasa, J., Montero, J.M., San-Segundo, R., Araujo, A., Nieto-taladriz, O.: A Project-Based Learning Approach to Design an Electronic Systems Curricula. IEEE Transactions on Education (2006)
10. Hedley, M.: An undergraduate microcontroller systems laboratory. IEEE Transactions on Education 41, 345 (1998)
11. Ambrose, S.A., Amon, C.H.: Systematic design of a first-year mechanical engineering course at Carnegie-Mellon University. Journal of Engineering Education 86, 173–182 (1997)
12. Harris, J.G. (moderator) Journal of engineering education round table: reflexions on the grinter report. Journal of Engineering Education, 83, 69–94 (1994)
13. Cheang, B., Kurnia, A., Lim, A., Oon, W.C.: On automated grading of programming assignments in an academic institution. Computers & Education, 41, 21–131 (2003)
14. Kurnia, A., Lim, A., Cheang, B.: Online Judge. Computers & Education, 36, 299–315 (2001)
15. Korhonen, A., Malmi, L., Nikander, J., Tenhunen, P.: Interaction and Feedback in Automatically Assessed Algorithm Simulation Exercises. Journal of Information Technology Education, 2, 241–255 (2003)
16. Baillie-de Byl, P.: An Online Assistant for Remote, Distributed Critiquing of Electronically Submitted Assessment. Educational Technology & Society, 7, 29–41 (2004)
17. Nieto-Taladriz, O., Araujo, A., Fraga, D., Montero, J.M., Izpura, J.I.: Antares: A synergy between University Education and Research, Development and Technology Innovation Groups. In: Proceedings of the Colloquium on Higher Education of Electronics, pp. 23–26 (2004)
18. Zlotnik, A., Montero, J.M.: DRAC (Distributed Remote ACcess System): An On-line Open Source Project-Based Learning Tool. In: ICL, Villach, Austria, Oral, September 2007, pp. 26–28 (2007)