AUTOMATIC TOOLS FOR SOFTWARE QUALITY ANALYSIS IN A PROJECT BASED LEARNING COURSE

J.M. Montero Martínez, R. San Segundo, R. De Cordoba, A. Marin de la Barcena, A. Zlotnik

Polytechnic University of Madrid
Madrid, Spain
juancho@die.upm.es, lapiz@die.upm.es, cordoba@die.upm.es, amarin.alumno.ie@gmail.com, alexander@zlotnik.net

Abstract

Over the last decade, the "Anytime, anywhere" paradigm has gained pace in Higher Education teaching, leading many universities to innovate in pedagogical strategies based on Internet and Web access technologies. Development of remote access technologies has enabled teachers to achieve higher levels of efficiency while students can access tools and resources no longer constrained by time or location. Additionally, students can submit their assignments, be evaluated and be provided feedback remotely. In this context arises the need for faculty to dispose of automatic tools that ease and support the evaluation process whilst facilitating the provision of student feedback. Project Based Learning (PBL) has emerged as a pedagogical strategy that can contribute to measure software quality and thus evaluate students in a more accurate and comprehensive way by devoting importance to a broad set of components, not just focused on functional aspects. This paper analyzes how the introduction of innovative automatic diagnosis and feedback tools, based on quantitative methods, can contribute towards a continuous process of student software quality enhancement and higher efficiency programming in PBL courses, without compromising functional aspects, as students are provided practical guidelines by instructors on a timely basis.

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]. The 21st century knowledge and information economy is shaping the way in which colleges and universities throughout the world are preparing undergraduate and graduate students with knowledge skills and abilities. 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]. Compared to the traditional ways of teaching, the PBL technique reveals a higher degree of learning [13] [14] and the difference is even greater when PBL is supported by new technologies [15] [16]. It is observed that PBL allows increasing student involvement in the learning process, obtaining better results in terms of knowledge and habits acquired by the students. In our PBL approach they must face a multidisciplinary project aimed at developing new capabilities that complete their instruction and are under great demand in ICT companies [17]: teamwork, self-learning, assumption of responsibilities, resources management and time-planning.

The PBL strategy also poses several implementation problems such as greater management and coordination effort (especially in courses with hundreds of students) and increasing complexity in the evaluation process, as 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 student work as well as facilitating feedback [18] [19] [20] [21]. Generally, these tools are applied to software assignments and circuit simulations, using test vectors or use cases. This approach is not feasible in our course because student systems are interactive and complex, non-functional aspects should also be evaluated and there are functional implementation differences amongst teams (the technical description of the project provides some guidelines to undertake the project, but in the end it is students' creativity which plays a key role in the definition of systems' functionalities).

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).

ISBN:978-84-612-9802-0

2 DESCRIPTION AND CONTEXT OF THE COURSE

Analyzed data proceed from and relate to the LDES 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 HW and SW) based on a MC5272 ColdFire microcontroller. LDES is a laboratory with a high students-to-faculty ratio and attended by ~400 students every year. Students, grouped in couples, have to design, build, test and document a complete microprocessor-based system (both HW and SW). Instructors 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. Typically, the system proposed is a simplified version of a consumer system and it changes every year. (e.g. talking calculator (2002/2003), RTTL-based music synthesizer (2006/2007)). Students' mandate is to develop a completely functional prototype, backed up by a written report which compiles the main results and conclusions of the work performed.

Students are initially provided a document with the assignment's specifications including a written description of the system to be implemented, the system's requirements, a modular description and the main subsystems, some guidelines for the implementation and a tentative planning schedule to help students manage the different laboratory sessions in order to achieve the objectives on time.

At the end of the course, students are evaluated based on a written 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.

3 PBL, AN INNOVATIVE PEDAGOGICAL APPROACH

Assessment, student centered, collaboration, real world connection, extended time frame and multimedia are considered key levers of the PBL approach. Compared to traditional instruction which emphasized on content coverage, knowledge of facts and development of complex problem-solving skills in isolation, PBL is oriented towards comprehension of concepts and principles and has a broader interdisciplinary focus that follows students' interests.

Assessment tools in traditional instruction, such as test scores and comparison with others, have largely focused on functional aspects. Nevertheless, in PBL, in order to devote importance to non-functional skills, there are attempts to provide students with mid-course feedback which also considers software quality.

Although it is not easy to provide a precise definition of software quality, experts are able to 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 APPLICATION OF REMOTE ACCESS TOOLS TO A PBL COURSE

A web-based portal has been implemented in the Department of Electronics Engineering. The system allows remote access to the hardware and software resources of several PBL laboratories on electronics. Specifically, it has been already tested for the LDES allowing students to use, "anywhere, anytime", the microprocessor platform and the integrated development environment, thus eliminating the dependency on physical attendance to access the lab's resources.

Although the system is currently in use, it remains under test and undergoes progressive improvements mainly related to real numerical scalability and a better fault tolerance.

As students complained that PBL courses often required more laboratory time than was actually available, the Department of Electronics Engineering developed its own Distributed Remote ACcess (DRAC) system, aimed at enabling a 24-hour access for students to the PBL laboratories' resources.

Development of remote access technologies has allowed teachers to achieve higher levels of efficiency by attending a larger number of students with the same effort and providing information on students' performance which was not known before. At the same time, such remote access technologies enable students not only to take advantage of tools and resources that were not previously available to them, but also to take courses, submit their assignments and be evaluated remotely.

To meet the need of faculty to dispose of automatic tools that ease and support the evaluation process whilst facilitating the provision of student feedback, automatic tools for software quality analysis have been developed. The output of these tools (Table 1.) is a report including how each student scores in selected code quality parameters and the comparison of these results with the overall average for all the students on a given year. Through these tools, the role of instructors to assess students' performance in a more objective way is facilitated, and in the near future it is possible that reports are adapted and delivered to students as a feedback mechanism.

Table 1. Illustrative output report

Estimated quality	Feature	Value	Mean +- typ. Deviation
	Number of lines of code	212.000	309.233 +- 147.204
+-	Number of non-empty functions	34.000	38.258 +- 10.014
+-	Length of the longest subroutine	22.000	51.767 +- 75.041
+	**Avg. Number of lines per subroutine**	8.260	10.130 +- 2.986
-	**Avg. Number of parameters in subroutines**	21.000	27.667 +- 9.221
--	**Máx. Number of parameters per subroutine**	3.000	3.900 +- 0.597
+-	Average number of points of return per subroutine	1.206	1.213 +- 0.183
+	**Number of global variables**	2.000	4.483 +- 4.157
+-	Number of typedef and struct	10.000	11.217 +- 2.751
-	**Number of messages and strings**	61.000	75.700 +- 17.563
+-	Number of lines with []	10.000	41.883 +- 92.949
-	**Number of files (#include)**	8.000	12.217 +- 6.080
+-	Number of #define	161.000	155.550 +- 22.794
+-	Number of complex #define	24.000	23.383 +- 3.194
+-	Number of constants	3.000	2.950 +- 0.617
-	**Number of loops**	12.000	16.150 +- 6.008
-	**Number of IF structures**	14.000	26.167 +- 17.323
+-	Number of goto instructions	0.000	0.017 +- 0.128
-	**Number of switch instructions**	0.000	2.617 +- 2.727
-	**Number of default cases**	0.000	1.033 +- 1.712
+	**% of commented lines**	56.911	49.354 +- 8.830

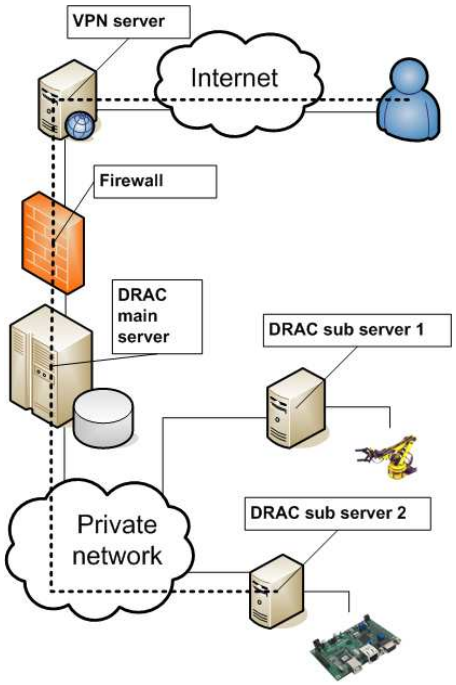
In **bold**, most outstanding features considered by the instructor when evaluating this student. These results correspond to a fake student whose code simply compiled but had not completed the assignment. Quality was estimated at 0.6 over 2.5

5 IMPLEMENTATION AND APPROACH TO SOFTWARE QUALITY FEATURES ANALYSIS

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 LDES is a Distributed Remote ACcess (DRAC) system (Fig. 1) mainly consisting on a web-based portal that provides simultaneous access to software and hardware resources for several students. [22]. 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 [23].

The DRAC mechanism can be summarized as follows: users connect to a server (DRAC main server or DRAC MS), which provides them a temporary and restricted connection to sub-servers (DRAC SS) or network-accessible resources (such as RFID readers with a web interface). The only requirements for the end-users are a modern Java-enabled web browser, a VPN client and a 128Kbps (or better) Internet connection.

Fig. 1. DRAC Network Scheme



Through the DRAC, teachers can access information on students' performance, as students' programs submitted through this system are evaluated based on a C code analyzer (RSM) which benchmarks each individual student's score on selected quality parameters against the overall students' average on a given year.

The relevance of code-quality parameters is subject to their correlation with students' final grades and helps instructors to provide students with performance feedback and diagnose areas for improvement.

Software quality analysis is conducted via a two phase approach:

A. Feature Extraction

Definition, analysis and categorization of variables. For the purpose of this study, two broad categories have been defined (Table 2.):

- 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 with the aim of making them more comparable. The normalization base is the number of code lines in order to avoid favoring longer programs.

Table 2. Variables considered

	Variable	Comments	Relative mean value	Relative Pearson s	Absolute mean value	Abs. Pearons s
Code structure & documenta -tion	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
	Number of Loops	Relevant even after normalizing; indirect complexity indicator	0,05	0,20	21,92	0,50
Use of data structures	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
	Num. of STRUCTs	Nobody has used them; they should be more used	0,00	0,00	0,00	0,00
	Num. of MACROS	Scarcely used; should be more used	0,00	0,22	0,28	0,25
	Num. of INCLUDEs	Relevant even after normalizing; should be more used	0,02	0,32	7,83	0,45
	Num. of DEFINEs	Relevant even after normalizing; should significantly increase	0,05	0,30	21,12	0,51
	Num. of TYPEDEFs	Low number of cases; should be more used	0,00	0,21	0,32	0,19

As the sample size (65 students) is not statistically relevant and has just been considered to determine trends, both the absolute and relative values should be evaluated. If conclusions based on absolute values coincide with those of relative values, there is a relatively funded basis to make a statement (e.g. the longer the subroutine length, the worse – both Pearsons are negative); else, uncertainty arises.

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

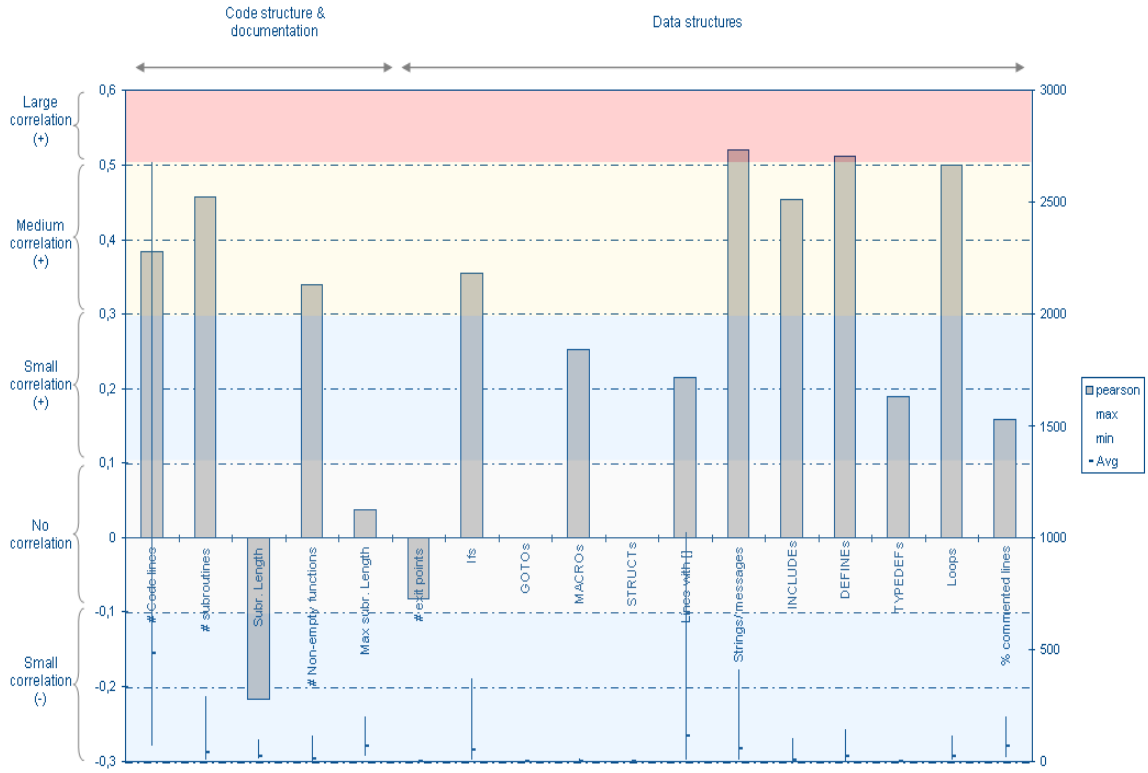
The correlation coefficient is a value ranging between [-1, 1] and the relevance of the features analyzed for grades' predicting in this study is subject to specific criteria (Table 3.). A null correlation coefficient means that the variables considered are non-correlated and when the absolute value of the correlation is close to 0, it is assumed that the feature is not relevant to predict students' grades.

Table. 3. 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	$-1.0 \leq \rho_{x,y} \leq -0.5$	$0.5 \leq \rho_{x,y} \leq 1.0$

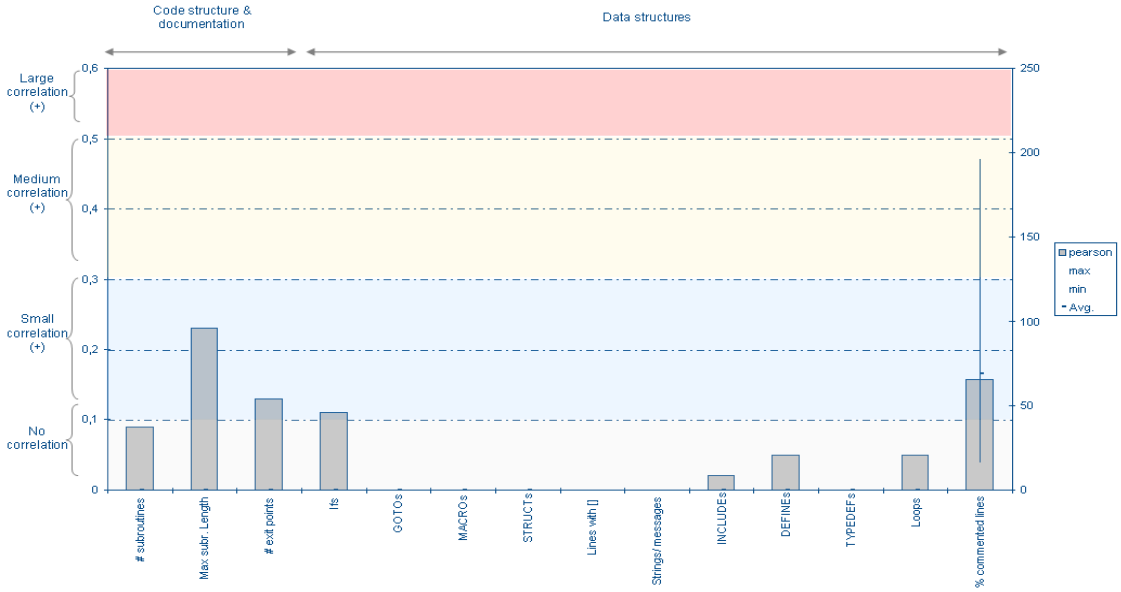
A comprehensive analysis of the selected variables' Pearson coefficient was conducted, considering the overall students sample; this revealed that none of the selected variables' Pearson was high enough to classify it as decisive to predict students' grades. Most of the variables showed a small or medium correlation and only the use of strings/ messages, and the use of DEFINEs surpassed the threshold of large correlation variables. (Fig. 2.)

Fig. 2. Analysis of selected variables



Nevertheless, results after normalizing showed a different picture, revealing that the use of several parameters in functions was the most correlated normalized variable to students' grades (Fig. 3.).

Fig. 3. Analysis of selected variables after normalization



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. 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.
- 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. (Average is currently 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.
- Length of the longest subroutine: it is measured in number of code lines and after normalizing its Pearson coefficient is -0.30. Based on our analysis 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 realize 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.

6 IMPLEMENTATION AND APPROACH TO SOFTWARE QUALITY FEATURES ANALYSIS

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
- Refine 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; in other words, to build-up a continuous learning process based on past experiences.

7 CONCLUSIONS

This paper has presented an analysis of 16 selected code quality parameters in the context of a PBL virtual laboratory. The objective was to present a methodology that faculty are approaching to innovate in High University teaching, mainly consisting on applying quantitative methods for the identification of key variables that could be relevant for students' programming behavior and assess to what extent these could have an impact on students' grades. Underscore that, since it is thought that the provision of mid-course feedback could significantly contribute towards students' software quality improvement there is an initiative underway in the laboratory that looks to generate automatic performance assessment reports that will be adapted and delivered to students in the near future. 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), being the number of strings/ messages the variable that showed the highest Pearson coefficient with the final grade (0.52).

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 only two iterations were completed, being the number of strings / messages (1^{st} iteration) and the length of the longest subroutine (2^{nd} iteration), the most correlated (Pearsons of 0.52 and -0.44 respectively).

Currently available automatic tools for software quality analysis and student assessment at the laboratory where this study was conducted remain under development, although they are proving reasonable results. As an example, when performance of a "fake" student is assessed, assuming that the student's code simply compiles (no added value involved), the automatic tool diagnoses that this student's performance is below average, specifying how he scores in each of the quality parameters evaluated.

8 REFERENCES

[1] Buck Institute of Education. (2007). **Project Based Learning Handbook**, 3-10.

[2] Katz, L.G. and Chard, S.C. (1989). **Engaging Children's Minds: The Project Approach**. Norwood, NJ: Ablex.

[3] Chard, S.C. (1992). **The Project Approach: A Practical Guide for Teachers**. Edmonton. Alberta: University of Alberta Printing Services.

[4] Barrows, H.S. (1996). **Problem–based learning in medicine and beyond:** A brief overview in Bringing problem–based learning to higher education: Theory and practice, 3-12. San Francisco: Josey-Bass.

[5] Solomon, G. (2003). **Project–Based Learning: A Primer.** Technology and Learning, 23, 20-27.

[6] Albanese, M. A. & Mitchell, S. (1993). **Problem–based learning**: A review of literature on its outcomes and implementation issues. Academic Medicine, 68, 52-81.

[7] Vernon, D. T. A. & Blake, R. L. (1993). **Does problem-based learning work?** A meta-analysis of evaluation research. Academic Medicine, 68, 550-563.

[8] D'Intino, Robert S., Weaver, K. Mark, Schoen, Edward J. (2007). **Integrating a Project Based Learning Model into the Entrepreneurial University**. NJ: Rowan University

[9] Macías-Guarasa, J. Montero, J.M., San-Segundo, R., Araujo, A. & Nieto-taladriz, O. (2006). "A Project-Based Learning Approach to Design an Electronic Systems Curricula". In **IEEE Transactions on Education**.

[10] Hedley, M. (1998). "An undergraduate microcontroller systems laboratory". In **IEEE Transactions on Education**, 41, 345.

[11] Ambrose, S.A. & Amon, C.H. (1997). "Systematic design of a first-year mechanical engineering course at Carnegie-Mellon University". **Journal of Engineering Education**, 86, 173-182.

[12] Harris, J.G. (moderator) (1994). Journal of Engineering Education round table: "Reflexions on the grinter report". **Journal of Engineering Education**, 83, 69-94.

[13] Ryser, G. R., Beeler, J. E. & McKenzie, C. M. (1995). "Effects of a Computer-Supported International Learning Environment (CSILE) on students' self-concept, self-regulatory behavior, and critical thinking ability". **Journal of Educational Computing Research***,* 13, 375-385, 1995.

[14] Pellegrino, J. W., Hickey, D., Heath, A., Rewey, K. & Vye, N. J. (1992). **Assessing the outcomes of an innovative instructional program**: The 1990-1991 implementation of the "Adventures of Jasper Woodbury". Nashville, TN: Vaderbilt University, Learning Technology Center.

[15] Means, B. & Olson, K. (1995). **Technology and education reform**. Tech. Rep. ORAD 96-1330. Washington, D.C: U.S. Government Printing Office.

[16] Coley, R.J., Cradler, J. & Engel, P.K. (1996). **Computers and classrooms: The status of technology in U.S. schools** (Policy information report). Princeton, NJ: Educational Testing Service.

[17] **ICTSC** (Information and Communication Technology Skills & Careers). Online at http://www.career-space.com [last accessed February 2005]

[18] Cheang, B., Kurnia, A., Lim, A. & Oon, W.C. (2003). **On automated grading of programming assignments in an academic institution**. Computers & Education, 41, 21-131.

[19] Kurnia, A., Lim, A. & Cheang, B. (2001). **Online Judge. Computers & Education**, 36, 299-315.

[20] Korhonen, A., Malmi, L., Nikander, J. & Tenhunen, P. (2003). Interaction and Feedback in Automatically Assessed Algorithm Simulation Exercises. **Journal of Information Technology Education**, 2, 241-255

[21] Baillie-de Byl, P. (2004). An Online Assistant for Remote, Distributed Critiquing of Electronically Submitted Assessment. **Educational Technology & Society**, 7. 29-41

[22] Nieto-Taladriz, O., Araujo, A., Fraga, D., Montero, J.M., Izpura, J.I. (2004). Antares: A synergy between University Education and Research, Development and Technology Innovation Groups, **Proceedings of the Colloquium on Higher Education of Electronics,** 2004, pp. 23-26.

[23] Zlotnik, A., Montero, J.M. (2007). “DRAC (Distributed Remote ACcess System): An On-line Open Source Project-Based Learning Tool”, **ICL**, September 26-28, 2007, Villach, Austria. Oral.