Int. J. Engng Ed. Vol. 22, No. 5, pp. 1053–1062, 2006
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Methodology for the Analysis of Instructors' Grading Discrepancies in a Laboratory Course*

JUAN M. MONTERO, RUBÉN SAN-SEGUNDO, JAVIER MACÍAS-GUARASA, RICARDO DE CÓRDOBA and JAVIER FERREIROS
Dept Ingeniería Electrónica. B-110, ETSI Telecomunicación, Ciudad Universitaria s/n, 28040-Madrid, Spain. E-mail: juancho@die.upm.es

This paper introduces a new methodology to analyze grading discrepancies in PBL software-engineering courses with a high student-to-faculty ratio. The methodology is based on a quantitative analysis of the more relevant software features according to the grades assigned by each instructor. In order to reduce the discrepancies detected, a new grading consensus has to be built, and automatic analysis tools must assist the instructors when grading. The evaluation of the methodology in two academic years revealed a 62% reduction of the grading discrepancies, achieving an average inter-instructor discrepancy of 0.10 in a scale from 0 to 1.

Keywords: grading discrepancies; software quality analysis; biases in assessment; project-based learning; massive laboratories; automatic analysis of grades.

INTRODUCTION

PROJECT-BASED LEARNING (PBL) fosters student initiative and increases student involvement in the learning process when compared to traditional techniques in technical and engineering courses [1–4]. Students' acquisition of knowledge and habits is greater in PBL than in traditional techniques, especially when new technologies support the PBL technique. These new technologies are focused on dealing with a higher teaching workload (in management and coordination) and the more difficult evaluation process that is required in this case.

When a relatively large number of instructors (7–10) have to grade a huge number of student teams (around 200) in an engineering PBL course, some bias in assessment (or grading discrepancies) can appear. The main reasons for these discrepancies are the following:

- When student teams develop complex systems involving multidisciplinary knowledge (communications, control, user interfaces . . .), as these systems are very difficult and time-consuming to evaluate, this increases the risk of grading differences between instructors.
- In the grading process, not only functional aspects should be evaluated but also non-functional ones such as: software quality, flexibility for improving the functionality, ease of maintenance . . . However, these non-functional aspects are difficult to measure in an objective way and the evaluation depends strongly on instructor experience.
- In a PBL course with a high student-to-faculty ratio (which could be 50 to 1, or even higher), the number of projects to assess is too large for the available advising hours, so the usual procedure adopted is to have a large number of instructors involved in the course. Nevertheless, the larger the number of instructors, the higher the risk of grading discrepancies. This fact is especially relevant when the grading workload is large and does not allow each project to be assessed by several instructors.
- When assessing multidisciplinary projects, instructors with different backgrounds can focus on different aspects of the project and they can use different grading criteria.

These discrepancies have traditionally been analyzed by calculating and comparing the mean and variance of the grades of each instructor [5]. As will be shown throughout this paper, this solution does not consistently take into account the fact that the quality of the subsets of projects graded by different instructors can vary greatly from one subset to another. In addition, the mean-and-variance strategy does not detect those cases where the mean and variance of grades are similar, but the evaluation criteria are rather different.

As an alternative to this, the instructors could be provided with 'standard samples' of project reports for assessment; if, at the time of assessment, the expected grading scores were known by the researchers (but not by the instructors), this could be another method for detecting bias. This method has proven useful in other contexts [6], but it seems to be impractical in the evaluation of multidisciplinary projects in PBL courses, where

* Accepted 20 November 2005.

more than one hour of work is needed for a thorough assessment of just one team.

For solving these grading discrepancies, the most extended solution is to consider a small number of instructors to assess the same subset of projects [7, 8]. Generally, the independent assessments are arithmetically averaged to get the final numerical grade, but in the literature one can find other strategies for combining the instructors' grades [9, 10].

In [8], Woods presents an iterative processing algorithm to detect and correct biases in the assessments performed by several instructors. From our point of view, this strategy does not provide a complete solution in PBL courses involving multidisciplinary projects and a high student-to-faculty ratio, because:

- When only a small number of instructors assess a complex project, only the criteria of these instructors are accounted for in the grading process, setting aside the criteria of the remaining instructors.
- If a number of instructors assess only a subset of the projects, they do not have a global view of all students, so they cannot take into account the variation in quality between different subsets of projects.

A solution to the assessment problem could be achieved by splitting the proposed projects into several non-overlapping and equally-sized subprojects. This way, each instructor could evaluate the same subproject for all students, diminishing the grading discrepancies. Nevertheless, this solution would require a large amount of work (especially in engineering PBL courses) and it is not feasible because of time limitations. Because of this, it is necessary to develop new detailed studies, supported by automatic tools, in order to reduce the amount of time necessary for the analysis.

In the last few years, there have been several works focused on developing automatic tools to assist instructors in their grading work [11–13]. Generally, these tools are applied to non-PBL software assignments or to circuit simulations. In these cases, it is possible to verify the software or hardware functionality in a fully automatic way, using test vectors (a test vector is a set of inputs a system can receive and the corresponding correct outputs the system must generate as a response to these inputs; if the output of the system is correct for every input, then the system is functionally correct). When someone wants to develop similar tools for engineering multidisciplinary PBL courses, the following problems can arise:

- In an engineering project, planned for a semester, students have to develop a complete communication or control system. These kinds of systems need interactive verification of the functionality that is very difficult to automate by means of test vectors.
- In PBL, one of the targets is to foster student initiative and creativity. Because of this, the final systems can exhibit important differences in functionality from team to team; therefore, it is very difficult to make an automatic verification based on standard test vectors. In our course, a relevant portion of the functionality is due to student creativity and this portion can account for more than 15% of the total numeric grade.
- Finally, an evaluation process based on test vectors focuses only on the functionality or the response time. However, this evaluation does not take into account other non-functional aspects of software development such as the structure of the developed system, the management of available resources, or the scalability of the proposed solution.

This article describes a new methodology to perform a detailed analysis of the grading discrepancies and their causes, fully supported by automatic tools. These tools provide a fine-grained analysis of an engineering PBL course where the grading process is costly and time consuming.

DESCRIPTION AND CONTEXT OF THE COURSE

This methodology has been used in a course called LSED (Laboratory of Digital Electronic Systems) of the Department of Electronic Engineering 1 at the Telecommunication Engineering School 1, of the Technical University of Madrid (UPM) 1. This Department is also responsible for several courses focused on the design of electronic systems such as Digital Electronic Systems (taken in the 5th semester, this is a theoretical course on the same microprocessor as LSED (Motorola 68000) and a common set of peripheral devices) and Laboratory of Electronic Circuits (in the 5th semester, on the design and implementation of analog and digital electronic subsystems). In the first and second semester, students must pass one course and one laboratory on standard programming issues, based on the Java language.

LSED is a mandatory laboratory the students take in their 6th semester. There are about 400 students attending each year, and it involves a number of instructors ranging from 7 to 10. The students, grouped in teams of two, have to design, build, test, and document a complete multidisciplinary microprocessor-based system (including some hardware but predominantly software).

The starting point is a written description of the system to be implemented (about 30–40 pages). It includes the functional specifications and requirements of the system, a modular description of the system and the main subsystems, some guidelines for the implementation (a proposal for a basic software architecture) and a tentative planning schedule to help students on how to manage the available laboratory sessions in order to achieve the objectives on time.

The students must complete the analysis of the system (the initial specification is always incomplete) and they must make the design, implementation, tests and associated documentation. The target system changes every year and the students must develop a fully functional prototype. Some of the specifications are open to student creativity. In order to achieve the maximum grade, the students must implement some optional functionality improvements on the basic proposed system; these improvements can account for more than 15% of the total score, as stated above.

The evaluation process of each team of students is comprised of two steps:

- The first is the evaluation of intermediate reports during the semester. These reports help instructors verify the evolution and authorship of the work.
- The second is the final evaluation, based on the complete documentation of the system and an oral examination. The instructors must verify that the prototype follows the specifications of the assignment, and they must make individualized questions to determine the capacity of each student to explain the obtained results, to determine his/her degree of involvement in the development of the system, etc. Other factors evaluated are: writing quality, skills for oral communication and teamwork, etc.

Typically, the proposed project is a simplified version (both economically and in terms of development effort) of a consumer system. In 2002–2003, the proposed system was a talking calculator based on a Motorola 68000 microprocessor. The system was able to add, to subtract and to multiply the numbers a user typed on a matrix keyboard, and it was able to read out loud the operators and operands through a loudspeaker, as the user pressed the keys (without missing keystrokes or degrading the voice quality. During the 2003–2004 academic year, the students had to implement a chat system based on an infrared link and a Motorola 68000. The user typed a message on the matrix keyboard of one of the chat terminals, in a several keystrokes-per-symbol fashion (as in mobile SMS phones); the typed message was transmitted to the other chat terminal that displayed the message. The infrared communication was based on a simple serial protocol.

METHODOLOGY DESCRIPTION

The assessment methodology proposed in this paper is made up of three main steps:

- *A quantitative analysis* of the relationship between the assigned grades and certain software features (using automatic tools) in order to determine which features characterize high-quality software. In this analysis, the relevance of each feature is obtained by computing the Pearson correlation between the feature's values that were automatically measured in each student program and the numeric grades assigned by the instructors to these programs. As a result of the analysis, a general vector of feature relevance is obtained. Section 3.1 will describe more in detail the computation of the relevance of each feature.
- *A discrepancy analysis.* In this step, the same numeric analysis of feature relevance is carried out only for the grades assigned by each instructor, obtaining one specific relevance vector per instructor. The differences between the general relevance vector and specific ones or between each pair of instructors' vectors, reveal the criteria discrepancies. Section 3.2 will provide the mathematical description of this analysis.
- *A discrepancy reduction task.* In order to reduce these discrepancies, a new detailed evaluation criterion must be achieved with the consensus of all instructors. In this process, the general relevance vector (computed in the first step) should be considered as the starting point for discussion and agreement. In addition to this new criterion, automatic tools must assist instructors when grading. Section 3.3 will provide the description of the discrepancy reduction we have obtained when using this methodology.

Quantitative software feature analysis

It is not easy to make a precise definition of software quality, although experienced instructors are able to estimate software quality and classify programs in terms of it. To avoid the difficulties of an explicit formal definition, one can use the final grades assigned by instructors in previous academic courses as a source of expert knowledge 1.

There are many quantifiable features that can be computed on a program which could be related to software quality. In this step of the methodology, a large set of characteristics is gathered and their relevance is computed according to the grades of the instructors, using the automatic tools implemented as described in 1. Up to 48 basic features of programs have been analyzed:

- *The use of CPU resources:* that is, the efficient use of the data and address registers, the set of available instructions, the diversity of addressing modes . . .
- *The data structures used by the programmer:* the number of declared variables, the number of constants, tables or messages . . . which help the programmer provide an elegant solution
- *The structure of the code:* such as the number and average length of the subroutines (or the interrupt service routine), the average number of exit points and entry points per subroutine, the average and the maximal length of a jump . . . which make the programming code easier to read and maintain.
- *The comments inserted in the programming code:* line comments, block comments . . . which also improve the readability of the code.

Table 1. A subset of the data used for computing the relevance of two features F1 and F2, according to the grades assigned by the instructors

Student Team Code	Feature values			Grade	Instructor
	F1	F2	...		
1	7	40	...	75	I5
2	8	45		70	I5
3	25	15	...	95	I6
4	20	21		92	I7
...	...	...	...	...	...

Using the data collected in the 2002–2003 academic year, we computed the relevance of each feature as the Pearson correlation coefficient between the numeric values of each feature and the numeric grades assigned by the instructors. This way we obtained one coefficient or relevance value for each feature. The Pearson formula is:

$$C_f = \frac{P \cdot \sum_{p=1}^{P} F_{fp} \cdot G_p - \left(\sum_{p=1}^{P} F_{fp}\right) \cdot \left(\sum_{p=1}^{P} G_p\right)}{\sqrt{\left(P \cdot \sum_{p=1}^{P} F_{fp}^2 - \left(\sum_{p=1}^{P} F_{fp}\right)^2\right) \cdot \left(P \cdot \sum_{p=1}^{P} G_p^2 - \left(\sum_{p=1}^{P} G_p\right)^2\right)}} \quad (1)$$

where coefficient C_f is the relevance coefficient of feature f, F_{fp} is the value of this feature f that was automatically measured in program p, G_j is the grade assigned to this program p by the instructors and P is the total number of programs that have been graded and analyzed. This coefficient C_f measures the relative importance of each feature f: it tells you which values of feature f characterize the best programs, and which ones are more common in the worst ones.

This relevance coefficient ranges from -1 (the student programs with a higher value in the feature have always received the lower grades and vice versa) to $+1$ (the student programs with a higher value in the feature have always received the higher grades and vice versa). When the absolute value of the relevance is close to 0, the feature and the grades are uncorrelated: the feature is not relevant in terms of software quality, so we can discard it.

Table 1 shows some feature values and the grades of a few student programs, and the instructors that graded these programs (in a 0–100 scale). F1 is the feature called 'Number of subroutines included in each program' and F2 is 'Average length of jumps in each program'. The table shows that the first student team included only 7 subroutines in their program and their jumps averaged 40 lines per jump. When comparing the whole sequence of values of F1 (7, 8, 25, 20, . . .) and the corresponding numeric grades assigned by the instructors (75, 70, 95, 92, . . .), we find a positive correlation between feature F1 and the grade: the students with a higher F1 have generally obtained a higher grade, and the students with a lower F1 have generally received a lower grade. However, the behavior of feature F2 is just the opposite (negative correlation): the higher the F2 value, the lower the grade. Table 2 shows the relevance of the most important features according to the final numeric grades. According to Table 2, F1 and F2 are two of the most relevant features for identifying high-quality programs.

The Pearson coefficient does not depend on the mean or the range of the feature values (this is an important property for a feature relevance estimator because, otherwise, the relevance of different features could not be compared), it does not depend on the average numeric grade (otherwise, as the proposed project changes from year to year, the results could be affected by a certain bias in one specific academic year, and the relevance vectors of two academic years could not be compared), and it is also invariant to any linear transformation of them.

When computing the relevance of a software feature according to the final grades assigned to the student programs, we are gathering information on how the values of these features affect the grades. From this computation, we can extract numeric information on which are the most relevant aspects of a software program according to the average instructor and, so, what has been the general grading criterion of the instructors.

Discrepancy analysis

From the software feature analysis performed in the previous section, we obtained a relevance value C_f for each feature considered in the study. In this section, we describe how to use this feature relevance values to analyze the evaluation discrepancies between instructors.

Table 2. Relevance of the main features in 2002–2003.

Feature category	Feature name (f)	Feature relevance C_f
CPU resources	Complex addressing modes	0.19
	Number of instructions	0.53
Data structures	Number of complex data structures	0.19
Code structure	Number of subroutines (F1)	0.48
	Number of exit points per subroutine	−0.15
	Number of interlaced subroutines	−0.26
	Average length of jumps (F2)	−0.32
Comments	Number of commented lines	−0.18
Lines of code	Number of lines of code	0.55

Table 3. Discrepancy between each Instructor Criteria Vector and the General Criteria Vector

	I5	I6	I7	I8	I9	I10	I11
Discrepancy between ICVs and GCV	0.17	0.23	0.21	0.24	0.15	0.33	0.58

By compiling the feature relevance values in a vector, we obtain the *General Criteria Vector* (GCV):

$$\mathrm{GCV} = (\mathrm{C_1}, \mathrm{C_2}, \ldots, \mathrm{C_f}, \ldots, \mathrm{C_N}) \qquad (2)$$

where $\mathrm{C_f}$ is the relevance of feature f and N is the number of automatically measured features considered in the study. It is important to notice that, as mentioned above, the components of this GCV have been computed by taking into account all the software programs graded by the instructors. The values of the main components $\mathrm{C_f}$ of this vector were presented in Table 2.

For every Instructor i, it is also possible to define a specific *Instructor Criteria Vector* ($\mathrm{ICV_i}$) by computing the relevance of every feature only on the software programs graded by that instructor:

$$\mathrm{ICV_i} = (\mathrm{C_{i1}}, \mathrm{C_{i2}}, \ldots, \mathrm{C_{if}}, \ldots, \mathrm{C_{iN}}) \qquad (3)$$

$\mathrm{C_{if}}$ is the relevance of feature f according to the grades assigned by Instructor i to the subset of programs that were graded by him or her, and **N** is, again, the number of features.

For computing the discrepancy between one instructor and the others, we propose Equation 4, where $PC_i(ICV_i, GCV)$ is a new Pearson coefficient that now correlates vector $\mathrm{ICV_i}$ (ICV of Instructor i) and vector GCV, in order to find out whether they are similar or not. As the correlation ranges from -1 to $+1$, the *discrepancy measurement* ranges from 0 (no discrepancy) to +1 (highest discrepancy).

$$Discrepancy(ICV_i, GCV) = \frac{(1 - PC_i(ICV_i, GCV))}{2} \qquad (4)$$

No discrepancy means that both the most relevant features and the least relevant ones according to Instructor i are exactly the same as those of an average instructor. If the discrepancy were as high as +1, the most relevant features for Instructor i would be the less relevant ones according to the general criterion and vice versa. An intermediate value (0.5) shows that half of the more relevant (or irrelevant) features according to Instructor i were considered as irrelevant (or relevant) by the others (or, the other way around, only half of the relevant and irrelevant features according to the general criterion of the instructors were also regarded the same way by Instructor i).

Table 3 shows the discrepancies between $\mathrm{ICV_i}$ and GCV for the instructors involved in the 2002–2003 academic year.

The main conclusions that can be drawn from these results are:

- As the discrepancy ranges only from 0 to 1, the discrepancies are rather high. These figures reveal important differences in the evaluation criteria applied by each instructor.
- In Table 3, one can distinguish several types of instructors:
  - In 2002–2003, I10, I11, I7 and I8 were new instructors in this complex PBL course. This fact can explain why I10 and I11 exhibit important discrepancies.
  - The remaining instructors (I5, I6 and I9), although having greater experience in this course, had evaluation criteria that also exhibited important differences.
  - The reasons for these differences can be found in the different attention that some instructors have paid to some software quality features that were considered very relevant (or absolutely irrelevant) by most of the instructors. For example, the relevance of the feature called 'number of subroutines' for I6 is very high (0.40), but for I11 is irrelevant (0.03). In this case, I11 has not paid enough attention to this aspect of software quality.

These results suggest an important warning: there are significant discrepancies in the evaluation criteria applied by instructors, which is not a desirable situation. Because of this, instructors decided to make a significant effort to address this problem by consensus in the following academic year.

As mentioned in the introduction, the grading discrepancies have been traditionally analyzed by computing the average grade assigned by each instructor. Table 4 shows these average grades per instructor in the 2002–2003 academic year.

From this analysis, one could think that I5, I7 and I8 are the instructors that show the higher discrepancies. However, we need to take into account that the distribution of the students among the instructors was uneven (although these instructors had to grade a randomly selected subset of students, the sample is too small to guarantee an even distribution):

The highest grade (100 in our grading system,

Table 4. Average grades given by every instructor in the 2002–2003 academic year, on a 0 to 100 scale

	I5	I6	I7	I8	I9	I10	I11	GLOBAL
Average grade in a 0 to 100 scale	72.6	75.4	79.9	80.6	75.7	73.5	74.9	76.0

corresponding to the top 10% students) is assigned by consensus of all instructors: none of the students that were examined by I5 achieved this highest grade after the consensus. However, the percentage of students achieving the highest grade was as high as 15.2% for other instructors.

The percentage of students that failed the examination (also after a general consensus) varies from as low as 0% (for the students examined by I7) to as high as 10.0% (I5).

The percentage of students that implemented an improved system (not just the basic one proposed in the assignment) also varies greatly from the instructors that examined only a few improved projects (38.5% for I11 and 50% for I10) to the instructor with more improved systems to grade (85.7% for I7).

Finally, the average grade due to these improvements is 14.7 for I8, but only 10.0 for I10 (a consensus was also built upon how to grade these improvements). These objective figures show that I8 graded some of the best systems, systems that were much better than those graded by I10.

These figures could explain that I5 had to assign the lowest average grades, and why I8 and I7 assigned the highest ones, but no significant grading discrepancy has been detected (as compared to GCV). However, the results for I10 and I11 may not be as coherent, because they seem to have examined the worst students. Average grades of I10 and I11 are very close to the global average grade, but their criteria have been very different from the rest (Table 4). This problem may be frequent in PBL courses where instructors can focus on different aspects when grading multidisciplinary projects

With the discrepancy analysis strategy proposed in this paper (supported by automatic tools), it is possible to make a detailed analysis to detect the real causes of these disagreements (not just detect disagreements). These differences not only appear in the average grade but also in the relevance or importance assigned to each feature by one instructor. It is possible to analyze which coefficients (feature relevance) are affecting the difference between criteria vectors in a greater way.

Achieving a consensus on a new detailed evaluation criteria

After the analysis of 2002–2003 academic year, the instructors analyzed the results and suggested several solutions to this problem.

The first was to build a new consensus on evaluation criteria: what features should be judged more important (as providing the best information related to software quality) and what is their relevance. For this analysis, instructors accounted for the GCV (General Criteria Vector) obtained from the results. This GCV gathers the contribution of the complementary backgrounds of every instructor. In their collective opinion, the most important features were the number, length and structure of the subroutines, the use of

Laboratory of Digital Electronic Systems
EVALUATION SHEET

Instructor:....	**Team Code:....**	**Date:...**	
Written report	Style and contents ...		**15**
Oral examination ...			**15**
...			
Software			**25**
Quality	Comments: block, line		3
	Subroutines: size, structure, exit points...		7
	Length of Interruption and main routines		4
	Use of constants and no magic number		3
	CPU resources: addressing modes...		4
	Data structures: tables, messages		4
Hardware ...			**5**
The prototype is working properly ...			**25**
Design improvements / initiatives			
	Liquid Crystal Display (basic or advanced)		8
	Additional keys: Delete, uppercase, etc.		2
	Infrared link (<1metres, >1metres or >2metres)		8
	Menus and software reconfiguration		2
	Buffer size counter (with additional hardware)		5
	Other improvements (must be detailed)		
Total score			**85**

Fig. 1. A simplified example of a detailed evaluation sheet of LSED (Laboratory of Digital Electronic Systems).

Table 5. A sample of automatic report on software quality

Team code: VT-11		
Feature Name	Measured feature value	Mean ± Deviation
Total number of lines in the file	934	804 ± 123.5
Lines of code—with actual code	490	460.8 ± 25.65
Software Quality Feature Name	Measured feature value	Mean ± Deviation
Relative use of indexed addressing mode	0.19	0.25 ± 0.24
Relative use of predecrement/postincrement mode	0.73	1.62 ± 0.59
Number of different instructions	47	41.2 ± 2.85
Number of complex data structures	11	7.3 ± 2.85
Number of constants	35	14.5 ± 19.47
Use of 'magic numbers'	0	0.04 ± 0.04
Length of Interruption/Main routines	67	82 ± 16.15
Average subroutine length	17.65	18.65 ± 0.95
Number of subroutines	27	21.3 ± 4.27
Number of interlaced subroutines	0	0.76 ± 0.72
Length of the longest subroutine	54	91.3 ± 15.2
Number of RTS per Subroutine	1	1.01 ± 0.03
Average jump length	9.67	13.86 ± 3.98
Maximum jump length	41	62 ± 19.95
Relative number of comments	0.32	0.3 ± 0.03
Relative number of lines with comments	0.02	0.37 ± 0.32

complex data structures or addressing modes, and the number of comments in the code.

Secondly, instructors defined a detailed evaluation sheet (0) by consensus. The global evaluation was ranged in a 0–100 scale but included many evaluation items with smaller scales (0–3, 0–5, etc). The granularity of these evaluation criteria increases the objectivity of the evaluation process. This sheet or rubric is offered to all the instructors as a guide (not as a rule) for the grading process.

APPLICATION TO A NEW ACADEMIC COURSE AND DISCUSSION

In the 2003–2004 academic year, the instructors used the automatic tools developed in the previous analysis to supervise the students though the course (for detecting bad programming habits or errors). Two analyses of the students' software were made and two brief reports on the software characteristics of each team (Table 5) were sent to instructors, including the relevance of each characteristic (as defined by consensus the previous year). The first analysis was performed after six weeks; the target was allowing the instructor to give feedback to the students on their coding abilities. The second analysis was carried out after ten weeks, at the end of the semester, just before grading: then the target was to help the instructor during the evaluation process. These two analyses were possible with a very low workload, thanks to the automatic tools developed in the previous academic year.

In Table 6, the discrepancies between ICVs (Instructor Criteria Vectors) and GCV (General Criteria Vector) for 2002–2003 and 2003–2004 academic years are shown.

From these results, one can conclude that:

- First, in 2003–2004 there were smaller discrepancies in the evaluation criteria when compared to 2002–2003 (a 62% decrease, from 0.27 to 0.10). In the second year there were no differences greater than 0.50 and four instructors were below average. The standard deviation was also lower (0.15 versus 0.04) and some significant differences seen in the previous year were avoided in 2003–2004.
- Regarding the instructors that were involved in both academic years, there is a reduction in their discrepancy to the general criteria. This reduction was enabled by the consensus achieved. In addition to this consensus, the automatic analysis of software quality also helped to reduce the

Table 6. Discrepancies between each ICV and GCV for 2002–2003 and 2003–2004

	Average discrepancy	I1	I2		I4	I5	I6	I7	I8	I9	I10	I11
ICVs–GCV (2002–2003)	0.27					0.17	0.23	0.21	0.24	0.15	**0.33**	**0.58**
ICVs–GCV (2003–2004)	0.10	0.09	0.11	0.11	0.03	0.15	0.07	0.07	**0.18**			

Table 7. Discrepancy matrix of the instructors' grading vectors in 2003–2004

2003–2004	I1	I2		I4	I5	I6	I7	I8
I1	—	0.23	0.25	0.11	0.31	0.18	0.18	0.29
I2	0.23	—	0.16	0.20	0.26	0.16	0.11	0.34
	0.25	0.16	—	0.17	0.21	0.19	0.18	0.24
I4	0.11	0.20	0.17	—	0.19	0.10	0.15	0.18
I5	0.31	0.27	0.21	0.19	—	0.17	0.19	0.26
I6	0.18	0.16	0.19	0.10	0.17	—	0.14	0.14
I7	0.18	0.11	0.18	0.15	0.19	0.14	—	0.23
I8	0.29	0.34	0.24	0.18	0.26	0.24	0.23	—

grading discrepancies since:

- Software reports included the main feature values of each team with the relevance of each feature (in the general criteria). This method was an efficient way of reminding the instructors of the grading consensus.
- Every report also included the average value of each software feature for all the teams involved in the course. This information provided the instructor with a global view of all the students' teams, and this view allowed the instructor to put the evaluated team in the context of the whole set of teams.
- The intermediate analysis allowed instructors to test the automatic tools before the final grading, increasing their confidence in them. This confidence allowed integrating quantitative measurements (obtained with the automatic tools) in the final grading process. These measurements helped increase both objectivity and uniformity in the evaluating criteria.

The GCV (General Criteria Vector) is a very good reference for comparing the ICVs (Instructor Criteria Vectors): it gathers the contribution of the complementary backgrounds of every instructor, and it is very easy to obtain (it can be computed automatically). These are two important advantages when comparing this method to ones that require several instructors to assess the same set of projects [11, 13]. In such a system, the effort is multiplied by the number of instructors and, when a smaller subset of instructors is considered, it is not possible to gather the opinion of all instructors without increasing the effort exceedingly.

By increasing the detail of the analysis, it is possible to compute the discrepancy between each pair of instructors. For computing the discrepancy between two instructors, one can use Equation 5 where $PC_{ij}(ICV_i,ICV_j)$ is the Pearson Correlation between ICVs of Instructor i and Instructor j.

$$Discrepancy(ICV_i, ICV_j) = \frac{1 - PC_{ij}(ICV_i, ICV_j)}{2} \quad (5)$$

Tables 7 and 8 show the discrepancies between one instructor and another one in the two academic years 2003–2004 and 2002–2003. The average discrepancy between pairs of instructors decreases 54% (from 0.44 in 2002–2003 to 0.20 in 2003–2004). A t-Student statistical significance test shows that only the criteria vectors of two instructors in 2002–2003 were significantly and positively correlated ($p < 0.01$). However, in 2003–2004 there is a set of five instructors with such significant correlation (their figures are highlighted in Table 7, I4, I5, I6 and I7.

In Table 7, this set (or sub-matrix) is typed in bold font within shadowed cells. In 2002–2003 (Table 8) it is not possible to identify a similar set. In addition to this, there is no pair of instructors with a discrepancy that is lower than 0.20 (average discrepancy in 2003–2004). On the other hand, in 2002–2003 there are seven pairs of instructors with a discrepancy higher than 0.50 (negative correlation). These results also show an important discrepancy reduction between the 2002–2003 and 2003–2004 academic years.

The proposed discrepancy measurement only accounts for the Pearson correlation between two vectors (Equations 4 and 5). This correlation provides information about the similarity in the evolution of two number sequences (feature relevance sequence) not taking into account a possible offset between both vectors. In order to consider this factor, a similar analysis was carried out by

Table 8. Discrepancy matrix of the instructors' grading vectors in 2002–2003

2002–2003	I5	I6	I7	I8	I9	I10	I11
I5	—	0.36	0.27	0.28	0.31	0.55	0.62
I6	0.36	—	0.49	0.45	0.47	0.39	0.66
I7	0.27	0.49	—	0.20	0.30	0.61	0.47
I8	0.28	0.45	0.20	—	0.38	0.58	0.43
I9	0.31	0.47	0.30	0.38	—	0.35	0.60
I10	0.55	0.39	0.61	0.58	0.35	—	0.66
I11	0.62	0.66	0.47	0.43	0.60	0.66	—

computing the Euclidean Distance between ICVs, obtaining two matrixes similar to Tables 7 and 8.

These results show an average 20% distance reduction: from 0.095 in 2002–2003 to 0.075 in 2003–2004. In the 2003–2004 matrix, it is possible to identify the same cluster of instructors (I4, I5, I6 and I7), with a distance lower than 0.075 (average for this academic year).

As is shown, for all the measurements analyzed in this paper, discrepancies in the evaluation criteria have been strongly reduced. The measures adopted by the instructors have had a significant impact on the results.

CONCLUSIONS

In this paper, the authors propose a new methodology for carrying out a detailed analysis of grading discrepancies and their causes, including a strategy to reduce them.

The proposed methodology is especially designed for PBL courses in software engineering, where the students have to develop a whole system involving multidisciplinary knowledge. The methodology is comprised of three steps:

- A quantitative analysis stage that evaluates the relationship between the assigned grades and certain software quality features, to determine which features are more relevant for predicting the quality of the student programs. In order to do so, automatic software quality analysis tools were developed. As a result of this analysis, a general vector of feature relevance is obtained.
- A discrepancy analysis stage calculates the relevance of software quality features for each individual instructor and it evaluates the discrepancies by comparing each individual vector of feature relevance with the general one. Discrepancy measurements based on the Pearson Correlation and the Euclidean distance are proposed in this case.
- A discrepancy reduction stage, based on two initiatives: the generation of a detailed evaluation criterion with the consensus of all the instructors, and the use of automatic tools to feed back students on their coding abilities and help instructors in their grading process.

In order to evaluate the proposed methodology, the first two steps were carried out during the 2002–2003 academic year. The analysis of the differences between the general relevance vector and every instructor's vector revealed a clear difference of criteria as the cause of grading discrepancies. The third step was carried out in the 2003–2004 academic year, resulting in a 62% reduction of the discrepancies. The cause of this overall improvement was the combined use of the consensus on a new detailed evaluation sheet, and the automatic software analysis reporting tools for feeding back the students and for helping the instructors in the evaluation process.

An additional and more detailed analysis based on computing the discrepancy and the Euclidean Distance between each pair of instructor criteria vectors also showed an important reduction of 54% and 20% in the average discrepancy and average distance respectively. In the 2003–2004 academic year, there were five instructors out of eight with a significantly correlated criteria set, compared to only two instructors out of seven in 2002–2003.

Acknowledgements—We would like to acknowledge the contributions from José D. Romeral (in memoriam), International Computer Science Institute at Berkeley (especially Michael Ellsworth for his comments on this manuscript) and the members of the Departamento de Ingeniería Electrónica at ETSIT-UPM (for their continuous effort to offer high-quality education to our students and for all the fruitful comments and suggestions that made this work possible).

REFERENCES

1. G. Solomon, Project-based learning: a primer. *Technology and Learning*, **23**(6), 2003, pp. 20–27.
2. J. Macías-Guarasa, R. San-Segundo, J. M. Montero, J. Ferreiros and R. de Córdoba, Tools and strategies for improving massive PBL laboratories management, *Frontiers in Education*, 2005.
3. J. G. Harris. Journal of Engineering Education round table: Reflexions on the Grinter Report, *J. Eng. Educ.*, **83**(1), 1994, pp. 69–94.
4. G. R. Ryser, J. E. Beeler and C. M. McKenzie, Effects of a computer-supported intentional learning environment (CSILE) on students' self-concept, self-regulatory behavior, and critical thinking ability. *J. Educational Computing Research*, **13**(4), 1995, pp. 375–385.
5. K. L. Chan, Statistical analysis of final year project marks in the computer engineering undergraduate program, *IEEE Trans. Education*, **44**, 2001, pp. 258–261.
6. G. Engelhard, M. Davis and L. Hansche, Evaluating the accuracy of judgments obtained from item review committees, *Appl. Meas. Education*, **12**, 1999, pp. 199–210.
7. Guide for Evaluator, Marie Curie Fellowships (Individual and Host), Eur. Commission, Brussels, Belgium 2000. ftp://ftp.cordis.lu/pub/improving/docs/mcf_guide_evaluators.pdf
8. R. C. Woods, Iterative processing algorithm to detect biases in assessments, *IEEE Trans. Education*, **46**, pp. 133–141 (2003).
9. N. L. Seed, *The Examination Algorithm*, Univ. Sheffield, UK (1998).
10. D. S. Moore and G. P. McCabe, *Introduction to the Practice of Statistics*, 3rd ed. pp. 632–633. Freeman, San Francisco, CA (1998).

11. J. English and P. Siviter, Experience with an automatically assessed course, *Proc. 5th Annual SIGCSE/SIGCUE ITiCSE Conf. Innovation and Technology in Computer Science Education*, 2000, pp. 168–171.
12. B. Cheang, A. Kurnia, A. Lim and W. C. Oon, On automated grading of programming assignments in an academic institution, *Computers* , **41**, 2003, pp. 121–131.
13. A. Korhonen, L. Malmi, J. Nikander and P. Tenhunen, Interaction and Feedback in Automatically Assessed Algorithm Simulation Exercises, *J. Information Technology Education*, **2**, 2003, pp. 241–255.
14. http://www.die.upm.es
15. http://www.etsit.upm.es
16. http://www.upm.es
17. R. San-Segundo, J. M. Montero, J. Macías-Guarasa, R. de Córdoba and J. Ferreiros, Automatic tools for diagnosis and feedback in a project based learning course, *Frontiers in Education*, 2005.

Juan M. Montero received his MSEE degree and Ph.D. degrees from Technical University of Madrid in 1992 and 2003 (with highest distinction). Currently, Juan M. is associate professor at the department of Electronic Engineering at ETSIT (UPM). His main research interests are related to microprocessor-based systems and Speech Technology.

Rubén San-Segundo received his MSEE degree and Ph.D. degrees from Technical University of Madrid in 1997 and 2002 (with highest distinction). Ruben did two summer stays in The Center of Spoken Language Research (CSLR-CU). From Sep. 2001 through Feb. 2003, Rubén worked at the Speech Technology Group of Telefónica I+D. Currently, Rubén is associate professor at the department of Electronic Engineering at ETSIT (UPM).

Javier Macías-Guarasa received his MSEE degree and Ph.D. degrees from Technical University of Madrid in 1992 and 2001 (with highest distinction). Since 1990 he is a member of the Speech Technology Group and associate professor in the Department of Electronics Engineering in the Technical University of Madrid. He spent six months in the International Computer Science Institute in Berkeley, California. His main research interests are related to Project-Based Learning and Speech Technology.

Ricardo de Córdoba received his MSEE degree and Ph.D. degrees from Technical University of Madrid in 1991 and 1995 (with highest distinction). Since 1993 he has had different teaching positions in the Department of Electronics Engineering in the Technical University of Madrid, and since 2003 he is a professor. He spent six months working as Research Associate in the Department of Electronics in Cambridge University, UK. His main research interests are related to Project-Based Learning and Speech Technology.

Javier Ferreiros received his MSEE degree and Ph.D. degrees from Technical University of Madrid in 1989 and 1996 (with highest distinction). Currently, Javier F. is professor at the department of Electronic Engineering at ETSIT (UPM).