2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

Code Repurposing as an Assessment Tool

Joseph Sant
Faculty of Applied Science and Technology
Sheridan College
Oakville, Canada
joe.sant@sheridancollege.ca

***Abstract*— Code repurposing is often used for system development and to learn both APIs and programming techniques. Repurposing code typically requires that you understand the code first. This makes it an excellent candidate as an assessment tool in computer science and software engineering education. This strategy might also have a special application in combatting plagiarism. This paper discusses experiences using code repurposing as an assessment tool in different courses and with different sections.**

***Index Terms*—Assessment, code repurposing, plagiarism, software development, cloze testing.**

I. Introduction

A commonly used repurposing technique is "Copy and Paste" programming. It involves copying code from some existing code base and incorporating it in the software for a new task. There are many resources that facilitate code repurposing using "Copy and Paste". Code sharing sites such as www.java2s.com provide thousands of code samples that are both tagged and categorized. www.stackoverflow.com and other "question and answer" sites often provide complete running code samples. Youtube (www.youtube.com) has many programming tutorials which reference code samples. One would expect the availability of these resources to change software developer workflow and it has. The modern developer's workflow includes code foraging, the use of "question and answer" sites and references to online API documentation.

This presents educators with a dilemma. The ability to quickly use another developer's code to either repurpose or learn from is a valuable job skill that is worth incorporating into curricula. There is also a darker side to code foraging; code plagiarism. Students might submit solutions using 'borrowed' code that they don't understand. The educator is faced with a continuum between constructive foraging on the one hand and plagiarism on the other.

The Stanford University HCI Group is studying this approach, now termed "opportunistic programming". Their studies have included lab studies of web use of students completing a task and analysis of the web logs of an online programming portal. They found that programmers were using the web for "just-in-time learning", to refresh their knowledge of complicated syntax and to clarify and extend their existing knowledge [1]. Whether done consciously or not, many Computer Science and Software Engineering courses promote code repurposing. Instructors provide code samples in class or on the course website then assign exercises that involve solutions that could be solved by incorporating snippets from the class examples.

"Copy and Paste" programming taken to an extreme is plagiarism. Source code plagiarism in education is a broad area. A search on *scholar.google.com* for [source code plagiarism] returned 28,900 results (September, 2014). A discussion of the many issues relating to source code plagiarism is not feasible here. Instead we will briefly discuss the issues that pertain to the strategy proposed in this paper:
the high costs of prosecuting plagiarists and the ambiguities in assessing plagiarism.

Pursuing a plagiarism incident can be expensive. Once a plagiarism incident has been detected the instructor often must follow time-consuming procedures to process the incident [2]. This might explain the disconnect between surveys of faculty indicating that they believe that plagiarism is significant in their institutions while the actual number of reported cases within the institution is minimal [3]. There are real and/or opportunity costs involved in prosecuting student plagiarism. Real costs will result if more staff is hired to deal with plagiarism or if legal costs are incurred. There could also be opportunity costs. The time spent by faculty dealing with plagiarism is time that could be spent on a variety of tasks that could improve the course (e.g. developing materials, improving delivery, updating material). Likewise, the time spent by departmental or faculty administration on plagiarism could also be spent more fruitfully (e.g. enhancing communications).

Cosma and Joy[4] conducted an opinion survey of UK academics on source-code plagiarism. The study found uncertainty amongst academics on whether reuse without acknowledgement constitutes plagiarism. This ambiguity is problematic since it could make prosecuting plagiarism charges more difficult and could result in inconsistent application of penalties.

There might be a special role that repurposing assignments can play to combat source code plagiarism. In "Plagiarism: The Good Practice Guide", one interesting observation is that when students were asked why they didn't copy others work many said "they didn't take shortcuts because coursework was necessary to understanding and that they would need to demonstrate understanding in another context"[5]. Instead of detecting and prosecuting plagiarism why not pre-empt it. If assigned course work is assigned a minimal value relative to repurposing assignments there is little benefit to plagiarism.

978-1-4799-1934-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICSE.2015.158

ICSE 2015, Florence, Italy
Joint SE Education and Training

II. PROCTORED REPURPOSING ASSIGNMENTS

This paper describes a strategy which supplements standard (and uncontrolled) exercises with highly controlled in-class repurposing exercises. These repurposing exercises might be based on one or more standard assignments. In the trials, these repurposing exercises were delivered much as an open-book written exam would be. Because the knowledge gained from a standard assignment was being purposely re-evaluated in another controlled assessment, uncontrolled assignments can safely be devalued. The strategy is probably most useful in highly applied courses, such as business application development or web application courses. In these application areas there are often standard problems with widely accepted solutions.

A. Implementation

The implementation of the approach was as follows:

- Advise in course introduction that in-class coding tests will be given in addition to standard exercises.
- Advise students a week before tests which assignments the coding exercise will be modeled on.
- Use 60 to 90 minutes of class for coding exercise.

The in-class coding exercises were administered as open-book written tests. Students were allowed to bring in as many written or printed resources as they wished. No electronic devices were allowed. The choice of using a written test will be discussed in *Discussion* section. Typically the students will be given a problem narrative, sample data, and a partially completed solution. When there is a visual component to the exercise, screenshots of the desired user interface are provided. Sometimes students were asked to finish off a partial solution. The practical and pedagogic benefits in providing a partial solution will be discussed later.

Where possible, the repurposing exercises were designed so that they were dependent a given data structure or a given document structure. This combined with the provision of partially completed code made it easier to produce multiple versions of an assignment. For example, the exercise might require that a remote XML document be consumed and processed to produce some result. Minor changes in the data structure or document structure would require different solutions. For example, representing data in an XML document as an element in one version, and attribute in another require different solutions. An example more pertinent to Introductory Programming courses would be to require the implementation and use of an inheritance hierarchy, where the base classes were different. Over time, some very simple refinements in administering the repurposing exercises were developed. If the problem narrative and data structure example could be reduced to one page, the test could be given on a single 11" by 17" page. Since the students might have to revisit the data structure example in their solution, it was convenient for it to be always visible. This format was very useful from the marker's perspective as well. Often the main difference between assignment versions was the data or document structure and perhaps the names of some variables in the provided code. In these cases, the marker would only need a quick glance to see if the students properly addressed the special features of the version. This also defends against mixing up tests with the wrong addenda.

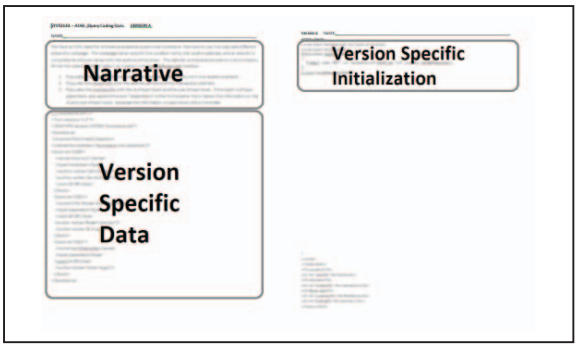

Fig. 1. Sample Repurposing Exercise.

We will look at one assignment pairing in detail. In a Web App Development Course, a standard assignment was developed that asked students to use AJAX to retrieve and parse a remote JSON file containing information on each planet in the solar system then to graphically represent the planets using Javascript and HTML5 graphics. The assignment was assigned a value of 2% of the final grade. Students were then asked to repurpose their code to retrieve a JSON file containing descriptors for graphics shapes. Students were expected to trap specific error conditions that might occur. The repurposing specifications provided a sample of the remote file with the expectation that the student be able to interpret which items in the file were necessary to implement the solution. There were minor differences in naming or structure in the data file samples. The repurposing exercise was valued at 10% of the grade. 80 minutes was allowed for completion. Assignments were marked manually according to a rubric.

A short list of assignment pairings is below:

TABLE I. SAMPLE ASSIGNMENT PAIRINGS

Preparatory Assignments	Repurposing Task
Sequence leading to key-driven HTML5 2-D Board game.	Key-driven board game with movement in just 1 dimension and with different behavior.
Responsive Web App with forms, lists, multiple pages. Form info loaded from remote JSON file	Alter partially completed multipage app by adding pages and functionality. Implement form
Draw solar system to scale accessing remote XML or JSON file with planet data.	Execute graphics drawing functions based on remote XML or JSON files.

As administered, uncontrolled standard assignments were reduced to 3, 2, 1 or 0% of the total grade. These could be formally submitted or simply lab-checked. The value of the repurposing exercises varied between 7% and 12%. Typically there would also be a major group project worth between 15% and 30% of the grade. In some cases, there might be an additional individual project. For several of the sections, a

repurposing exercise similar to those described in this paper was given as a component of the midterm and/or final exam.

A proper study of the efficacy of any assessment strategy would require multiple instructors and multiple courses, preferably from different institutions. Studying the efficacy of a strategy used to combat plagiarism is even more problematic. How do you assess the prevalence of plagiarism? The introduction discussed some of the problems with detecting plagiarism. Thus far the strategy has only been used by one instructor in seven different courses. As a result, this paper will focus on an experiential report and a preliminary analysis of the grade distributions of different assessment techniques.

The grade distributions of 68 students in four sections for standard assessment types (multiple choice and exercises) were compared to those of the repurposing test. A *Cloze test* was also added to the mix. A *Cloze* test is commonly used to test language comprehension in spoken languages. Scores on well-designed Cloze tests have been shown to be highly correlated with free-response comprehension tests [5]. It involves blanking out every *nth* word from a passage then asking the student to fill in those blanks. The Cloze test was modified so that specific tokens from a code passage were blanked out rather than automatically blanking every nth token. Scatter plots for the comparisons of the repurposing tests against the Cloze test and exercises are shown below.

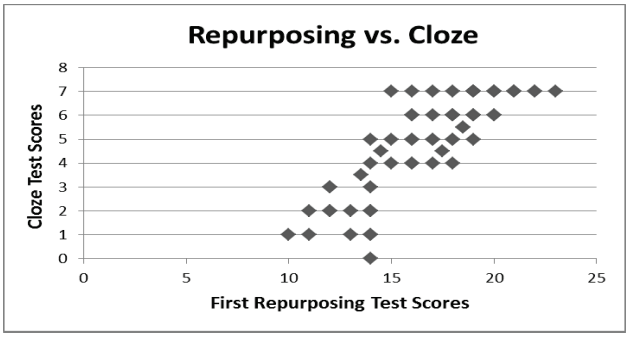

Fig. 2. Repurposing Test Scores vs. Cloze Test.

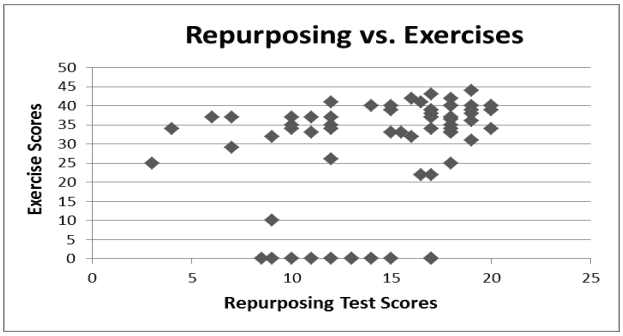

Fig. 3. Repurposing Test Scores vs. Exercises.

There was medium correlation between performance on the repurposing test and the cloze test (r=.55, n=68). The correlation between the repurposing scores and the multiple choice component of the midterm was .357 and between repurposing scores and exercises it was .47.

III. Discussion

When dealing with assessment tools there are many questions that need to be answered, especially if the assessment is intended to address the growing issue of plagiarism:

- What is the goal of the assessment and how effective is it in achieving those goals?
- Is it practicable?
- Under what conditions should it work well?
- Under what conditions might it work poorly?
- How well does it address the issue of plagiarism?

Code repurposing tests are intended to assess a student's ability to comprehend a code sample and use it to create a new solution based on different specifications (e.g. different outputs, different inputs). There was good correlation (r=.55) between a Repurposing Test and a Cloze question dealing with the same problem. Cloze tests are intended to test the student's code reading comprehension. The two tests were given two weeks apart so that may have affected the results. Traditional hand-in exercises on different material were compared to the results in the code repurposing tests. Here the correlation was weaker but still significant (r = .47). Scatter plots comparing Repurposing tests versus Cloze Test and Repurposing tests versus Exercises show a different distribution pattern. This deserves further investigation. It should be noted that if enough students did not complete their assignments independently we wouldn't necessarily expect a strong correlation between Repurposing Tests and Exercise scores.

Code repurposing tests are definitely practicable. The marking effort is typically less than a hand-in exercise since the instructor has the option of providing the student with a partially completed program. They are more difficult to mark than a multiple choice or short answer question test. Creating a code repurposing test should be easier than developing a multiple-choice test if the authors of the multiple-choice test do not have access to a question bank or previous exams.

The method is not considered to be applicable to all elements of a software development course. The proper assessment of a student's progress might require several different methods. There are obvious limits to the complexity of a task that you can assign to be completed in a 90 minutes. The approach, as implemented, was not used to evaluate software design, user interface design, or software testing.

Empirically assessing the efficacy of the method in addressing code plagiarism is a difficult task. It likely will require multiple years and multiple institutions. As a preliminary step we might look instead at whether it is an enabler or inhibitor of best practices. In "Plagiarism: The Good Practice Guide" [6], Carroll and Appleton discuss best practices for designing out opportunities for plagiarism. Two techniques suggested are to change your assessments every year and to create individualized assignments. Both these techniques are problematic for instructors of software development courses. Common components of a software development assignment document are a problem narrative based on a specific scenario, a specification for the submission,

sample or test data, and sample output or look and feel. With a cohort of 80,100 or 200 students how many scenarios and sample data sets would be sufficient to protect against plagiarism. Even if such an approach was practicable, we are left with two major problems. One is that this huge investment in time and effort doesn't protect against "helpers" (cousins, friends, discussion board participants at StackOverflow.com) who in helping actually end up creating the bulk of the solution. The other is pedagogic. Assignments can be made more interesting by using scenarios that resonate with the students. In an attempt to deal with plagiarism you could exhaust all interesting scenarios in one year.

The approach of using a pairing of standard assignments with associated controlled repurposing assignments makes these two techniques for designing out opportunities for plagiarism practicable. For each pairing, only one version of the standard assignment was given but multiple versions of the repurposing assignment were developed. Because of their lower value, even if plagiarism or unconstructive collaboration did occur on the standard assignments, it would have a reduced effect on the final grade. Only as many versions of the repurposing exercise were created to prevent plagiarism in a test situation (this was based on the size of the room, the size of the class and the number of different sections). In many cases the versions were easy to create since it would only entail the changing of a document structure, or the structure of a JSON or XML file, and perhaps changing some variable names in the provided source code. Sometimes this involved only a few global "search and replace" actions applied to a master document. "Cosmetic" changes are possible in other courses (e.g. Changing the table structure in an application that uses relational databases).

One contentious issue is the appropriateness of asking students to hand write code rather than use proper lab tests. This issue has been discussed exhaustively on one of the SIGSCE discussion boards [7]. Each approach has its advantages and disadvantages, and each technique could probably be used interchangeably here. Whether to implement the repurposing strategy as a written test or a formal lab test is a personal choice. The personal choice of using written tests is based partially on a healthy respect for students' ability to game the system given enough motivation. This is based on directly asking students about common exploits and reports from other teachers. In one case, the students had exploited a simple flaw in an expensive commercial software package purchased to lock down students' computers during online tests. The exploit was not recognized until 2 years after it had been in wide use. The exploits change over time suggesting that it might be difficult to keep ahead of the exploits. There are also practical benefits to written tests. Written tests do not require the in-class setup time that would be required for lab tests. Pencils and papers do not crash in the middle of a test.

IV. Summary

Repurposing code is a technique that is widely used in the software industry. Repurposing code is used because it can increase both productivity and quality. To repurpose code, it is necessary to first understand the logic of the code you are repurposing. These features of repurposing can be used to advantage in the assessment of student performance in software development courses. Asking a student to repurpose code rather than create a solution from scratch means that assignments can be given with a reasonable expectation that they be finished independently in a single class period. These controlled assignments can now act as a measure of the understanding of the original code. Traditional assignments are given as before but they are paired with a controlled in-class repurposing assignment. The traditional assignments can be given a low weighting or not counted at all. This scheme reduces the benefit to the student of submitting assignments which they don't understand, either because they plagiarized or they received too much help in their completion. A side benefit of the method is that it is an enabler of best practices in student assessment. It is considered a best practice to change your assessments every year and ideally to assign multiple versions (isomorphs) or the assignment to control plagiarism. This is often impractical for traditional assignments. Given the controlled conditions of the in-class repurposing assignment, it becomes practical to provide multiple versions. Another benefit is that it incorporates code repurposing, an important job skill, into the curriculum.

References

[1] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, Scott R. Klemmer. "Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code", CHI: ACM Conference on Human Factors in Computing Systems, Boston, MA, 2009.

[2] "Kent State Plagiarism Flowchart", Internet:www2.kent.edu/academics/resources/plagiarism/images/Plagiarism-Flowchart-Kent_1.JPG, Oct.8, 2014.

[3] Garner, Andrew D., and Larry Hubbell. "Institutional models for adjudicating plagiarism in the United States." *International Journal for Educational Integrity* 9.1 (2013).

[4] Cosma, G., & Joy, M. 2008,. "Towards a definition of source-code plagiarism. Education", IEEE Transactions on, 51(2), 195-200.

[5] Gellert, Anna S., and Carsten Elbro 2012 "Cloze tests may be quick, but are they dirty? Development and preliminary validation of a cloze test of reading comprehension." *Journal of Psychoeducational Assessment* (2012): 0734282912451971.

[6] Carroll, J. and Appleton, J. 2001. "Plagiarism: A good practice guide". Tech. rep., Joint Information Services Committee. Available: http://www.jisc.ac.uk/index.cfm?name=project_plag_practise [28th December 2013].

[7] "SIGSCE Discussion Board", http://listserv.acm.org/scripts/wa-ACMLPX.exe?A1=ind1305C&L=SIGCSE-members#7Tennessee, United States, February 26 - 28, 1995). K. M. George, J. Carroll, and D. Oppenheim, Eds. SAC '95. ACM, New York, NY, 10-13. DOI= http://doi.acm.org/10.1145/315891.315894.