Lucas Layman
April 5, 2009

Reflection on Grading Rubric

A feature of software development in general is that a given problem may have many solutions. Part of assessing a software engineering task is whether the program generates the expected output for a given input (often called "verification"). While many different solutions may generate the expected output, the solutions may differ greatly in terms of quality, and achieving a solution of high quality is equally important to achieving a correct solution. Quality itself is less well-defined and is often in the eye of the various stakeholders (be they developers, managers, customers, etc.). Designing the most efficient solution (that runs the quickest and consumes the fewest resources) may be much different than a solution that is easily extensible further down the line in the project.

Grading programming assignments in software engineering can be particularly challenging if one wants to consider the quality aspect. Personally, I believe the quality of a solution (or an attempt) is more important than its correctness. When seeking employment, students will be interviewed not on "can you solve this problem?" but "how would you solve this problem and why would you do it that way?" Furthermore, some students by virtue of their personalities and sill sets, will be better at seeing the broader picture, thus leading them to design considerations that take into account more than just finding a solution that is correct. It would be remiss not to reward such students for these traits that will undoubtedly serve them well professionally.

The challenge of grading the quality of a programming project is that attaining a consistent, repeatable, and comparable scheme for assessing quality is nearly impossible. How do you judge the extensibility of a solution, or its reliability, and, most importantly, how do you compare the two solutions such that they can be graded fairly? The easiest way to assign points based on quality is to leave a few points on the assignment (or parts thereof) up to the grader. As seen in the grading rubric linked above, "use your judgment" appears throughout when solution quality (and not correctness) is in question. This freedom works well during the grading process itself, as oftentimes it is self-evident that one solution is clearly better than another. However, the weight and allocation of points may differ between graders, and some not all graders may judge different quality characteristics equally. This problem usually surfaces whenever students ask about why they lost points on "quality problems" and then (rightly) demand an explanation. The issue is compounded when students compare their solutions and ask why one's solution was better than the other when the root difference is a judgment call on the part of the grader. Not only does this lead to tension between student and instructor, it also leads students to believe that assessment of their course performance is entirely a matter of opinion on the part of the instructor instead of fair to all students. Also, students often prioritize their work in order of importance, e.g. "If I complete X, Y, and Z, I will get at least a 90." When judgment calls are a part of the grading scheme, students can no longer prioritize their work. While as instructors we of course desire that our students complete each assignment in its entirety, the reality is often different.

Ultimately, I think that a fair and balanced means of grading assignments that is not subject to the judgment of instructors is the best policy. If assignments are to be judged based on the quality of the solution, then the students deserve that those assignments be graded equally and fairly by a system that is not subject to the perceptions of the grader. All measurements in the rubric should be well-defined, not subject to interpretation, and repeatable. Some measures of the quality of a solution can be well-defined. For example, if efficiency is an important quality, then it is possible to construct tests that will determine how long it takes a program to run and how many resources it consumes. Measuring other attributes, such as maintainability of a program, may not be straightforward. One means of overcoming the problems in measuring difficult quality attributes is to avoid the challenge of direct measurement and instead make these qualities a part of the students' overall learning experiences or "life lessons." One mechanism for achieving such lessons in software engineering is a series of longitudinal assignments that build upon one another. For example, suppose a desirable solution quality is maintainability, therefore, the students' solutions should follow a strong object-oriented design to promote reuse, have clear code naming convention and comments, and should have thorough unit test suites to ensure that current program behavior is not unexpectedly altered when later changes are made to the program. If students fail to follow the principals of good design and maintainability, then later, the lack of these qualities will inhibit them in the completion of their program assignments and thus result in lower grades based strictly on the correctness (or other explicitly measureable) aspects of the program. However, the instructors are still responsible for educating the students of the importance of these traits, and revisiting them frequently if they will affect student performance in the course. Perhaps one means of giving this concrete feedback and emphasizing its importance would be to include a section on these unmeasurable qualities in the grading rubric, but not include them in the actually tally of the grades. This visual reinforcement may help students understand the importance of such attributes.