ABSTRACT

KATAKAM, UMA MAHESH. Testing of Open-Source Software Developed in Class. (Under the direction of Dr. Edward Gehringer).

Open-source software (OSS) has seen considerable growth in the software industry. Many companies have started using OSS for their mission-critical applications. The transparency of OSS technology and the easy availability of the source code makes it ideal for students to participate in OSS project development. Also working on OSS projects helps improve students' software development skills [j].

Students often come up with brilliant ideas and develop wonderful software applications. These artifacts have the potential to become good OSS. We would like this work to benefit the OSS community. But these artifacts are developed over a short period of time, with limited resources and knowledge, and usually with minimal or no testing. Thus, the quality is not assured.

We provide, as a solution, a generic process that students can use to improve the quality of the code that they write. The framework encourages students to write comprehensive test cases.

Testing of Open-Source Software Developed in Class

by
Uma Mahesh Katakam

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Computer Science

Raleigh, North Carolina

2010

APPROVED BY:

_____	_____
Dr. Edward Gehringer Committee Chair	Dr. Laurie Williams
_____	_____
Dr. Thomas Honeycutt	Dr. Stephen Edwards

DEDICATION

To my parents, grandparents, Divya and Niki…

BIOGRAPHY

Uma Mahesh Katakam was born on 15th May 1984 in Khammam, Andhra Pradesh, India. He finished his schooling in 1999. He received his Bachelor of Engineering (Computer Science) degree from Osmania University, Hyderabad, Andhra Pradesh, India in 2006. He worked as a Software Engineer with Satyam Computer Services Ltd. from 2006 – 2007 and later on as an Associate Consultant with the same company from 2007 – 2008. He did a Co-op at GlaxoSmithKline from May – 2009 to November – 2009. He joined the graduate program at the Computer Science Department in North Carolina State University in Fall – 2008. He has been working with Dr. Edward F. Gehringer on his master's thesis since January 2009. With the defense of this thesis he is receiving the degree, Master of Science in Computer Science from North Carolina State University.

ACKNOWLEDGMENTS

I would like to thank the following people without them it would have been impossible to be in the position that I am today.

Dr. Ed Gehringer, my mentor and advisor, for his support and guidance. I am very thankful to him for spending a lot of his valuable time to review my thesis and helping me improve the work that I have done.

Dr. Thomas Honeycutt and Dr. Stephen Edwards for being on my committee and making time for me out of their busy schedule.

My parents and grandparents, who have always been my source of inspiration and encouragement.

Divya Teja Vavilala for always being with me.

My Brother Nikhilesh Katakam and my friends (Kishore and 2346 group) for helping me out whenever I needed their advice.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

Open-source software (OSS) is becoming an increasingly popular way of developing software in industry. More and more companies are moving towards using OSS. There are a few companies that are using OSS for their mission critical applications as well [j]. According to a study by the Gartner group, 85% of the companies surveyed use OSS in their enterprise applications and the remaining 15% expect to do so in the near future [o, p]. The rate at which OSS is being adapted is an example of its growing popularity [k].

Unlike traditional software development, OSS is developed using open standards where users and a group of software developers collaborate to add features and fix bugs [l, m]. Also, the source code is available to the world, so that people can use the code, modify it according to their needs, or enhance it [l]. The easy availability of the source code facilitates student participation in OSS development. Students and the OSS can improve symbiotically. Students learn software development and gain real-life experience on software development and at the same time contribute to the open-source community [j]. Therefore, academic projects on OSS are some of the best vehicles for students to improve their software-development skills.

The advantages of student participation in OSS development are—

- Students gain real-life software development experience [j, n]

- Having OSS projects on their résumés [j] improves students' career prospects.
- Students can make significant contributions to the open-source community

However, there are a few concerns in involving students in OSS development. Quality is a major concern with student-developed code [e, r]. In our experience with students, we had students come up with much-needed extensions to our open-source application *Expertiza*, as well as the Java OSS applications JFreeChart and Buddi. But when it came to testing the application, only a few of them submitted comprehensive tests along with their code. Students claim that they have done sufficient testing even when they have not. This leaves us in a situation where we have the code but yet it is not useful because it is not well tested.

There are various reasons for poor quality of student-developed code.

- Lack of time and resources [q]
- Lack of knowledge of which development process to follow in order to produce a quality output [q]
- Little knowledge of, or enthusiasm for, testing [e]

This thesis attempts to provide a framework for software development in classrooms. This framework takes into consideration all the above mentioned concerns and constraints. We provide a generic framework for allowing students to improve the quality of the code they develop in classrooms, thereby allowing students to contribute more efficiently to OSS. The

thesis will contain two possible implementations of the framework, and discuss their pros and cons. Each approach is appropriate, but in different situations.

The thesis is structured in the following way. Chapter 2 summarizes the results of the survey done to identify the problems. Chapter 3 discusses the relevant work and its shortcomings. Chapter 4 discusses the generic framework and its two implementations. It also describes the pros and cons of each implementation and gives recommendations. Finally, Chapter 5 concludes the thesis.

Table 1: Abbreviations used in this thesis

Abbreviation	Description
OSS	Open-source software
BID	Build – Improve – Deploy
LOC	Lines of code
TDD	Test-driven Development

Chapter 2: Background

The goal of the thesis is to identify problems that are associated with poor quality of the code developed by students and provide recommendations to fix them. In order to find out the problems, we conducted a survey in NCSU's CSC 517 class in Fall '09. The survey was intended to see how well students test their projects and identify the problems that stop them from testing. The questionnaire included these questions.

Table 2: Survey questionnaire

SNo	Questions
1	What is the duration of the project in weeks? (Project assignment date to project final submission date)
2	Average number of hrs. /day you worked on the project.
3	Out of the total time spent on the project, what percentage of the time was spent on writing test cases and testing the code?
4	What was the size of your project team?
5	What is the approximate length of the code you wrote (in lines of code)?
6	Did you develop test cases before you started coding?
7	If you answered the previous question "yes" or "sometimes," how many of your test cases did you write before you started coding, and how many after coding was complete?
8	What is the percentage of code covered by your tests?
9	Please list any difficulties that you faced in testing the application (e.g., shortage of time, did not know what to test).
10	What sort of testing did you do? (unit testing, functional testing, integration testing, etc.)
11	Please list all the testing tools that you used for this project.
12	If yours was a team project, did you assign the testing responsibilities to a specific team member? If yes, was this helpful?
13	Please list the number of defects or issues you found, with the application, during the entire course of the project.
14	Do you believe that the test cases helped you improve your code?
15	An e-mail address we can use to communicate with you

Thirty one students responded to the survey. A summary of the results can be found in the table below. The entire survey results can be found in Appendix.

Table 3: Summary of student survey

What is the duration of the project in weeks? (Project assignment date to project final submission date)	Average number of hrs. /day you worked on the project.	What was the size of your project team?	Out of the total time spent on the project, what percentage of the time was spent on writing test cases and testing the code?	Did you develop test cases before you started coding?	What is the percent age of code covered by your tests?
3.03 weeks	3.32 hours	2 - 3	29.93 %	Yes- 4, No - 23, Sometimes- 4	68.18%

From the survey we can conclude that typically students are allotted 3 weeks for completion of the project and students work 3 – 4 hours per day on the project. The important derivation of this survey is that 23 out of the 31 students did not write a single test case before they started coding. Nine out the 31 students used testing tools like JUnit. Students also said that on an average they spent around 30% of their project time on testing and on an average they achieved code coverage of 68%. This amount of time spent on testing and the code coverage achieved is decent enough. To verify students' claims and to see how well the test cases were written we then reviewed all the project submissions. The following are the observations –

Table 4: Submissions review summary

SNO	Project ID.	Unit testing	Functional testing	%
1	E101	X	X	
2	E102	X	X	
3	E1101	Yes	Yes	Partial
4	E1201	Yes	X	Complete
5	E1301	Yes	Yes	Partial
6	E201	X	X	
7	E301	Yes	Yes	Complete
8	E401	Yes	Yes	Partial
9	E402	X	X	
10	E501	X	X	
11	E502	X	X	
12	E601	Yes	X	Partial
13	E701	Yes	Yes	Partial
14	E801	X	X	
15	Leaderboards	X	X	
16	P101	X	X	
17	RRD	X	X	
18	RRD2	X	X	
Count		**Yes -7, No -11**	**Yes -5, No – 13**	

The following deductions can be made from the above submission review –

- Only 38% of the teams did some sort of unit testing.
- Only 28% of these teams did some functional testing.
- Out of the 7 teams that submitted test cases only 2 of these did extensive testing.

There is a high level of discrepancy between what students claimed and what they actually submitted. Students overestimated the amount of testing they did by a factor of about two.

They thought they did enough testing, but they did not. Students were instructed to submit test cases and were also made to realize the importance of testing. Even after this only 2 out the 18 teams did comprehensive testing. So we need a process that shall ensure that students write test cases.

When asked about the reasons for not able to sufficiently test the software, students said the following –

- 10 out of the 31 teams said they were short of time.
- 5 out the 31 teams said they were not sure what to test.
- 4 teams said they did not know how to test.
- 3 teams were not satisfied with the documentation provided about the project and/or the technology used.
- 2 teams said that the code was too complex to write test cases

The major conclusions from the survey are –

- Students do not have enough time to test
- Students have little testing knowledge
- Students do not have a process that they can follow
- Students make less use of tools that can help them work faster

We therefore need a process that the students can follow during development that will help them work in an organized way and use best practices and tools to come up with well tested code during the stipulated amount of time.

Chapter 3: Relevant Work

Traditionally, in classrooms, the general procedure is that the instructor hands out the assignment(s) to the class, and the students are required to submit the code with some test cases before a deadline. For the final project, the time allocated is 3 – 4 weeks [Appendix], which we believe to be typical of courses. The assignment is done in teams or individually [s]. The typical team size is 3 - 4 [Appendix]. Given that students take multiple courses during a semester, the time per day that they spend on a project is 3 – 4 hours [Appendix]. Therefore, the total effort per day in a team assignment is 9 – 16 person-hours. Each team follows their own methodology to develop software. Generally teams write the code and fix the bugs as they encounter them, using the trial-and-error method [t]. Little if any testing is done. As a result the quality of the code is very low [e, r].

By contrast, in the software industry, based on the software development methodology used, there are separate teams and time set out for testing. There are various levels of testing done. Testing the software includes a unit-testing phase, a functional/ system testing phase, and a user acceptance testing phase (UAT). Unit testing is usually done by the development team. To perform system testing there is a separate testing team involved. The responsibility of this team includes writing functional test cases, getting these test cases verified and approved, execute these test cases and report bugs as they find. UAT is generally done by the users of the software. These different levels of testing done and the

amount of resources and time spent on testing ensure that the final software is of very high quality. Our experience is not unique. To improve code quality, a software engineering course in the Department of Software Engineering at the University of Belgrade recommended that the students use traditional software development methodologies [s]. Students were grouped into small teams, and each team was assigned a unique project. Each team was asked to use an appropriate software development methodology and tools as required. After the projects were submitted, a review of the submissions revealed that the quality of the software developed was still very poor. In fact the quality was so poor that only 5 out of 41 projects met the requirements and did not show any unexpected behavior. The major constraint was lack of testing knowledge. Even though students followed a methodology for development, the number of test cases written and the amount of testing done on the software was purely ad hoc, and was entirely dependent on student's professional habits and experience [s].

On the final project in CSC 517, fall 2009, we asked students to sign up for particular open-source software modules. Most of the students signed up for adding additional functionality to *Expertiza*, an open-source application used to generate reusable learning objects using peer evaluation. Students were allowed to work in teams of 3 – 4. The time for completion of the project was 4 weeks. Students initially were asked to submit the project design. The design documents were peer-reviewed and the teams incorporated the changes suggested.

Then the students submitted their working code along with test cases by the deadline. The students were then asked to demonstrate their work to the instructor and to the TAs. While it was not apparent during the demonstrations, later code reviews revealed that most of the teams wrote limited test cases for the code. Limited amount of testing leads to low code quality.

The reasons cited by the students were:

- Shortage of time
- Limited testing knowledge
- Lack of awareness of a process to follow
- Lack of documentation

There are studies that support usage of TDD and automatic grading tools to quicken the process so that students get immediate feedback on their work. TDD helps students to test their code better their by improving the quality. Using tools for automatic grading helps students get feedback quickly and this gives students an opportunity to correct the mistakes and resubmit the code before deadline [e, u]. However, this approach requires someone else (usually the course staff) to write the test cases. It works where all students are writing the same module(s); it is inapplicable where students select from a list of projects to do, with no more than a handful of teams doing the same project. Further, it is hard to write test cases generally enough so that all reasonable implementations will pass, unless the students are spoon-fed a design, with little if any room to exercise their creativity.

Another way to encourage writing tests would be to use tools that automatically measure code coverage of a set of tests. This doesn't require someone else writing the tests; the programmer(s) can write them themselves. This way we eliminate the need for outside parties to write the tests. But this approach also does not meet our goal, because it assumes that the students write the tests before we can use the tool to verify the code coverage. But the problem is that students tend not to write tests. We have seen that time is always a constraint and this approach does not overcome it. Therefore, we need an effective motivation for students to write comprehensive test cases, thereby leading to an improvement of the code.

To sum up the major reasons for poor quality of code have been identified as –

- Lack of time and resources [q, s]
- Lack of knowledge of which development process to follow in order to produce a quality output [q, s]
- Little knowledge of, or enthusiasm for, testing [e, s]

The approaches discussed above (e.g., the usage of traditional software engineering methodologies, design review, automatic grading systems and code coverage tools) address a few of these mentioned concerns but create new ones. In the following section, I discuss a generic framework that can be used to tackle these constraints. The generic framework makes use of the best practices discussed in above methodologies and other approaches.

Chapter 4: Solution

4.1 BID framework

Figure 1: BID framework

We will propose a solution that encompasses three steps, at a higher level of abstraction. This is called the **Build-Improve-Deploy (BID)** framework. Build, Improve and Deploy are namely the three phases of this framework. Each of these phases in the framework is relatively independent of the others. This means that the process followed in each phase is not dependent on the process followed to complete the other phases. Only the initiation of a phase is dependent on the completion of the previous one. Using this framework, we believe that we can improve the quality of the code that is being produced in the classroom and thereby, these outputs can be used in OSS.

The following is a brief description of each of the three phases:

Build: Step one is the initiation part of the lifecycle. This is called the *building step*. Students here follow various approaches to produce a working program. The inputs to this phase are a problem definition, evaluation rubric, and the guidelines for the output. The output of this

phase is a working piece of code with a relevant, sufficient, and comprehensive set of test cases. Students in this phase start the project by understanding the requirements of the project and then start writing the actual code. There are different processes that can be followed to complete this phase. Each of this processes have been discussed in detail in the following pages.

Improve: This phase of the BID model is an important one for that fact that most of the error correction and fine-tuning of the program is done this phase. The output of the Build phase is the input for this phase. The submitted code is reviewed and feedback is provided on how to improve the code and fix or correct the bugs that are present. This particular phase can be repeated multiple times so as to improve the quality of the code. The output of this phase is well refined and production ready code that is ready to be deployed/installed/integrated into a system.

Deploy: Once we have the refined code ready, the next step is to integrate/install the code. This is what is exactly done in the Deploy phase. The code, which is the output of Improve phase, acts as the input for this phase. Based on the type of the application developed the code is either integrated with an existing application or hosted if it is an entirely new application or made available to others for download and installation. For example, if it is a standalone web application, then the team/participant that owns the code has to take steps

to host the code on a server so that it can be open to users. Completion of this step marks the end of the phase.

4.2 Approach 1

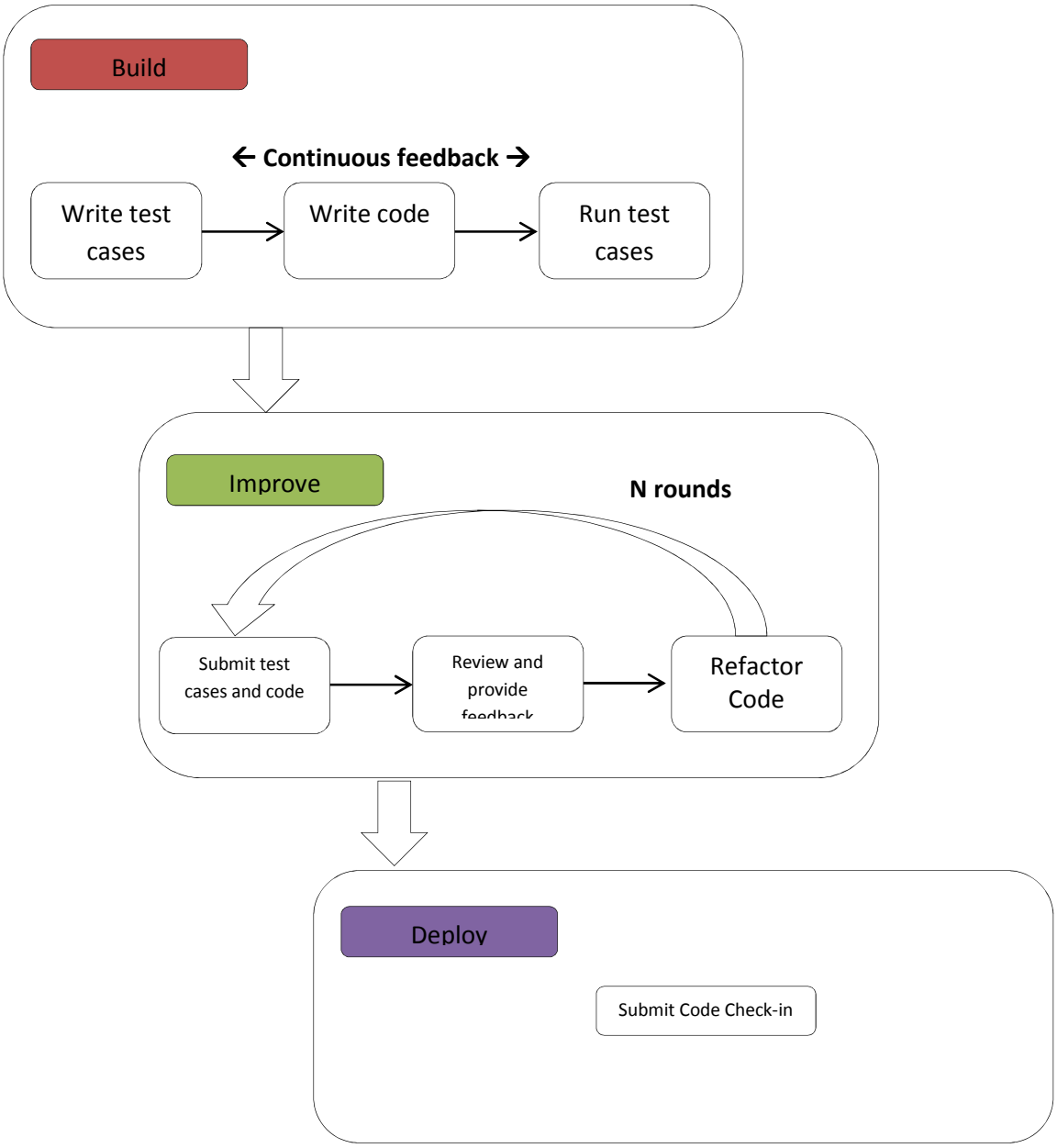

Figure 2: Approach 1 process flow

Figure 2 shows an approach that can be used by students to improve the code, so that it can be used in an OSS application. This is derived from the generic three-phase framework. The process that is followed to produce output in each phase is explained in detail below.

Build

The Build phase of the framework can be realized through Test-Driven Development. The student is provided with the project requirements and the artifacts that he/she needs to submit. Unlike the normal approach, where a student begins writing the code, we suggest that the student start by writing the test cases [e]. TDD is generally followed by developers who are very experienced and have a high command over the programming language that they are using. It would be unfair to expect the same from the students. Therefore, one variation is to have the student write the code for unit test cases in plain English. The following are the advantages of test-first strategy [e]:

- "It is useful in small projects with minimal training.
- It gives the programmer a great degree of confidence in the correctness of their code;
- It encourages students to always have a running version of what they have completed so far;
- It encourages students to test features and code as they are implemented. Doing so preempts the integration problems that students face while working on large projects."
- It helps improve students' understanding of the requirements. [d, b]

So in this phase the student first writes the test cases, then codes the requirements and the test cases simultaneously, modifying either the code or the test cases as required. Once the code and the relevant test cases are complete, the student runs these test cases against the code and checks the correctness and quality of the code. Importantly, this encourages providing continuous feedback to the code in development. Frequent, useful and continuous feedback on students' works helps them improve the quality of their code [e]. One way to encourage this is by using message boards. Students post their relevant questions on the message board and there is a constructive feedback provided by the other students of the class. This feedback helps the student to come up with efficient and quality code in the very first phase, saving time later. But no feedback may be provided, or the feedback may be incorrect. To mitigate this risk, the instructor can promise extra credit for students who provide constructive feedback. This ensures that students help each other by providing positive and useful feedback [i]. Once the code is sufficiently tested and/or the deadline is met, the student submits his/her code along with the test cases. This would be the output of the build phase.

Improve

Once the code is submitted, the Improve phase is for seeking feedback on the quality of the code and the test cases. The review mechanism suggested for this phase is peer review. Peer reviews are formal, effective and efficient way of detecting bugs in the design and

code and thereby improve the quality of the code [b, c, g, h]. Students in the classroom review others work and provide their constructive feedback, this helps students improve their code. To make the reviews fair and useful, the instructor can provide some incentives for a quality review [i], for e.g. extra credit.

Peer review can be done in two different ways:

1) **Code and test-case review by individuals:** The code and test cases are assigned to students for review and students execute the test cases, inspect the code and provide feedback and a score on the submission. The scoring methodology will be discussed in the following sections. Briefly, each student is provided with a rubric and a questionnaire to be used in reviewing the submission. More people reviewing the code and multiple rounds of review shall improve the code [g, h]. When there are multiple reviews more errors can be found that where ignored or omitted during other reviews [g, h].
2) **Team/group review [b]:** In this method, we ask teams to review and test the code written by other teams. For example, Team A does assignment A, Team B does assignment B and Team C does assignment C. Then the instructor can decide based on any criteria that Team A reviews Team C's submission, Team B reviews Team A's submission and Team C reviews Team B's submission. The reviewers can be changed

for every round of review as well; doing this helps find more bugs and better feedback [g, h].

3) **Paired peer review [b]:** This is similar to the earlier approach but, here the instructor swaps team members among the teams and asks them to perform review. This way the new team member can get information about the project from the original team members, thus eliminating the constraint of lack of knowledge about the project.

Other than improving the quality of the programs, peer review has the following advantages [d, b]:

- "Students get additional practice in reading Z specifications
- while inspecting students realize the value of precise, unambiguous and verifiable requirements;
- Students get to see and study an alternative solution to the problem they worked on;
- Students receive peer evaluation of their work and see the rather dramatic results that such assessment can produce;
- They get practice in technical communication by articulating inspection results"

In a study conducted by Nicole Clark from the University of Tasmania, and as reported in her paper [b], these are the positive outcomes of following peer testing in classrooms [b]:

- There was an increase in the quality of the final output.
- There was increased collaboration between teams.
- Students learned new things in a faster and better way.
- Students realized the importance and usefulness of testing.

Scoring Methodology

In industry, test-case completeness would be an ideal metric to measure code quality, because a programmer should check in code only if all the test cases pass. When dealing with students, we cannot guarantee the same; therefore we need other criteria. We propose to use the Web-CAT scoring model, which takes into account test-case validity and code correctness [e]. The score for the submitted code is dependent on the factors below [e] -

1) "Test case validity (TV): Are the test cases related to the code and the functionality?
2) Test case completeness (TC): Are there enough tests to cover the entire code?
3) Code correctness (CC): What is the number or percentage of test cases that pass?"

This means that the submission is graded on how many test cases have been submitted and how many of the test cases submitted are valid and how many of these the code passed. This means that the score for the code is a product of the above three factors.

$$(1)\ \text{Score} = (TV \times TC \times CC)$$

This formula prevents students from skimping on test development; if any one of the three factors is ignored, one of the factors is 0, and hence, so is the score [e]. The second and third factors can be calculated using test-coverage tools. One could use Clover if the code is written in Java. Clover also provides a set of other metrics beyond basic code coverage

information [v]. For Ruby we can use tools like *rcov* which is easy to install and use [w]. Using test coverage tools here in this process ensures that the review can be done better and faster.

One change to the Web-CAT scoring model is that we bring in another factor in the final score computation. In Web-CAT, the instructor has a reference implementation against which all the student submissions are compared and graded, whereas, in our situation, where students do multiple projects it becomes impossible to have the instructor write reference implementations for all the submissions. Therefore, to check whether the student has written the correct code, according to the requirements, we recommend that the final score be computed on the basis of how well the students test the code and how well the students meet the requirements as well (RM). Therefore the final score now becomes:

(2) "Score = (TV $\times$ TC $\times$ CC) + RM"

This approach 1 works perfectly in an environment where there are limited resources and there is a time constraint. It is easy to follow and implement both for the students and the instructor or the TA.

The advantages of approach 1 are-

- Firstly, it encourages test-first strategy and suggests that students be graded on how well their code is tested. Therefore, it is imperative that students do write sufficient test cases. Doing this helps improve the code quality.
- Because the test-first strategy is used, students do not need to spend extra time to write test cases. Students write tests as a part of their coding activity. Therefore, this process helps save students time [x].
- The code along with the test cases is peer-reviewed for completeness, validity and correctness. The feedback gives students a motive to improve the code and test cases submitted.
- Since BID is a structured software-development process, students are developing code in an organized fashion. This diminishes students' confusion over what to do next and when to do next. Finally, it saves time for the instructor and the TAs, which they can use to monitor the entire process and pitch in whenever there are any conflicts.

4.3 Approach 2

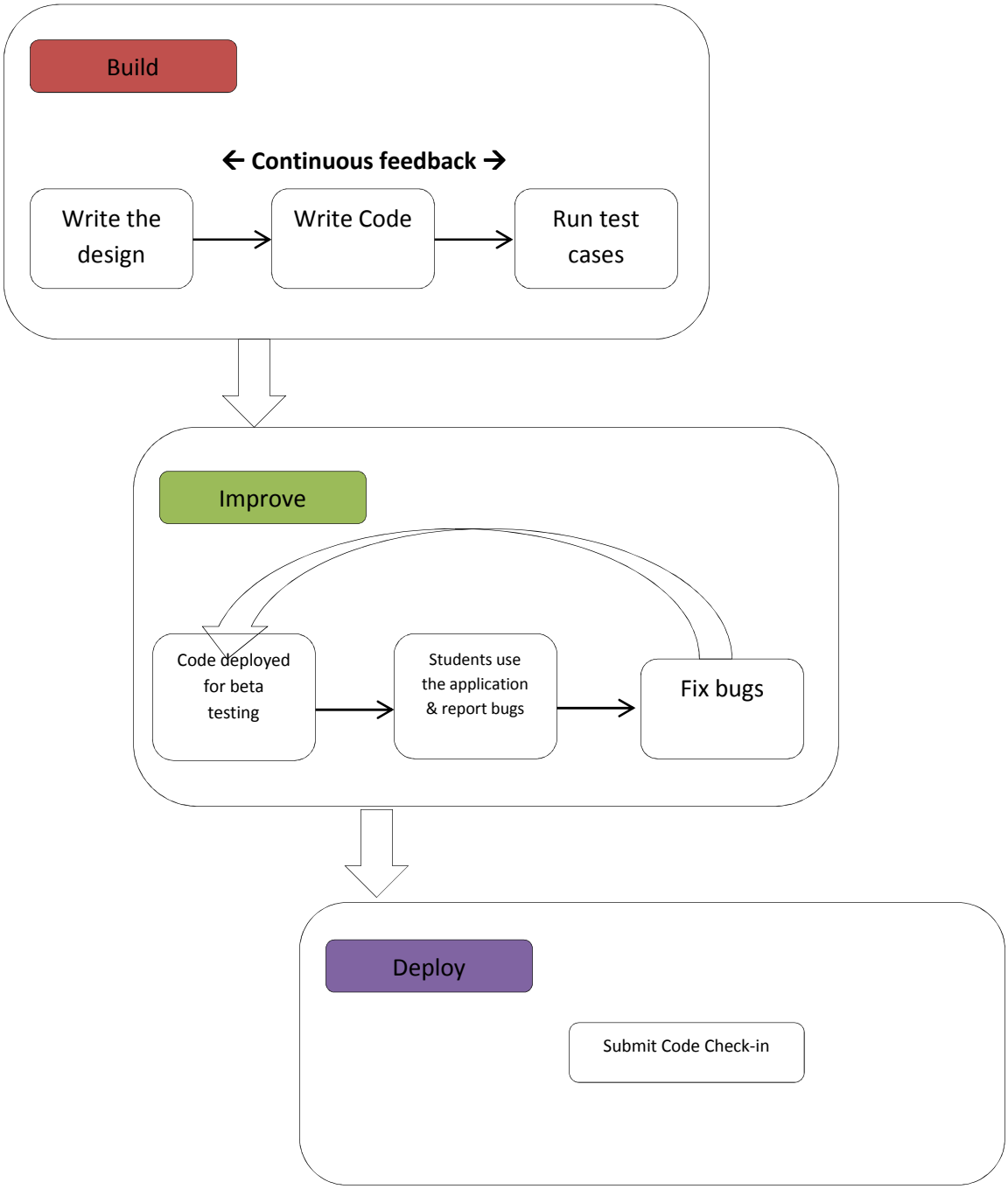

Figure 3: Approach 2 process flow [a]

The other approach, which is based on the BID framework, differs considerably from approach 1. This approach is similar to how F/OSS test framework [a, f] works. The process flow in individual phases is detailed below.

Build

In this phase, unlike the earlier approach, students start by designing the system. Much like the traditional approach, students then code their application and test it after the coding is done. Any issues found during test execution are fixed by the students in this phase. This is more similar to how things are usually done in the classroom. As in approach 1, continuous feedback is provided to the students. Continuous feedback helps them identify problems as early as in the design phase, thereby improving the quality of the code. As discussed earlier in approach 1 message boards can be used to provide feedback; all the advantages and disadvantages discussed earlier apply here as well. The outputs of this phase are the code written by students and the test cases to be used.

Improve

The quality of output of the build phase entirely depends on the skill of the students who wrote the code. This means that this code might have minimal bugs or might contain a large number of bugs. This output isn't of production quality. Therefore, in this this lifecycle, the Improve phase plays a very important role. This is where the majority of bugs are identified

and fixed, thereby maturing the code that was submitted in the build phase. This is much like the F/OSS test framework [a]. The process followed here is, once the students submit their code, they provide a copy of the code to be installed or host the application on a server where students can access it. The instructor then decides the set of students who will act as dummy users and test the code. Accordingly these students would install the software or work on the hosted application that they are supposed to review and test. The students act as real users and perform user tests. Student-testers log all the bugs that they encounter during this phase, in a repository provided for that particular project. The developers of this project then address the issues/bugs that have been reported. This way the students mature their respective projects in the Improve phase. So even if the quality of code is poor in the Build phase, it is continuously refined in the Improve phase. The output of the Improve phase should be a production-ready piece of code.

Scoring Methodology

The intent behind this approach is to make student developers write code that adheres to the requirements, and fix the bugs that have been found in their code by the student testers. Using approach – 2, we also want to make sure that student testers identify bugs by testing the submissions assigned to them. To ensure all the above factors are addressed, we need a scoring model that takes into account all above said criteria. The factors that are used in computing the score are:

1) Requirements Met (RM): This is based on how well the code meets the requirements.
2) Bug Fixes (BF): Every bug that is identified by the student tester is assigned a severity score. So this score is the sum of all the bug severity scores that the student developer has fixed. This way we ensure that the student developer fixes the bugs that have been found in his/her submission.
3) Number of Bugs Found (NBF): Every student developer is also a student tester for another project. In order to ensure that he/she does his testing duties we have made NBF as a part of the final grade the student gets. NBF is a relative score and is computed as the relative number of bugs found by that student tester.

Therefore, the following are the computations made:

$$BF = \sum_{k=1}^{n}(SB_k)$$

Where SB_k is the severity of the k^{th} bug fixed and n is the number of bugs fixed.

$$NBF = \frac{bf}{mbf}$$

Where *bf* is the number of bugs found by a student tester for a submission and *mbf* is the maximum number of bugs found for that submission by any tester.

Score = (RM + BF + NBF)

Approach 2 can be very helpful in situations where students have few projects during the class, and the instructor wants to teach the course with the help of one big project that the students can build and modify through the end of the course. The project should be assigned early in the course. The instructor can have various deadlines and milestones for the project, as stated in approach 2. Different assignments in the course are effectively different phases of the *same* project, which avoids the need to learn existing code time and time again. This way the instructor gets to teach his course with the help of a single project and the students also have enough time to spend on the project.

The advantages of approach 2 are –

- Students have enough time to dedicate to the project. This helps students improve the quality of the code they have written.
- The code is deployed for beta testing and a set of students acting as virtual users test the code over a period of time. This ensures that the code is well tested and the bugs are identified and fixed before it is deployed.
- Since this approach also encourages a more structured software development, as does approach 1, students develop their code in a more organized fashion.
- Finally, it saves times for the instructor and the TAs, which they can use monitoring the entire process and pitch in whenever there are any conflicts.

4.4 Future Implementation

We are planning to implement the BID approach in the fall ’10 class of CSC 517. The approach that would be followed during this class would be a combination of both approach 1 and approach 2. The ground rules for the final project in this class would be:

- Students would be working as individuals or in teams of 3 - 4 students.
- Students are allowed to work on one project per team through the semester.
- Different assignments in the class would be essentially different phases of the project. Newly covered material, e.g., design patterns, can be covered by asking students to modify their code to incorporate it.

Students would start working very early on the project therefore, giving them at least 10 weeks of time to finish the project. The following sections describe the process to be followed in each phase.

Build

During the build phase like in approach 1 students would follow test-first strategy to develop code. Students have a time of 3 - 4 weeks to complete their code along with the test cases. During this time, like in approach 1, students are given the requirements and by the end of this phase they are required to submit code and relevant test cases.

Improve

Once the code is submitted the review phase begins. Each project is reviewed by 4 – 5 students other than those who worked on the project. These students act as virtual users of the project. During the two and half to three weeks of this phase, student testers review the code and submitted test cases for TC, TV and CC and provide feedback accordingly. This process of reviewing is same as in approach 1. Apart from reviewing the submission, the student testers also perform testing and reporting bugs as they find, like in approach - 2. Part of a student's score also depends on how well the reviews are rated and how well the testing duties are full filled.

Deploy

After the Improve phase, the student developers have 3 weeks of time to fix the bugs found and implement the feedback provided during the Improve phase.

Scoring Methodology

The factors that contribute to the students' final score are:

- How well is the code tested?
- How well did the students meet the requirements?
- How well did the student perform the reviews?
- How well did he fulfill the tester duty?
- How many bugs have been fixed?

Therefore the final score would be computed as:

Score = {(TC*TV*CC) + RM + (BF + NBF) + RS}

Where—

TC: Test case completeness

TV: Test case validity

CC: Code correctness

RM: Requirements Met

BF: Bugs fixed

NBF: Bugs found

RS: Review score

Since the emphasis is on how well students test their code, we made the testing score as a product of the three factors TV, TC and CC. This way, a student must receive a good grade on all three criteria in order to get a good score on the project. This way we make sure that students perform sufficient testing on the code they wrote and complete all the responsibilities assigned to them.

The advantages of this approach are similar to the advantages of both approaches 1 and 2.

Chapter 5: Conclusions

The two implementations of the BID framework could be used by students in the classroom to develop code. We believe it would help them improve the quality of the code. Instructors can use different processes [methodologies] for each of the build, improve and deploy phases. Each of these phases is independent of the others, and the usage of one process in a phase does not affect the other phases.

Table 5: Approach – 1 vs. Approach – 2

Approach – 1	Approach – 2
Test first, code next	Code first, test next
Multiple projects – in course duration	Multiple projects are parts of the same project
Peer reviewed for TC, TV and CC.	Peer reviewed (tested) to find bugs.
No beta deployment	Beta version deployed
Grade on how well the test cases are written	Grade on how many requirements are met and how many bugs fixed

Approach 1 makes use of the best practices from the processes suggested by Nicole Clark [b], Stephen H. Edwards [e, t, and u], and Sulayman K Sowe, Ioannis Stamelos and Ignatios Deligiannis [a]. Approach 1 is based on test-first strategy that encourages students to write test cases first, even before they start coding. This ensures that sufficient test cases are written for the code and thereby improves the code quality. The feedback provided through peer review also facilitates improvements in the code and test cases. It helps students with

any questions they have during development. Finally, to ensure that students write enough valid test cases, their work is grading against criteria like code correctness, test-case validity and completeness. This determines whether the code is correct and well tested. All the abovementioned factors help improve the quality of the code developed by students, and therefore help this software to be used as OSS.

Approach 2 makes use of the F/OSS testing framework with slight modifications to suit the classroom environment. This process suggests that students start on their final project very early during the course of the project and then submit a beta version of their project by a deadline. The instructor chooses a set of beta testers who act as virtual users and test the project and report bugs. This way student gets his submissions tested and has enough time to fix bugs and integrate the enhancements. Since students start very early with the project, they should have ample time to code and fix the bugs. Also getting the project tested by others helps in finding and fixing bugs.

Therefore, it would be very useful to follow the BID framework in classrooms and this way we can increase the chance of student contributions being usable as OSS.

References

a) Sowe, Sulayman K, Stamelos, Ioannis, Deligiannis, Ignatios, "A Framework for Teaching Software Testing using F/OSS Methodology", *2nd International Conference on Open Source Systems (OSS2006)*, Como, Italy, 8 - 10 June 2006, pp. 261-266.

b) Nicole Clark, "Peer testing in Software Engineering Projects", *Proceedings of the sixth conference on Australasian computing education*, Dunedin, New Zealand, 2004, pp. 41 - 48.

c) Michael Fagan, "Design and code inspections to reduce errors in program development", *Software pioneers: contributions to software engineering*, 2002, pp. 575 - 607.

d) Thomas B. Hilburn, "Inspections of formal specifications", *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, Philadelphia, US, 1996, pp. 150 - 154.

e) Stephen H. Edwards, "Improving Student Performance by Evaluating How Well Students Test Their Own Programs*", Journal of Educational Resources in Computing*, September 2003.

f) Audris Mockus, Roy T. Fielding and James Herbsleb, "A Case Study of Open Source Software Development: The Apache Server", *Proceedings of the 22nd international conference on Software engineering, Limerick*, Ireland, 2000, pp. 263 - 272.

g) Cho, K. and Schunn, C., "Scaffolded writing and rewriting in the discipline: A web-based reciprocal peer-review system", *Computers & Education*, 2007, pp. 409–426.

h) Cho, K., Schunn, C.D., and Wilson, R. W., "Validity and reliability of scaffolded peer assessment of writing from instructor and student perspectives*", Journal of Education Psychology*, 2006, pp. 891–901.

i) Hamer, J., Ma, K. T., and Kwong, H. H, "A method of automatic grade calibration in peer assessment", *Proceedings of the 7th Australasian Conference on Computing Education,* Newcastle, New South Wales, Australia, 2005, pp. 67-72.

j) Ju Long, "Open Source Software Development Experiences on the Students' Resumes: Do They Count? - Insights from the Employers' Perspectives", *Journal of Information Technology Education*, 2009.

k) Gary K., Koehnemann H.,et al, "A Case Study: Open Source Community and the Commercial Enterprise", *Sixth International Conference on Information Technology: New Generations*, Las Vegas, Nevada, 2009, pp. 940 - 945.

l) Raghunathan S., Prasad A., Mishra B.K., Hsihui Chang, "Open Source Versus Closed Source: Software Quality in Monopoly and Competitive Markets", *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, November 2005, pp. 903 - 918.

m) T.O'Reilly, "Lessons from open-source software development," *Communications of ACM*, April 1999, pp.32–37.

n) Robert Charles and Yonglei Tao, "Evaluating Student Participation in Open Source Software Development with an Annotation Model", *The Fourth IASTED International Conference on Knowledge Sharing and Collaborative Engineering*, KSCE 2006.

o) "Gartner Says as Number of Business Processes Using Open-Source Software Increases, Companies Must Adopt and Enforce an OSS Policy", www.gartner.com/it/page.jsp?id=801412, Nov 17, 2008.

p) Sandro Morasca, Davide Taibi, Davide Tosi, "Towards Certifying the Testing Process of Open-Source Software: New Challenges or Old Methodologies?", *ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, 2009, pp. 25 - 30.

q) Eric Allen, Robert Cartwright and Charles Reis, "Production Programming in the Classroom", *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, 2003, pp. 89 - 93.

r) Olly Gotel, Christelle Scharff and Andrew Wildenberg, "Teaching Software Quality Assurance by Encouraging Student Contributions to an Open Source Web-based System for the Assessment of Programming Assignments", *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, 2008, pp. 214 - 218.

s) Bojan Tomić, Siniša Vlajić, "Functional testing for students: a practical approach", *ACM SIGCSE Bulletin, 2008*, Vol. 40, Issue 4, pp.58 - 62.

t) Stephen H. Edwards, "Using software testing to move students from trial-and-error to reflection-in-action", *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, 2004, pp. 26 - 30.

u) Stephen H. Edwards, "Teaching software testing: automatic grading meets test-first coding", *Conference on Object Oriented Programming Systems Languages and Applications*, 2003, pp. 318 - 319.

v) "Clover", Code coverage analysis tool http://www.atlassian.com/software/clover/

w) rcov: code coverage for Ruby http://eigenclass.org/hiki.rb?rcov

x) Stephen H. Edwards, "Rethinking Computer Science Education from a Test-first Perspective", Conference on Object Oriented Programming Systems Languages and Applications, Anaheim, CA, USA, 2003. pp. 148 - 155.

Appendix

Table 6: Results of Student Survey

What is the duration of the project in weeks? (Project assignment date to project final submission date)	Average number of hrs./day you worked on the project.	What was the size of your project team?	Out of the total time spent on the project, what percentage of the time was spent on writing test cases and testing the code?	Did you develop test cases before you started coding?	What is the percentage of code covered by your tests?
4	2	2	65	Yes	100
4	3	4	15	No	100
3	3	2	10	No	20
3	2	2	50	No	
4	2	4	50	No	90
3	5	4	20	No	90
3	14	4	2	Sometimes	100
2	4	2	50	Sometimes	--
5	4	3	2	No	--
2	1	1	20	No	90
3	6	4	1	No	--
2	1	4	10	No	--
3	3	1	1	Sometimes	80
3	3	2	85	Sometimes	65
3	5	2	2	No	15
3	4	4	20	No	20
4	2	2	50	No	70
3	4	2	60	No	70
4	2	1	40	No	90
3	3	2	40	No	60
3	3	4	5	No	30
4	2	7	20	No	80
4	3	4	20	No	--
4	3	4	90	No	--
2	4	2	50	No	80
2	1	2	40	Yes	50
3	4	3	20	Yes	80
2	4	2	40	Yes	80
3	3	4	40	No	40
2	1	1	0	No	--
1	2	2	10	No	--
3.03	3.32	2.80	29.93	Y- 4, N - 23,S- 4	68.18