Automated grading system

Education, Grading System

If bridges and buildings were made like we make software, then we would have disasters happening daily. I have heard this several times from many people. It is sad but true. Buggy software is the bane of the software industry. One of the ways of increasing software quality is by propereducation. Several professionals from the software industry also attest to this.

They believe that a greater emphasis should be given to quality and testing in university courses. But simply explaining the principles of software quality is not sufficient. Students tend to forget theoretical principles over time. Practical exposure and experience is equally important. Students should be put in anenvironmentwhere they can appreciate the importance of quality software and can experience the benefits of processes that enhance quality. Many universities have a period ofinternshipfor the students in which they work in a software company and experience these factors first hand. However because the internship usually is of a duration of 3-6 months, it is not sufficient to instill the importance of quality.

Emphasis on code quality should be made a part of the entire software curriculum for it to have proper impact. Every assignment that the students submit should be subjected to the same quality standards that an industrial project would be subjected to. Having university assignments adhere to industrial standards will result in the faculty having to spend more time grading the assignments. The faculty can no longer just give an assignment, wait for the students to submit it, and grade them. The faculty must be more like a project manager who constantly mentors the students and helps them improve the quality of their work. Along with spending a good amount of time mentoring students off class hours another challenge is timely evaluation of student assignments. Faculty members are already overloaded with the task of teaching, designing projects, grading, and research.

Once we incorporate testing and quality into the curricula, each assignment will have to be graded along many more dimensions, such as quality of the tests, coverage of the tests, etc. This can be very time consuming. We need a mechanism which will automatically grade student assignments to the best possible extent, so that students are iven a timely feedback, and faculty can focus more on providing feedback on the style, design, and documentation of the project. Such a system will also bring consistency to the grading process and will eliminate discrepancies due to instructors bias and lethargy. A good automated grading system should be capable of executing the test cases written by students as well as the faculty on the project, determining the coverage of the test cases, and compiling and executing the submitted programs. It should be configurable so that faculty can determine the importance of various factors that make up the final grade. Several efforts have been made to design and implement automated grading systems in universities.

Some existing systems are: 1. WEB-CAT[1] 2. Curator[2] 3. ASSYST[3] 4. Praktomat[4] 5. PGSE[5] 6. PILOT[6] In this article I will briefly explain two such automated grading systems - WEB-CAT, and the Praktomat systems, and propose a system that contains useful features from them as well as some new features.

WEB-CAT WEB-CAT was created at Virginia Tech university to address the need for incorporating software testing as an integral part of all programming courses. The creators realized the need for a software to automatically grade student assignments to enable faster feedback to students and to balance the working load of faculty members. Since Test Driven Development (TDD) was to be used for all the assignments, the students had to be graded not only on the quality of code, but also on the quality of their test suite. WEB-CAT grades students on three criteria. It gives each assignment a test validity score, a test correctness score, and a code correctness score. Test validity measures the accuracy of the students tests. It determines if the tests are consistent with the problem tatement.

Test coverage determines how much of the source code the tests cover. It determines if all paths and conditionals are adequately covered. Code correctness measures correctness of the actual code. All three criteria are given a certain weight-age and a final score is determined. WEB-CAT’s graphical user interface is inspired by the unit testing tool JUnit[7]. Just like JUnit it uses a green bar to show the test results. A text description containing details such as the number of tests that were run, and the number that passed is also provided.

Basic features provided by WEB-CAT are: Submission of student assignments using a web based wizard interface • Submission of test cases using a web based wizard interface • Setup of assignments by faculty • Download of student scores by the faculty • Automatic grading with immediate feedback for student assignment WEB-CAT follows a certain sequence of steps to assess a project submission. A submission is assessed only if it compiles successfully. If compilation fails, then a summary of errors is displayed to the user. If the program is compiled successfully then WEB-CAT will assess the project on various parameters. It first tests the correctness of the program by running the student’s tests against the program. Since these tests are submitted by the students, and it is expected that 100% of the tests will pass, because we do not expect students to submit a program that fails their own tests. After this the student’s test cases are validated by running them against a reference implementation of the project created by the instructor.

If a student’s test case fails on the reference implementation then it is deemed to be invalid. Finally the coverage of the student’s test cases is evaluated. Once the scores are obtained a cumulative score out of 100 is calculated applying a certain formula on the scores from all criteria. The results are displayed immediately to the student on an HTML interface. It was observed that the quality of student assignments increased significantly after using WEB-CAT. It was found that the code developed using WEB-CAT contained 45% fewer defects per 1000 (non commented) lines of code[8]. Praktomat Praktomat was created at Universitat Passau in Germany.

The purpose of creating Praktomat was to build an environment which would help students enhance the quality of their code. Along with automated grading it also has a focus on peer reviews. The creators of Praktomat felt that reviewing others software and having one’s software reviewed helps in producing better code. This is the reason why Praktomat has a strong focus on peer review and allows users to review as well as annotate code written by other students. Students can resubmit their code any number of times till the deadline. This way they can improve their code by adopting things they learned by reviewing other students code as well as lessons they learned by others feedback of their own code. Praktomat evaluates student assignments by running them against a test suite provided by the faculty.

The faculty creates two test suites – a public suite and a secret suite. The public suite is distributed to the students to help them validate their project. The secret test suite is not made available to the students, but they are aware of its existence. An assignment is evaluated by automatically running both the test suites against it, and also by manual examination by the faculty. Praktomat was developed in Python, and is hosted on SourceForge[9]. ObservationsMy contention that student project submissions should be backed by a process to encourage best practices, and a software to automate as well as facilitate the process, has become stronger after reviewing WEB-CAT and Praktomat. What best practices should we incorporate in the process? What are the features that an automated grading software should contain? WEB-CAT, Praktomat, and several other software give a good starting point.

We can learn from their successes and failures, and enhance the offering by adding our own experience. WEB-CAT and several other sources[10] have shown us that TDD is definitely a good practice. In a university environment TDD will work best if it is complemented by instant feedback to the students. We want to have a process that will encourage students to improve the quality of their code. They should be graded on the best code they can submit till the deadline. Two things are needed for this – instant feedback and the ability to resubmit assignments. WEB-CAT achieves this by assessing submissions in real time, and displaying the results to the students immediately.

WEB-CAT allows students to re-submit assignments any number of time till the due date. Since faculty members are already overloaded with work, the software should take some of the faculties responsibilities. WEB-CAT automatically evaluates and grades the student’s assignments, leaving faculty with time for more meaningful activities. Praktomat has shown us that there is a definite benefit to peer review. When we review code written by others, we can go beyond the paradigms set in our own mind. Having our code reviewed by others can help us see our shortcomings which we may have earlier overlooked. Praktomat allows students to review code written by others.

However the review is hidden from the faculty, to ensure that it does not impact grading. Praktomat does not rely on 100% automatic evaluation of the assignments. Praktomat evaluates certain aspects automatically and the rest are evaluated manually. Factors like code quality, documentation, etc are reviewed and evaluated manually by the faculty. There may be two reasons for this. Software to support automatic evaluation of these things may not have been available when Praktomat was written, or the creators felt that certain things are best evaluated by the faculty. A proposed system for automated grading Based on my observations from reviewing the above software systems and from my own experience, I have defined a process and the functional expectations of a software system that supports TDD and automated grading.

The Process • Every project should have a deadline, just like the real world • The project should be defined as a set of use cases and a functional test suite. Both should be made available to the students. • Students should start developing their project using the TDDphilosophy. • They should also be provided a source code repository like CVS or VSS. Once the students have completed their project they should tag the build and should upload the tag number to a web based submission software. • It must be clearly defined how the students should submit their unit test suite. • They should also provide one file which will trigger the remaining unit tests.

• The software will pull the source from the repository, and evaluate it. oFailureis reported to the student if the project fails to compile. Failure here does not mean that the student fails in the assignment. Assignments can be corrected and submitted any number of time till the deadline. Once the compilation succeeds, the software will run the unit tests written by the student on their code. o After collecting results from the unit tests, the test coverage is measured. o Then the functional tests created by the faculty are executed against the software.

o The software is then run through a source code format checker which evaluates it for adherence to coding standards, The software is then run through a source code quality checker which evaluates the quality of code based on known best practices, and anti patterns. o The software is finally channeled to the faculty who evaluates it for design. Results from all the tests are given out of 100%. o After collecting all the results a formula (provided by the faculty) is applied to derive the final score. The Software • The software should provide an account with a username and password to each student and faculty. • The software should be web based so that it can be accessed from anywhere using a standard web browser. • After logging in students should be able to browse to the homepage for a particular assignment and view the details, such as specification, due dates, and any other details posted by the faculty.

When a student completes her assignment, she should be able to upload the CVS tag number to the server.• Once the tag number is uploaded the server should pull the source code from a CVS repository and perform the checks mentioned above. • Results from each check is recorded in the database. • The detailed result is then displayed to the student. • Students should be able to resubmit an assignment any number of times till the deadline. • Student code should be available for peer review and annotations if the faculty desires. The faculty should be able to create an assignment and upload details and files.

• The faculty should be able to trigger the final evaluation of all assignments either manually, or at a scheduled time. • An evaluation should take the latest tag numbers provided by the student and perform tests on the respective source code. • Results should be made available to the faculty, and students. • The faculty should be able to add their own scores for parts that were checked manually. • The final result is computed by applying a formula provided by the faculty. The final results should be downloadable as a csv text file. Several technologies such as Java, Python, PHP, .

NET, and Ruby can be used to implement such a system. Each have their pros and cons. We will not cover the implementationtechnologyin this paper. Evaluation of these technologies and a final choice based on the evaluation will be dealt with in a separate paper. Reference: 1. http://scholar. lib.

vt. edu/theses/available/etd-05222003-225759/unrestricted/Web-CAT. pdf 2. http://www. cs. vt. edu/curator/PublicInfo/CuratorIntroduction.

pdf 3. http://portal. cm. org/citation. cfm? id= 268210 4.]http://www. infosun.

fmi. uni-passau. de/st/papers/iticse2000/iticse2000. pdf 5. Jones, E. L. Grading student programs – a software testing approach.

J. Computing in Small Colleges, 16(2): pp. 185-192. 6. http://www-2. cs. cmu.

edu/~rsbaker/pilot. pdf 7. http://www. junit. org 8. Using Test Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance. http://web-cat.

cs. vt. edu/grader/Edwards-EISTA03. pdf 9. http://sourceforge. net/projects/praktomat/ 10. http://www.

testdriven. com