# ASSESSING STUDENT LEARNING IN SOFTWARE ENGINEERING[*]

*Taehyung (George) Wang, Diane Schwartz, and Robert Lingard*
*Department of Computer Science*
*California State University Northridge (CSUN)*
*18111 Nordhoff Street*
*Northridge, CA 91330*
*{twang, diane.schwartz, rlingard}@csun.edu*

**ABSTRACT**

Program assessment plays a key role in measuring student learning and improving the program. Assessing programs is an iterative and incremental process that consists of a series of activities conducted by stakeholders such as faculty members, students and alumni. These activities include defining learning outcomes, developing assessment methods and rubrics, conducting assessment, analyzing assessment results, and recommending actions for change. Over two years, we, as the software engineering group in the Computer Science Department, assessed four student learning outcomes relating software engineering. In this paper, we describe our assessment activities for two learning outcomes: 1) Demonstrate an understanding of the principles and practices for software design and development; 2) Be able to apply the principles and practices for software design and development to real world problems.

## 1. INTRODUCTION

Higher education is changing from teacher-centric to learner-centric [1]. The learner-centric approach emphasizes students' active engagement, while the faculty develop courses and curriculum in such a way that students are able to achieve the *student learning outcomes of the program.*

The curriculum generally consists of several program areas and each program area encompasses related courses. For example, the Department of Computer Science at

[*] Copyright © 2008 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

CSUN defines seven program areas (fundamental concepts, systems, language/theory, software engineering, societal issues, communications, and career/lifelong learning). Among them, software engineering is the largest program area in terms of the number of related courses: Fourteen courses are directly or indirectly related to the software engineering program. "Learning outcomes are statements that describe what students are expected to know and/or be able to do by the time of graduation" [3]. On the other hand, course objectives describe skills, content, or tools that students will master by the end of a course. Each course objective has either a strong or partial connection with one or more learning outcomes. For example, a course objective, "Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a software system of non-trivial size," is strongly connected to the learning outcome, "Demonstrate an understanding of the principles and practices for software design and development," and partially connected to another learning outcome, "Be able to effectively communicate orally." In general, course objectives are aligned with the student learning outcomes and then the student learning outcomes are assessed. We call this assessment student learning outcome assessment or program assessment.

Assessment is a key factor in learner-centric education [1]. It is a continuous process that improves student achievement as well as improving the curriculum. Six basic activities are conducted for the student learning outcome assessment: 1) Identify and develop student learning outcomes; 2) Develop an assessment plan; 3) Determine assessment methods; 4) Develop the assessment metrics or rubrics; 5) Collect and analyze assessment data; 6) Recommend actions for changes to improve the program. Each activity is described below.

Student learning outcomes (SLOs) should be used to design both the curriculum and the individual courses. It is important that faculty incorporate activities into their courses that support the SLOs. For example, if they are not provided appropriate instruction, assignments, and homework, students are unlikely to apply the principles and practices for software design and development to real world problems [1].

There are two major assessment methods: *direct* and *indirect*. Direct assessment requires students to demonstrate what they have achieved. For example, by having students take an assessment test, we could see whether or not students understood the principles of software design. Indirect assessment of student learning involves the inference of student abilities and knowledge rather than the observance of direct evidence of learning or achievement [1]. It is *formal* if a careful and reliable assessment study has been planned and carried out. If not, it is *informal*, for example, when a group of instructors just get together and discuss/evaluate student achievement with respect to some outcome. This may involve an examination of actual student work (direct assessment) or just a subjective opinion from faculty about how well students seemed to learn (indirect assessment) [2].

Rubrics are used to classify the assessment results into a series of evaluative categories. A well-designed rubric helps evaluators judge student achievement of the student learning outcome while also generalizing the detailed test data. For example, four categories, "unacceptable," "marginal," "adequate," and "excellent," were used to assess the first student learning outcome (refer to Section 2.1.2 for details), and three categories, "strong," "acceptable," and "weak" were used for the second student learning outcome (refer to Section 3.1 for details).

The next step in the assessment process is to analyze the collected data. A statistical analysis of the data can provide valuable information needed to improve the course, the program and even the test instrument. In particular, 1) we used a one-way Analysis of Variance (ANOVA) to obtain statistical inferences relating test scores to the demographic questions, and 2) we correlated the scores on individual questions with the total scores to evaluate the validity of the individual questions.

Once the data analysis is complete, the assessment team suggests recommendations for program improvement. For example, we recommended that software engineering principles and practices be both introduced in lower-division courses and reinforced in relevant upper-division courses. These recommendations then needed to be approved by the department.

In this paper, we describe the assessment activities conducted for two student learning outcomes, 1) *Demonstrate an understanding of the principles and practices for software design and development (SLO1 hereafter);*2) *Be able to apply the principles and practices for software design and development to real world problems (SLO2 hereafter).*

The paper is organized as follows. In Section 2, we describe the assessment activities for SLO1. In Section 3, the assessment of SLO2 is described. Section 4 summarizes this work.

## 2. THE ASSESSMENT OF SLO1

The student learning outcome, *Demonstrate an understanding of the principles and practices for software design and development*, was assessed formally and directly once during the 2004/2005 academic year and once during the 2005/2006 academic year.

### 2.1 The 2004/2005 Assessment

#### 2.1.1 The Methods of Assessment

An iterative process was used to assess SLO1. This process consists of five activities: 1) Develop assessment test questions; 2) Develop scoring rubrics for the assessment of SLO1; 3) Conduct the assessment test; 4) Analyze test results; 5) Prepare an assessment report. Seven faculty members and 74 students participated in this assessment. Students answered 26 questions including six demographic questions. These six questions were about the current student class level, the year and institution the students took the software engineering course, the number of undergraduate-level software engineering classes the students took, and the number of graduate-level software classes the students took. The 20 assessment multiple-choice questions covered software requirements and specification (4 questions), design (11 questions), testing (3 questions), and computer-aided software engineering (CASE) tools (2 questions).

### 2.1.2 The Results of Assessment

Before conducting the assessment test, scoring rubrics based on the number of questions answered correctly were established as shown in Table 1:

| | |
|---|---|
| **Unacceptable** (0-6 correct answers) | Evidence that the student has mastered this learning objective is not provided, unconvincing, or very incomplete. |
| **Marginal** (7-11 correct answers) | Evidence that the student has mastered this objective is provided, but it is weak or incomplete. |
| **Adequate** (12-16 correct answers) | Evidence shows that the student has generally attained this objective. |
| **Excellent** (17-20 correct answers) | Evidence demonstrates that the student has mastered this objective at a high level. |

Table 1. Scoring rubric for assessing SLO1

The percentages of students scoring Excellent, Adequate, Marginal, and Unacceptable were 4%, 38%, 46%, and 12%, respectively, as shown in Figure 1. The mean score of the students was 10.65 correct answers out of 20 questions, which falls in the Marginal category. Given the test scores and demographic information, the test results were analyzed by running a one-way ANOVA and comparing mean test scores with demographic data. As a result, we obtained two statistically significant inferences: 1) *The mean score of the graduate students is higher than that of the undergraduate students;* 2) *The mean score of the students who took a graduate-level software engineering class is higher than that of the students who did not.* Note that undergraduate students are allowed to take 500-level graduate courses such as software verification and validation, software engineering economics, and software engineering metrics. In this assessment process, two undergraduate and ten graduate students had taken at least one graduate-level software engineering course before they took the assessment test. See Figure 2 and Figure 3 for significant inferences.

We also analyzed the assessment questions by examining the frequency of the right answers for each question. For example, as shown in Table 2, more than half of the students (56.8%) chose answer A as the answer to the question 15; only 29.7 percent of the students chose the correct answer C.

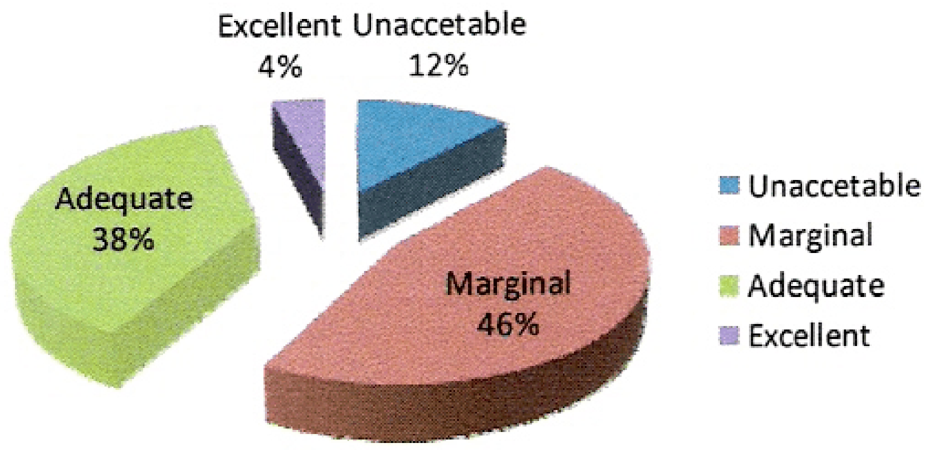

Figure 1. Assessment results for SLO1

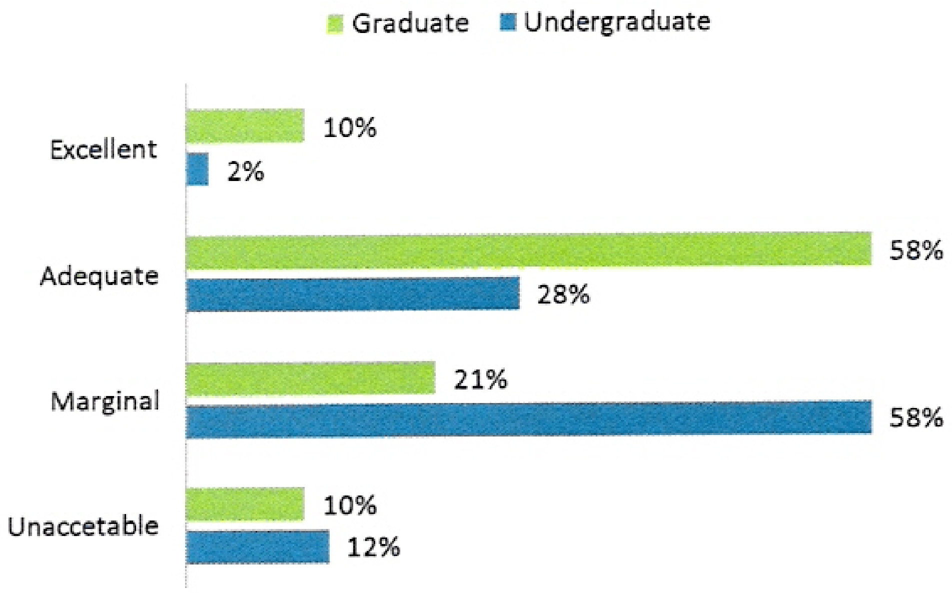

Figure 2. Comparison of test scores with student class

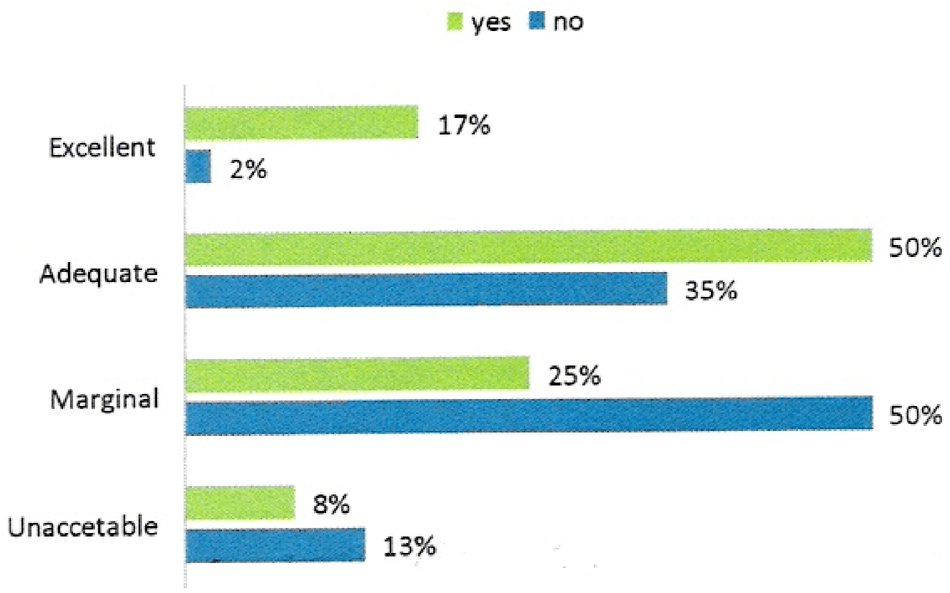

Figure 3. Comparison of test scores between students who took graduate-level software engineering courses and those who did not

| | Q15 (correct answer = C) | | | |
|---|---|---|---|---|
| | **Frequency** | **Percent** | **Valid Percent** | **Cumulative Percent** |
| **A** | 42 | 56.8 | 56.8 | 56.8 |
| **B** | 1 | 1.4 | 1.4 | 58.1 |
| **C** | 22 | 29.7 | 29.7 | 87.8 |
| **D** | 7 | 9.5 | 9.5 | 97.3 |
| **E** | 2 | 2.7 | 2.7 | **100.0** |
| **Total** | **74** | **100.0** | **100.0** | |

Table 2. A frequency table for the question for which more than half students chose a wrong answer

### 2.1.3 The Recommendations Based on the Assessment Results

Given the first iteration of assessment results, it was recommended that: 1) software engineering concepts be reinforced in the senior elective courses which require the junior-level course COMP 380/L entitled "Introduction to Software Engineering" as a prerequisite; 2) software engineering concepts be introduced in lower-division core courses; 3) assessment test questions be reevaluated and some questions be replaced with new ones.

### 2.2. The 2005/2006 Assessment

#### 2.2.1 The Methods of Assessment

The second iteration of the formal assessment of SLO1 was conducted with the same process model used in the first assessment. Ten faculty members and 82 (undergraduate and graduate) students participated.

#### 2.2.2 The Results of Assessment

The mean score on the assessment test was 11.51 correct out of 20. The categorized results were 7.3% (6 out of 82) of the students were in the unacceptable category; 41.5% (34 out of 82) were in the marginal category; 43.9% (36 out of 82) were in the adequate category; 7.3% (6 out of 82) were in the excellent category. Based on the scoring rubrics above, the mean score, 11.51, fell between Marginal (7-11 right answers) and Adequate (12-16 right answers), respectively, as defined in Table 1.

#### 2.2.3 The Recommendations Based on the Assessment Results

Although the results of the second assessment showed slight a improvement over the first assessment, the higher scores are probably due mainly to the improvement of the assessment instrument (i.e., the replacement of ineffective questions). Since this assessment has been carried out twice with similar results, there is a valid concern that students do not retain their knowledge of the principles and practices of software engineering a year or more after taking the junior-level software engineering course. Under the current curriculum, COMP 380/L is the only required software engineering course in the curriculum. Many students might graduate without even taking any software engineering related upper-division courses.

It was therefore recommended that additional changes to the program be made to reinforce student's knowledge of software engineering principles and practices before students graduate from the program. Therefore, we recommended that students be required to complete a senior software engineering project as a requirement for graduation. Since several of our elective courses already require the completion of such a project (for example, database design and software system development), one way for students to satisfy the requirement would be to take one such course as part of their senior elective package. Over time, additional ways of meeting the requirement could be developed.

## 3. THE ASSESSMENT OF SLO2

The student learning outcome, *Be able to apply the principles and practices for software design and development to real world problems,* was assessed informally and directly during the 2005/2006 academic year.

### 3.1 The Methods of Assessment

The assessment was carried out by three faculty members who evaluated group project deliverables from two courses related to software engineering. We reviewed eight project deliverables submitted by six teams from the junior-level course, "Introduction to Software Engineering" and two teams from the senior-level course, "Software System Development." The project deliverables included software requirements, and specification documents, software design documents and source code documents. The assessment rubrics for this assessment were developed by the faculty evaluation team. The rubrics are as follows:

**Artifact:** Software Requirement and Specifications (SRS) Document

**Rubrics:** Building the correct SRS document is critical to the entire software life-cycle. The SRS document should be clear and understandable, and should address reasonable set of requirements in sufficient detail.

- Strong: The SRS document fully meets the set of SRS document assessment rubrics.
- Acceptable: The SRS document generally meets the set of assessment rubrics.
- Weak: The SRS document fails to meet one or more assessment rubrics.

**Artifact:** Software Design Document (SDD)

**Rubrics:** The SDD is a blueprint for building a software product. The SDD should include appropriate designs (for example, the Unified Modeling Language (UML) class diagrams and sequence diagrams), which properly reflect the SRS document.

- Strong: The SDD fully meets the set of SDD assessment rubrics.
- Acceptable: The SDD generally meets the set of SDD assessment rubrics.
- Weak: The SDD fails to meet one or more SDD assessment rubrics.

**Artifact:** Source Code

**Rubrics:** Source code should follow coding standards such as (file, class, methods, attribute, variable, and parameter) naming, indentation, comments, declarations, statements, white space, etc. Error conditions should also be handled.

- Strong: Source code fully meets coding standards.
- Acceptable: Source code generally meets coding standard.
- Weak: Source code fails to meet one or more code standards.

### 3.2 The Results of Assessment

The faculty members who evaluated the project deliverables found that 87.5% (7 out of 8) of the SRS documents evaluated were in the acceptable category. Although the documents contained some faults such as minor UML use case syntax errors and spelling errors, overall, the documents were well written with a glossary, a UML diagram and its associated use case descriptions, a system-level state diagram, and a system-level activity diagram.. One SRS document (12.5%) fully met the SRS document assessment rubrics. On the contrary, 62.5% (5 out of 8) of the SDDs evaluated were in the weak category mainly because of inconsistency with corresponding source code. For instance, the class diagrams did not include full descriptions about the attributes and methods that were shown in the corresponding source code. Three SDD (37.5%) turned out to be in the acceptable category, which generally satisfied the design rubrics. It turned out that 75% (6 out of 8) of the source codes were in the acceptable category. A lack of comments was the only weakness in these six source codes. Two source codes (25%), however, did not follow most of the coding standards.

Overall, we found that there was weakness in the SDDs. In the traditional software development process, each document should be complete, consistent, and correct before next phase begins. For example, SRS documents should be complete, before design starts. Similarly SDD should be complete before implementation begins. Even if an agile process model that emphasizes working code over software documentation is adopted, the final version of documents should be completed and correct. The assessment results showed, however, that SDDs were incomplete, and/or inconsistent with source code. It was suspected that the design documents were created before implementation was started, and were not updated when implementation was finished. Also, UML syntax errors were found in some SDDs.

### 3.3 The Recommendations Based on the Assessment Results

Given the assessment results, it was recommended that: 1) A formal assessment be done by organizing a panel whose members are former software engineering course instructors; 2) Rubrics be more specific and the project deliverables be reviewed; 3) Design activities be stressed in the software engineering courses; and 4) The basic design activities also be addressed in lower-level course courses.

## 4. CONCLUSIONS

Assessment is a key component of learner-centric education. Through a series of assessment activities, the curriculum and program can be continually improved. We conducted the assessment activities for the student learning outcomes of the software engineering program and obtained valuable lessons and information.

After assessing SLO1, we found important information: 1) *The mean score on the software engineering assessment test of the graduate students is higher than that of the undergraduate students;* 2) *The mean score of the students who took a graduate-level software engineering class is higher than that of the students who did not.* We concluded that if students take more software engineering courses, then they will have a better understanding of the software engineering concepts and principles.

As for the results from assessing SLO2, we found that there was a mismatch between design documents and source code which we attribute to a weakness in the students' ability to design software before implementing it, and a weakness in their ability to describe their design using formal/semi-formal modeling techniques.

Given the information obtained from the assessment of SLO1 and SLO2, we recommended that: 1) the concepts and principles of software engineering be introduced at the lower-division courses and as well as reinforced upper-division courses; 2) at least one upper-division elective course be assigned as a software engineering required course; 3) software design, particularly object-oriented design, should be stressed in the introductory software engineering course.

With regard to recommendation (1), modifications have been made to the objectives of several of our lower-division courses to ensure that fundamental software engineering concepts are introduced. Additionally, the objectives of upper-division courses involving software development projects have been reviewed and augmented as necessary so that appropriate software engineering concepts are reinforced. Recommendation (2) prompted significant discussion among the faculty and resulted in a decision to add a new capstone course to the program which, among other things, would require students to be involved in software development project, thus providing them with additional software engineering experience. A task force has been formed and is currently developing the necessary curriculum proposal for this change. Finally, recommendation (3) has been implemented by modifying the way our introductory course in software engineering is taught to place a greater emphasis on software design.

## REFERENCES

[1] M.J. Allen, Assessing Academic Programs in Higher Education, Anker Publishing Company, Bolton, Massachusetts, 2004.

[2] Robert Lingard, "A Process for the Direct Assessment of Program Learning Outcomes Based on the Principles and Practices of Software Engineering," ASEE Annual Conference & Exposition, Honolulu, Hawaii, June 24-27, 2007.

[3] G. Rogers, "Assessment for Quality Assurance," http://www.abet.org/Linked%20Documents-UPDATE/Assessment/QualAssurance_Corrected_May2004.pps, 2003.