

An Objective Approach for Evaluating Individual Performance in Group Projects

David Coppit
College of William and Mary
Department of Computer Science
Williamsburg, VA
david@coppit.org

Abstract

Group projects are an essential component of many courses. However, accurately assessing the performance of individual students in group work is notoriously difficult. As a result, students often feel that they are not properly rewarded for the effort they expend. In the worst case, some students may take advantage of a poor grading system, exploiting the hard work of their fellow group members. In this paper we present a new approach to grading individuals in group projects. The approach is integrated with the use of a dynamically updated task list for the project. It provides an automatic, quantitative measure of student performance. To evaluate the approach we employed it as part of a software engineering course at The College of William and Mary. Our experience suggests that the approach is flexible, has reasonable overhead, and can scale to group sizes of at least 20 students.

1. Introduction

Group projects are an essential component of many college-level courses. Unlike individual work, group projects emphasize skills such as leadership, communication, modularization of problems, and delegation of duties. In this paper, we focus on group projects associated with software engineering courses, where a key objective is to design and implement a sizeable software system. Arguments have been made that software engineering courses that employ group projects, perhaps involving very large groups, provide a more effective learning experience for computer science students [6].

The problem of evaluating individual students who work in groups has long been recognized as a key problem. Existing methods are prone to manipulation by the students, do not assign accurate grades, or require much work on the part of the instructor. In Section 2 we describe a set of requirements for an effective evaluation technique, and in Section 3 we evaluate previously developed techniques.

In this paper we present an approach for evaluating the contributions of students who work in groups. While the approach is generally applicable, it is best suited for software engineering courses that employ an agile development process. At its core is a web-based task management system that documents and tracks all project-related work. Other than the creation of tasks and evaluation of their satisfactory completion, the system automatically computes a quantitative evaluation of both the group and the individual students.

This approach has a number of benefits. It is quantitative and fine-grained, allowing instructors to accurately assess the grades of individual students. The approach is also very flexible, since tasks can be created for all manner of project activities. The approach is fairly objective in that

task point values and completion criteria are set before any student signs up for the task. It is also resistant to manipulation. Finally, this approach is very cost-effective, providing the instructor an easy way to track the project work along with student performance.

We applied this approach during the Spring 2004 software engineering course at The College of William and Mary. We present results from this case study, evaluating our approach both in terms of our own requirements for a successful evaluation method, and from the students' points of view, as indicated by survey data collected at the end of the course.

The next section describes requirements for an effective evaluation method, and the following section assesses some traditional approaches in terms of these requirements. Section 4 describes our approach in detail using an illustrative example. Section 5 presents an evaluation of the approach, including the results of our case study. Section 6 concludes.

2. Requirements

There are a number of key requirements for any method for evaluating students who work in groups. In this section we elaborate upon the criteria described in the excellent paper by Hayes et al [3]. We also characterize certain student behaviors that affect evaluation methods.

Perhaps the most important requirement is that the system be fair. It should be quantitative and as objective as possible, with little grader bias or arbitrariness. The system should not be easily manipulated by students in order to achieve a particular grade for themselves or others. Students should feel that they receive the grade that they earn—good or bad.

The system should also be consistent. Ideally, it should be independent of the type of project. It should be robust in the face of uncontrollable factors such as project size and scope, changing technology, etc. It should also be independent of the quality of the class. It should enable the grading of different types of work, such as documentation, team leadership, software implementation, testing, and version management.

The system should be open to scrutiny, and provide students with timely and understandable feedback. The more responsive the system is, the more quickly students and the instructor can receive feedback on the student's progress, and take any necessary action. The system should also reflect the educational objectives of the course. Importantly, the system must not require an inordinate amount of work for the instructor. It should scale well to different sizes and numbers of groups. It should be possible to delegate some of the evaluation work to trusted people such as teaching assistants.

The assessment approach should encourage the behaviors that lead to effective teamwork. Students should be allowed to specialize their expertise in certain aspects of the system or certain types of work. At the same time, the instructor should be able to require all students to participate in certain activities for pedagogical reasons. The evaluation approach should encourage high quality work, and instill a sense of pride and accountability in the students for the work that they do.

On the other hand, certain unproductive behaviors should be discouraged by the grading approach. The following descriptions provide a useful characterization of such behaviors:

Hitchhiker (AKA slacker, freeloader): Seeks to maximize their grade while minimizing the amount of work that they must do. This type of student is a common source of much bitterness on the part of his or her group members [5], because the group members must do an inordinate amount of unrewarded work in order to make up for the hitchhiker.

Overachiever: Seeks to do as much work as possible, even beyond that which is required to achieve the highest possible grade. This type of person can make it difficult for other students to contribute in a meaningful way.

Procrastinator: Fully intends to do his or her share of the work, but only immediately prior to a deadline. This makes it difficult for team members to plan work that depends on that of the student—forcing everyone to become a procrastinator.

Underachiever: Does not perform their full share of the work, and is content earning a lower grade (if that grade is properly given).

Dilettante: Becomes involved in many portions of the project, but only in a superficial way, never fully completing their portion of the work. Other students can be overwhelmed or intimidated by the student, choosing not to contribute toward work that is already “taken.”

Tutor: A stronger student who knowingly helps an underachiever or hitchhiker to achieve a higher grade, perhaps by doing the weaker student’s share of the work.

Conspirator: One of a group of students who collude to exploit the grading system to achieve a higher grade.

3. Previous Approaches

We now evaluate previous approaches in terms of the requirements described in the previous section. For additional discussion, see Hayes et. al [3] and Rosen [4].

All individuals get the team grade: This approach is easy to implement, but it is unfair to the students because it allows for much undesirable behavior, especially hitchhiking.

Peer evaluation: In this approach, team members give a grade to each other and possibly themselves. This simplistic approach fails for a number of reasons. Students are often reticent to “rat out” each other. It is too easy for students to collude, giving each other good scores, or “ganging up” on a particular team member. Personality, gender, and race prejudices may also unfairly affect team grades. This approach and its variants all suffer from the problem that they are subjective in nature, and place the responsibility of evaluation on the student and not the professor, who has the experience with which to accurately assess a student’s performance.

Personal evaluation: The most accurate individual assessment method is to personally observe every group activity. This is a time-intensive process that involves attending every group meeting, and inspecting every team member’s work. The obviously requires an inordinate amount of work for the instructor, and does not scale well with the number of groups.

Evidence: A less time-consuming approach is to have students accumulate and submit a “body of evidence” that documents their work. Examples include email, a weekly report, a weblog, or a journal. This approach is more time-consuming for students, and remains fairly qualitative. It is also prone to manipulation as students exaggerate their contribution.

Knowledge assessment: Another approach for evaluating individuals is to assess their knowledge of the group work using pop quizzes, oral examinations, or questions during the project presentation. Professors can usually identify underperforming team members from their vague answers to specific questions. However, this approach is subjective and prone to bias, especially when students have contributed but are unable to adequately express their contribution.

A combined approach: Some authors [1, 3] suggest that a combined approach should be used. For example, written weekly reports can be combined with oral evaluation at project demonstrations in order to assess students. Using a combination of approaches helps to alleviate the problems with individual approaches, but requires more overhead on the part of the instructor.

4. Our Approach

Our approach evaluates students by answering two questions: “How much of the overall work was completed?” and “To what extent did each student do his or her share of the work?” To that

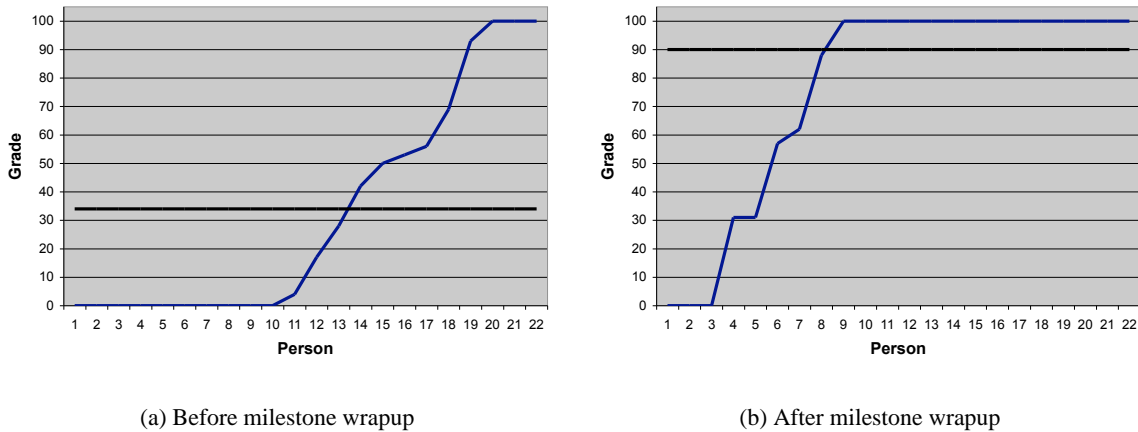


Figure 1. Grades for milestone 3

end, we carefully track the work done by every student in the group, and compute both a group grade and individual grades. These two values are composed to compute the grade for each student.

In this section we describe the mathematics behind our evaluation approach, taking into consideration complexities resulting from the need to ensure that the system is fair. While these complexities can be daunting, the reader should keep in mind that the computations are fully automated by the task management system’s report generator. We use an example to illustrate the computations.

4.1. Overview

The central component of our approach is a web-based task management system. The lifecycle of a task is as follows. First, someone creates the task, along with initial estimates of the difficulty and importance, a description, and completion criteria. Then one or more students sign up for the task, at which point the task status changes from “New” to “In Progress.” The students finish the work, then set the task status to “Waiting for Manager.” The manager then inspects the work, optionally updates the difficulty and importance values, and changes the task status to “Closed.”

As tasks are created, they are assigned a point value that is the composition of the importance, difficulty, and a modifier. Currently we use $importance \times difficulty + modifier$. The modifier allows the person evaluating the completion of the task to adjust for unexpected difficulty, or exceptionally good or bad work. (Standards must be set to ensure that consistent values across tasks.)

Students earn points by completing tasks, dividing them evenly if they work together. The points that they earn count toward their individual grade, which is averaged with the group grade to compute the final grade. Basically, the group grade is the percentage of total group points completed. The individual grade is the percentage of a student’s completed share of those points.

For pedagogical reasons, some work should be done by all students. In this case, the professor can create duplicate tasks, and assign those tasks to each of the students. In order to reduce the task management overhead, additional levels of management can be introduced. For example, students can set the status of a completed task to “Waiting for Team Leader.” The team leader then inspects the work and promotes its status to “Waiting for Manager” for final closing of the task.

4.2. Computing a Fair Individual Evaluation

There are several details which complicate the evaluation method described above. One complication that arises when computing points is the earning of points by non-students. For example, a

student may drop the course after completing some tasks, or a manager may have to intervene and complete some key task which the students are not addressing. In these cases, we recompute the total number of group points, discounting the points associated with those tasks.

We begin with a group consisting of four students, `devel_1`, `devel_2`, `devel_3_over`, and `devel_4`. In addition, there is one student who dropped the course, `devel_5_drop`. The task list is as follows:

ID	Summary	Points [Diff* Priority+Mod]	Points/ Person	Status	Assigned To
1	Improve the appearance of the 3D objects	6 (3*2+0)	6	Waiting for Manager	devel_1
2	[BUG] You can hit the 8-Ball first.	8 (1*4+4)	8	Closed	devel_3_over
3	implementing 'push out' with 9 ball	12 (3*4+0)	12	Closed	devel_1
4	Update top-level build.xml file so that all tests are run.	6 (2*3+0)	6	Closed	devel_4
5	Place Cue Ball	9 (3*3+0)	9	Closed	devel_3_over
6	Make script to do hourly javadoc runs	12 (4*3+0)	6	Closed	devel_1, devel_2
7	Make a script to do hourly jalopy runs	9 (3*3+0)	9	In Progress	devel_4
8	Milestone 2 Bonus	10 (10*1+0)	10	Closed	devel_2
9	[ISSUE TRACKER] Clicking on an issue just refreshes listing	3 (1*3+0)	3	Closed	devel_5_drop
10	Consult the physics tutorial	25 (5*5+0)	25	Invalid	devel_4

These tasks are samples from our use of the system, described in the next section. Tasks 1 and 7 are finished but waiting for team leader or manager verification. Task 8 is a bonus from the previous milestone. The last task was deemed to be invalid. Task 6 was performed by two people, who share the points.

Bonus points given to individual students should also not affect either the group grade or the individual grades for other students. Similarly, invalid tasks do not contribute toward the grade in any way. Thus, the *earnable points* are computed by subtracting all bonus points for all students, points for invalid tasks, and points earned by non-group members.

Next we compute the group grade. The total number of points from the task list above is 100. From this we subtract the 10 bonus points, and 25 points for the invalid task. This leaves 65 points. Of these, 50 points have been earned. This yields a group grade of 77%.

It may be the case that all the issues are 100% complete, but the software is still unfinished. The situation occurs if tasks are not entered into the system for unfinished work. One approach is for the professor to evaluate the system and unfinished tasks, estimating the percentage of requirements that are being tracked by the task system. This estimate is then used to scale the group grade, effectively penalizing the group for not using the system to track all of the required work.

The professor determines that the group is only tracking 95% of the requirements for the finished product. The adjusted group grade is therefore $77 \times .95 = 73\%$.

Next we compute the individual grades. Overachievers complicate the evaluation by “stealing” work from other students. For example, a student may already have a 100% individual score, but a 70% group score. The student decides to help the group (and themselves) by doing extra work. We do not disallow this behavior, but we do need to compensate for the extra points earned by the overachiever which are no longer available to the other students.

The *adjusted earnable points* are computed by subtracting all points earned by overachievers from the earnable points. The adjusted earnable points are the pool of remaining points which must be earned by non-overachievers. We divide this value by the number of non-overachiever students to determine the *target points per person*. The individual grade is then the percentage of points earned (including any bonus points), relative to the target points per person.

With a total of 65 points for the group and 4 students, this means that each student must earn 16.25 points. However, we must compensate for the overachiever “devel_3_over,” who has earned 17 points, which is more than his share. We remove the 17 points earned by this student from the pool of earnable points, yielding an adjusted earnable points of 48, and a target points for each of the non-overachievers of $48 \div 3 = 16$. The effect of the overachiever’s extra work was to reduce the amount of work for the rest of the group by 1 point.

Next we compute the individual grades, adding in any bonus points. Once that is done, we can compute the final grade as the average of the group and individual grades.

Student Name	Points Earned	Bonus Points	Target Points	Individual Grade	Group Grade	Final Grade
devel_1	12	0	16	75	73	74
devel_2	6	10	16	100	73	87
devel_3_over	17	0	16	100	73	87
devel_4	6	0	16	38	73	55

5. Evaluation

In this section we discuss our use of this approach for the Spring 2004 software engineering course at The College of William and Mary. We then discuss the strengths and weaknesses of the approach, and evaluate it in terms of the requirements we described in Section 2.

5.1. Case Study

We implemented this approach using the open source, PHP-based Issue Tracker [7] task management system. We extended the basic functionality to support features we needed, such as the ability to set a manager for each task that is created, support for priority, importance, and modifier values, and support for multiple people per task. We also implemented a custom reporting module that computes student grades.

We used this system as part of an agile development process for a class of 22 undergraduate and 3 graduate students. The graduate students served as managers, maintaining the task list. An innovative aspect of this course was that we treated the entire class as one large group, and allowed the students to self-organize into teams according to activity or module of the system. The details of this approach are presented in another paper [2].

The student project was to design, implement, and document a billiards game. The game was Java-based, and supported several games, including 8-ball and 9-ball. It also supported both a top-down 2D view and a multiple-angle 3D view of the table. We found that the system was very flexible for the project. Managers and students created tasks for activities such as adding a section to the documentation, implementing a periodic reformatting of the code using Jalopy, refactoring code, and creating the project website.

Approximately 400 tasks were entered into the system across three milestones, for a grand total of about 3200 points (1200 points for milestone 1, 1100 points for milestone 2, and 900 points for milestone 3). After each milestone we reset the points and gave exceptional students bonuses for the next milestone. The unadjusted group grades for the three milestones were 99%, 99%, and 90%. These were scaled by 84%, 96%, and 86% respectively because of unsatisfied user requirements.

Figure 1 shows graphs of the individual and group grades prior to and after the end-of-milestone work just before the deadline. The horizontal line represents the group grade. We see that prior to the final effort to finish the milestone, almost half of the students had not performed any work.

Afterwards, more than half had performed their share. A similar pattern occurred at the prior two milestones—a few students would earn their points early, while most earned their points late.

During the course of the semester we made minor changes to make the assessment more fair, resulting in the approach we present here. One important change was to introduce team leaders as an additional level of management to help relieve the work of the managers. We also factored the bonuses out of the computation in order to avoid unfairly penalizing other students. After identifying the problem of the “dilettante,” we modified the system so that it only allowed a student to sign up for at most 2 unclosed tasks at a time. Our scaling of the group grade was in response to the observation that the milestone 1 group grade of 99% did not reflect the quality of the produced software. In order to help prevent procrastination, we also created “mini-milestones” for key tasks so that later work would not be unnecessarily delayed.

5.2. Discussion

From the viewpoint of an educator, we believe that our experiences using the system demonstrate its effectiveness. Students who did not contribute received lower grades, and the group grade was an effective motivator for most of the class. The managers’ impressions of the system were generally positive, in that it provided them with an effective way to managing the group’s work. However, they lamented the fact that points were the only motivator they had for students.

We surveyed the students at the end of the course to assess their impressions of the evaluation approach. Understandably, many students were not happy with having to share the consequences of other student’s lack of contribution, as it caused them to work harder in order to get a better group grade. Others felt that the system created competition and stress, since students had to compete to sign up for the “good” tasks. From our point of view, competition for tasks is good in that it prevents procrastination. Furthermore, we suspect that the definition “good task” varies from student to student. When we asked overachievers why they worked so hard, we found that they wanted to create a point buffer in case tasks were added later during the milestone. Other overachievers stated that they “just wanted to get it done.”

As far as we are aware, the approach we have presented is fair and resistant to manipulation by students. It appears to accurately measure the performance of individuals within the group effort—in the case study we performed, we felt that the computed grades did in fact reflect the grades earned by the students. It is oriented toward measuring deliverables, but its flexible definition of “deliverable” allows the instructor to assess students in many different ways. Students who work hard are primarily rewarded with higher individual grades, and also modest increases in the group grade. Bonuses help reduce the impact of resetting point values at the end of milestones. We believe that our approach is resistant to collusion because managers directly assess the completed tasks. We recommend more aggressive use of mini-milestones to help prevent procrastination.

Our approach also appears to be quite consistent, despite the fact that students performed a wide array of different work. There is less grader bias because the point values and requirements for completion of tasks is set before students sign up for tasks. The approach scales well if one employs additional levels of management. As far as we are aware, this is the first time that such an approach has been used to quantitatively evaluate a large group of 22 students.

Although the details of the grading system are somewhat complicated, the use of automatic reporting in the issue tracking system removes the difficulty of actually computing the grades. The report is also documented to explain the computations performed, so that students can be assured that they are getting the grade they have earned.

Using this system, the instructor left all of the task management (and therefore project management) to the three managers and the team leaders. Only occasionally did the instructor intervene

to resolve issues related to the completion of tasks and their point values.

In terms of the unproductive types of behaviors, we believe that this approach is very effective at preventing hitchhiking. Freeloaders quickly gain a reputation and are not able to leech off of the hard work of other students for very long. Overachievers are free to work as hard as they want. In the extreme, an overachiever who does all the work for the project will leave only a few points of required work for each of the other students to do. However, this situation is unlikely to happen if the project is sufficiently large.

We addressed procrastinators using mini-deadlines for key tasks. This prevents procrastinators from adversely affecting the project by signing up for a key task and then not completing the work in a timely manner. Similarly, students are free to underachieve if they wish, but they will receive the grade that they earn. The unavoidable burden of the added work is still shouldered by the other students in the group. As our point graphs show, a significant proportion of students can procrastinate and still complete the project successfully.

The dilettante is prevented by limiting the number of tasks that a person can sign up for. Our system helps to prevent tutoring from occurring by splitting the earned points between the tutor and the weaker student. (An additional approach might be to force students to choose different partners for different tasks.)

6. Conclusion

In this paper we have presented a flexible, quantitative approach for evaluating individuals who work in groups. It is based fundamentally on a task management system that tracks the work related to the project. This approach is well-suited for software development projects, especially those using an agile development process.

Acknowledgments

The author would like to thank the Spring CSci 435 class for their participation. The author also thanks Dorothy Coppit for her insightful comments on a previous draft, including the identification of a rather embarrassing mathematical error.

References

- [1] James S. Collofello and Marla Hart. Monitoring team progress in a software engineering project class. In *Proceedings of the 29th Annual Frontiers in Education Conference.*, volume 1, pages 11B4/7–10, San Juan, Puerto Rico, 10–13 November 1999. IEEE.
- [2] David Coppit and Jennifer Haddox-Schatz. Large team projects in software engineering courses. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, St. Louis, Missouri, 23–27 February 2005. IEEE. Submitted for publication.
- [3] Jane Huffman Hayes, Timothy c. Lethbridge, and Daniel Port. Evaluating individual contribution toward group software engineering projects. In *Proceedings of the 25th International Conference on Software Engineering*, pages 622–7, Portland, Oregon USA, 3–10 May 2003. IEEE.
- [4] Clive C. H. Rosen. Individual assessment of group projects in software engineering: A facilitated peer assessment approach. In *Proceedings of the 9th Conference on Software Engineering Education*, pages 68–77, Daytona Beach, Florida, 21–24 April 1996. IEEE.
- [5] Thomas W. Schultz. Students assessing teams. In *Proceedings of the 29th Annual Frontiers in Education Conference.*, volume 3, page 13B2, San Juan, Puerto Rico, 10–13 November 1999. IEEE.
- [6] M. Shaw and J. Tomayko. Models for undergraduate project courses in software engineering. Technical Report CMU/SEI-91-TR-010, Software Engineering Institute, 1991.
- [7] TuxMonkey.com. The Issue-Tracker homepage. URL: <http://www.issue-tracker.com/>.