Software Engineering Frameworks: Perceptions of Second-Semester Students

Kirby McMaster
Fort Lewis College, Durango, CO, USA
kmcmaster@fortlewis.edu

Samuel Sambasivam
Azusa Pacific University, Azusa, CA, USA
ssambasivam@apu.edu

Steven Hadfield
U.S. Air Force Academy, Colorado Springs, CO, USA
steven.hadfield@usafa.edu

Stuart Wolthuis
Brigham Young University-Hawaii, Laie, HI, USA
stuart.wolthuis@byuh.edu

Abstract

This research examines the frameworks used by Computer Science students at the conclusion of two semesters of study in Software Engineering. A questionnaire listing 64 Software Engineering concepts was given to students at three universities upon completion of their second-semester course. To identify which topics were most important, students were asked to rate each concept on a ten-point scale. From their responses, we calculated the average perceived importance for each concept. This paper analyzes the results of the survey. We also compare these concept ratings to similar ratings obtained earlier from a sample of first-semester Software Engineering students. Using both data sets, we describe how Software Engineering perceptions evolve as students progress through a two-semester course sequence. This knowledge can be valuable to Software Engineering instructors as they decide which concepts to emphasize and how to unite these concepts into a consistent, meaningful framework.

Keywords: Software Engineering, framework, schema, paradigm, mental model, concept, rating.

Introduction

Learning is more effective in a Computer Science course if topics and concepts are organized within an overall mental *framework*. Each concept is introduced as a piece of a puzzle. The framework allows the pieces to fit together into a meaningful *whole*. Other similar terms used by authors include *schema*, *cognitive style*, *paradigm*, and *mental model*.

According to Donald (2002), a course needs a schema to improve understanding.

> A *schema* ... is a data structure of generic concepts stored in memory and containing the network of relationships among the constituent parts.... If we are to understand the relationships between concepts, we need to know in what or-

> der and how closely concepts are linked and the character of the linkage.

Which schema is preferable for a given course? In an ideal world, course concepts would blend naturally into the general mental framework of each student. In *The Mathematical Experience*, Davis and Hersh (1981) observe that:

> People vary dramatically in what might be called their *cognitive style*, that is, their primary mode of thinking.

Ken Bain (2004) emphasizes the ubiquity of frameworks in education when he states:

> The students bring *paradigms* to the class that shape how they construct meaning. Even if they know nothing about our subjects, they still use an existing *mental model* of something to build their knowledge of what we tell them.

Frameworks, explicit or implicit, are available for most Computer Science courses. Some courses organize primary concepts into a *layered* framework, where services received at one layer are provided by algorithms and data structures in a lower layer. Computer network courses favor layers consisting of some blend of the OSI Model and the Internet Protocol Suite (Tanenbaum & Wetherall, 2011). Operating systems courses include topics from the hardware, kernel, system services, and user-interface layers (Silberschatz, Galvin, and Gagne, 2008). Database courses insert a DBMS layer between application programs and operating system files (Connolly & Begg, 2009).

Not all Computer Science frameworks are layered. A common framework for object-oriented programming (Lafore, 2001) includes groups of interrelated classes, arranged according to established *design patterns* (Gamma, Helm, Johnson, & Vlissides, 1994). Data structures course topics are divided into algorithm and data structure categories, such as stacks, queues, linked lists, searching, and sorting (Lafore, 2002). Artificial intelligence utilizes a variety of frameworks that describe search strategies, game playing, learning models, knowledge-based systems, and intelligent agents (Russell & Norvig, 2009).

But which frameworks are suitable for Software Engineering (SE) courses? The most common SE framework is a horizontal *life-cycle* sequence of stages for software development (McConnell, 2004; Sommerville, 2004). Some SE textbooks add a vertical dimension, separating the user interface, algorithms/business rules, and data components (Pressman, 2009). Other SE books promote a quasi-religious experience that endorses a particular development practice (Beck & Andres, 2004; Cockburn, 2006; Jacobson, Booch, & Rumbaugh, 1999; Beck, 2004).

In previous research (McMaster, Hadfield, & Anderson, 2008), we examined frameworks for software development from the viewpoint of textbook *authors*. We determined which words are used frequently in three samples of books: object-oriented programming, database, and Software Engineering. Our assumption was that the words used most often in a book suggest the framework of the author. A framework is certainly more than a set of concepts, but concepts are the building blocks used to construct frameworks. Frameworks help highlight and integrate the meaning of the concepts.

In this study, we sought to determine which concepts are considered most important by *students* after they had completed their second SE course (SE-II). We examined whether their concept ratings were consistent among students within a course. We also compared concept ratings across courses taught by different instructors at different schools.

The remainder of this paper is organized as follows. First, we present our methodology for gathering data on student ratings of SE concepts. Next, we analyze the results to determine which concepts students perceive as most important. We then look at ratings variation within courses and between schools. Last, we compare the second-semester SE ratings with ratings obtained in

an earlier study of first-semester SE (SE-I) students (McMaster, Hadfield, Wolthuis, & Sambasivam, 2011).

Methodology

A questionnaire listing 64 Software Engineering concepts was given to Computer Science students upon completion of their second SE course. All but one of the concepts are described by a single word or acronym (e.g. *agile, design, quality, UML*). The concept *use case* is presented as a word pair.

These concepts were selected from a variety of sources. First, we chose topics that appeared in Amazon concordances of Software Engineering books. The concordances list the 100 most frequent words (excluding common English words) in the books. We supplemented the concordance words with topics we felt were important, along with words recommended by other SE instructors. To encourage responses at the low end of the scale, we intentionally added several words that are not SE-specific (e.g. *activity, language*). Once the word list was compiled, the concepts were randomized so that there would be no implied significance to the order in which the concepts were presented to students.

The concept list was included on a survey given to samples of SE-II students at three schools. The School-1 sample consisted of 11 students at a small state university. The School-2 sample included 16 students from a small private college. The School-3 sample of 16 students was drawn from a larger private university. The combined sample size is 43. Almost all students were juniors or seniors. The course sections had different instructors and textbooks, but each sample of students received a traditional, project-oriented SE-II course.

To identify which SE concepts were valued most, students were asked to rate each concept on a 10-point scale, with 1 indicating “least important” and 10 indicating “most important”. From the responses, we determined the average perceived importance for each concept at each school.

We found that the means for the 64 concepts differed in a biased way between the three schools. To make the data for the samples comparable, we rescaled (*standardized*) the concept means for each school, so that the three sets of 64 concept means had the same average (7.20) and standard deviation (1.00). This rescaling changed the concept means only slightly.

We did not rescale individual student ratings. Rather, we rescaled the mean ratings in a way that preserved the ordering of concepts within each school. We could have achieved a similar result by converting the means to ranks, but then the concepts would have been equally spaced (except for ties).

After gathering and transforming the survey results, we had two sets of data to analyze and compare: (1) the concept ratings from SE-II students as described above, and (2) similar ratings from a previous study of SE-I students using the same SE concepts questionnaire. We first examine the SE-II concept ratings for the three schools, both separately and combined. Next, we look at the ratings variation for each concept within courses and between schools. Finally, we compare the combined SE-II ratings with concept ratings collected earlier from SE-I students.

Concept Ratings

In this section, we analyze the concept ratings for the three SE-II student samples. Table 1 presents the 20 top-rated Software Engineering concepts (out of 64), along with the rescaled means for School-1, School-2, and School-3. We include a column showing the average rating of each concept for the combined sample. The combined averages are weighted, so the two larger samples have a slightly greater effect on the results. The concepts are listed in decreasing order, based on combined rating.

Table 1. Top 20 concepts for SE-II students.

SE Concept	School-1 N = 11	School-2 N = 16	School-3 N = 16	Combined Rating
implementation	9.39	9.09	8.66	9.01
team	8.90	7.80	9.02	8.53
development	8.41	8.37	8.30	8.35
design	9.39	8.08	7.81	8.32
quality	8.49	7.15	8.90	8.14
integration	7.11	8.23	8.54	8.06
analysis	7.19	8.81	7.81	8.02
test	7.28	8.37	8.18	8.02
architecture	8.33	7.87	7.93	8.01
requirement	7.36	8.52	7.93	8.00
schedule	7.44	7.72	8.54	7.95
organization	8.41	8.01	7.57	7.95
software	6.06	8.08	9.02	7.92
algorithm	7.93	5.99	9.63	7.84
interface	9.06	7.80	6.96	7.81
customer	8.17	7.51	7.81	7.79
specification	6.38	8.23	8.18	7.74
class	7.76	8.08	7.33	7.72
project	7.93	7.22	7.93	7.67
performance	7.44	8.37	6.96	7.61

A visual inspection of the three schools in Table 1 reveals modest rating similarities for the concepts. In this table, the 10 highest rated concepts, all with combined ratings above 8.00, are *implementation*, *team*, *development*, *design*, *quality*, *integration*, *analysis*, *test*, *architecture*, and *requirement*. The above list is dominated by core concepts that appear in the first three SE lifecycle phases, with quality applying to all phases. The breadth of this list might reflect that students had been working on team projects throughout the semester. All of the concepts in Table 1 have combined ratings above 7.60. The remaining 44 concepts having combined ratings below 7.60 are not shown in this table.

Another way to view these results is with an ordered list of the 10 highest-rated concepts for each school. These three lists are presented in Table 2. Only two concepts--*implementation* and *development*--are included in the Top-10 lists for all three schools. The concepts *team*, *quality*, *test*, *integration*, and *specification* are listed for two of the schools. We note that *test* and *specification* tied for 10-th place in the School-3 ratings, giving that school a Top-11 list. The remaining concepts in Table 2 appear only once.

We can gather the five top-rated words at each school into brief descriptions of how the SE-II courses differ:

School-1: *Design*, *implementation*, *data*, *interface*, and *maintenance* are most important.

School-2: *Implementation*, *analysis*, *information*, *database*, and *requirement* are rated highest.

School-3: *Algorithm*, *software*, *team*, *quality*, and *implementation* are emphasized.

The students at all three schools agreed on the importance of *implementation*, especially while they were the final stages of completing their semester-long projects.

Table 2. Top 10 SE-II concepts by school.

Rank	School-1		School-2		School-3	
1	design	9.39	implementation	9.09	algorithm	9.63
2	implementation	9.39	analysis	8.81	software	9.02
3	data	9.14	information	8.59	team	9.02
4	interface	9.06	database	8.52	quality	8.90
5	maintenance	9.06	requirement	8.52	implementation	8.66
6	team	8.90	development	8.37	integration	8.54
7	quality	8.49	performance	8.37	schedule	8.54
8	development	8.41	test	8.37	development	8.30
9	organization	8.41	integration	8.23	prototype	8.30
10	architecture	8.33	specification	8.23	test/specification	8.18

Among the bottom 44 concepts (not in Table 1), five received combined ratings below 6.00: *language* (5.92), *domain* (5.92), *state* (5.65), *pattern* (5.60), and *formal* (4.81). There are several possible reasons why a concept received a low rating.

Some low-rated concepts apply primarily to early stages in the software development life cycle, such as *incremental* (6.48), *tool* (6.30), *diagram* (6.27), and *problem* (6.00). These concepts presumably would receive more emphasis in a SE-I course. Some concepts appear late in the life cycle, such as *deployment* (7.14), *maintenance* (7.12), *validation* (6.93), and *verification* (6.74), so they might receive delayed emphasis in a SE-II course.

Other concepts relate to a specific technology, so they are less likely to receive sustained focus throughout a semester. This includes concepts such as *use case* (7.15), UML (6.81), *pattern* (5.60), and *formal* (4.81). And, as mentioned earlier, some concepts are fairly general rather than SE-specific, such as *document* (6.85), *change* (6.58), *activity* (6.38), *discipline* (6.01), *language* (5.92), and *state* (5.65). This might have affected the ratings of these concepts.

For most of the 64 concepts, the mean ratings for the three schools are moderately consistent. The correlation coefficients between pairs of schools range from 0.361 (School-1 vs. School-3) to 0.431 (School-1 vs. School-2). For School-2 vs. School-3, the correlation is 0.387. These values suggest a small positive relationship between the concept ratings for the separate samples. The fact that the correlations are not larger indicates that notable differences in perceptions exist between the three SE-II courses. We examine sources of this variation in the next section.

Ratings Variation

We collected concept ratings from students in second-semester SE courses at three schools. The previous section focused on ratings differences between SE concepts, especially with respect to concepts that are considered most important by students. In this section, we describe how ratings vary for each concept.

Within-School Variation

The variability in ratings for each SE concept can be divided into two sources: within-schools and between-schools. We are primarily interested in between-school variation, which better reflects which concepts are emphasized by instructors in their courses. We examine within-school variation as a reference point for evaluating differences between courses, as well as to judge the level of agreement among students in their concept ratings.

For each of the 64 SE concepts, we calculated the *standard deviation* for student ratings within each course. Rather than present individual values of these statistics, we summarize the distribution of variation by school in Table 3.

Table 3. Summary of within-school ratings variation for individual concepts at each school.

Statistic	School-1	School-2	School-3
Min Std Dev	0.94	1.01	0.51
Max Std Dev	2.94	3.37	2.36
Avg Std Dev	2.01	2.10	1.21

The 192 (= 64*3) standard deviations ranged from a low of 0.51 (School-3) to a high of 3.37 (School-2). The average standard deviation value is near 2.0 at School-1 and School-2, but is much smaller for School-3. A "typical" standard deviation of 2.0 represents a relatively large amount of variation for a 10-point scale, considering that most scores fall within two standard deviations (plus or minus 4.0) from the mean.

The above table describes how SE-II students' ratings varied across all concepts within each school. We also wanted to determine which concepts were rated most consistently by students within the schools. To do this, we calculated a pooled within-school standard deviation for each concept. The concepts having pooled standard deviations below 1.50, along with their combined ratings, are listed in Table 4.

Table 4. Concepts having the *smallest* within-school ratings variation.

Topic/Concept	Pooled Std Dev	Combined Rating
implementation	0.95	9.01
team	1.32	8.53
development	1.34	8.35
specification	1.36	7.74
component	1.37	6.89
maintenance	1.38	7.12
software	1.42	7.92
function	1.47	7.55
application	1.51	7.24
interface	1.53	7.81

A relatively small standard deviation indicates that students within each school gave similar ratings for a concept. The concept having the smallest within-school variation is *implementation*, with a pooled standard deviation of 0.95. Other concepts with low variation are *team*, *development*, *specification*, *component*, and *maintenance*.

Consistent ratings are not equivalent to high ratings, although *implementation* is the highest-rated concept and has the smallest pooled standard deviation. However, 9 of the 10 concepts in Table 4 have average ratings above 7.0. Only *component* has a rating below 7.0.

The concepts having the *largest* pooled within-school standard deviations are listed in Table 5. A large pooled standard deviation suggests a lack of agreement among students on the importance of a concept. The concept having the largest within-school variation is *problem*, with a pooled standard deviation of 2.52. Other concepts with high variation are *formal*, *algorithm*, and *engineering*.

Table 5. Concepts having the *largest* within-school ratings variation.

Topic/Concept	Pooled Std Dev	Combined Rating
diagram	2.16	6.27
state	2.17	5.65
change	2.21	6.58
database	2.21	7.37
document	2.22	6.85
tool	2.23	6.30
engineering	2.37	7.25
algorithm	2.42	7.84
formal	2.42	4.81
problem	2.52	6.00

Only three of the Table 5 concepts--*algorithm*, *database*, and *engineering*--have ratings above 7.00. The other seven concepts have ratings below 7.00, and *formal* has the lowest rating of all concepts. For all 64 concepts, the correlation between combined ratings and pooled within-school variation is -0.645. This negative relationship suggests that student ratings are more consistent for the higher-rated concepts.

Between-School Variation

We now summarize the variation in ratings between schools in terms of patterns for concept means. Table 6 lists the SE concepts for which the between-school ratings showed the largest differences. We performed a one-way Analysis-of-Variance (ANOVA) for each concept, calculating the Between Mean-Square (BMS), Within Mean-Square (WMS), and F-ratio. Degrees of freedom for the F-statistic are 2 for BMS and 40 for WMS.

Table 6. ANOVA for concept rating differences between schools.

SE Concept	School-1 N = 11	School-2 N = 16	School-3 N = 16	Between Mean Sq	Within Mean Sq	F-ratio (p<.01)
maintenance	9.06	7.15	5.75	35.72	1.89	18.88
software	6.06	8.08	9.02	29.01	2.02	14.34
algorithm	7.93	5.99	9.63	52.95	5.86	9.04
specification	6.38	8.23	8.18	13.54	1.85	7.33
data	9.14	7.15	6.60	22.27	3.48	6.39
interface	9.06	7.80	6.96	14.34	2.34	6.13
cost	7.28	8.16	5.63	26.14	4.48	5.84
user	8.09	8.01	6.11	18.69	3.29	5.68

Our methodology does not justify the usual ANOVA probability model for testing differences between group means. Our samples at each school were not random (as in survey research), and we did not randomly assign students to the three groups (as in experimental design). Nevertheless, the concepts listed in Table 6 exhibited the largest sample differences as measured by the F-statistic. If the ANOVA model were appropriate, the significance level for each of the above concepts would be less than 0.01 (for individual tests).

The three concepts with the largest F-statistic values are *maintenance*, *software*, and *algorithm*. For these concepts, the range between highest and lowest sample means is 2.96 or greater. This suggests that the perceived importance of these concepts varies widely at the three schools, perhaps due to differences in course content or instructor emphasis.

When a large variation is obtained from three values, several patterns are possible:

1. One value can be much *higher* than the other two. For example, *data* (9.14) at School-1.
2. One value can be much *lower* than the other two. For example, *software* (6.06) and *specification* (6.38) at School-1; *cost* (5.63) and *user* (6.11) at School-3.
3. The values can be evenly spread, with the middle value spaced about equally between the high and low values. For example, *maintenance* (5.75 < 7.15 < 9.06), *algorithm* (5.99 < 7.93 < 9.63), *interface* (6.96 < 7.80 < 9.06).

We can look *vertically* at the concept ratings in Table 6 to view the distinct ratings patterns for each school. From this perspective, School-1 ratings are high for *data*, *maintenance*, and *interface*, but low for *software* and *specification*. School-2 ratings are low for *algorithm*. School-3 is high for *algorithm* and *software*, but low for *cost*, *maintenance*, and *user*.

SE-II vs. SE-I Student Ratings

The previous sections of this paper have presented an analysis of concept ratings collected from second-semester Software Engineering (SE-II) students. In an earlier study (McMaster, et al, 2011), we used the same questionnaire to obtain concept ratings from first-semester Software Engineering (SE-I) students. In this section, we compare the results from the two studies to determine how concept ratings change over a two-semester course sequence.

The concept ratings from the SE-I students were drawn from three schools, only one of which was part of our SE-II study. No students appeared in both studies. The ratings for the 64 SE concepts in the SE-I data were standardized in the same manner as the current research. That is, the

concept ratings for each school were rescaled to a mean of 7.20 and a standard deviation of 1.00. This consistent rescaling allows a meaningful comparison of the SE-I and SE-II data sets.

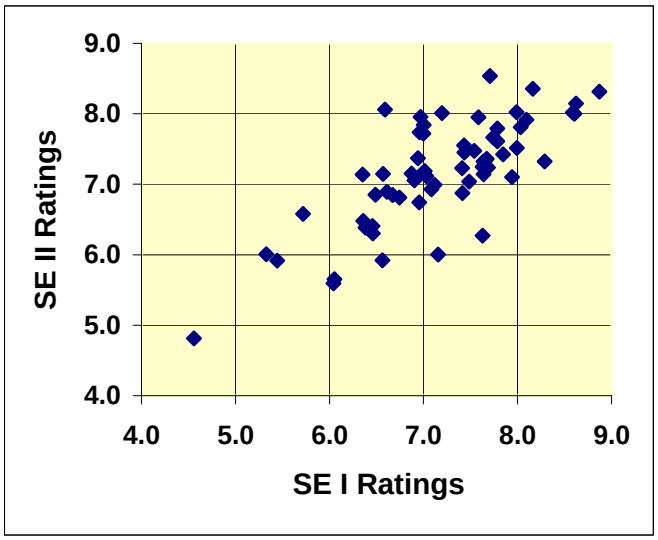

Figure 1: Concept ratings for SE-I vs. SE-II students.

The pairs of SE-I and SE-II combined ratings for the 64 concepts are displayed graphically as a scatter diagram in Figure 1. A strong positive relationship between the two sets of ratings is apparent. The correlation coefficient between SE-I ratings and SE-II ratings is 0.770. Thus, there is a substantial amount of agreement in SE combined ratings over the two semesters.

We wanted to learn how student perceptions about SE concepts changed over two semesters. In particular, we were interested in which concepts showed the largest changes, both increases and decreases. Table 7 lists the 12 concepts having a rating *increase* from SE-I to SE-II of 0.50 or greater (on the standardized scale).

Table 7. Largest concept rating *increases* from SE-I to SE-II.

SE Concept	SE-I Rating	SE-II Rating	Change SE-II - SE-I
integration	6.59	8.06	1.47
schedule	6.97	7.95	0.99
change	5.72	6.58	0.86
algorithm	7.00	7.84	0.84
team	7.71	8.53	0.83
architecture	7.19	8.01	0.82
framework	6.35	7.14	0.79
specification	6.96	7.74	0.78
class	7.00	7.72	0.72
implementation	8.32	9.01	0.69
discipline	5.33	6.01	0.68
deployment	6.57	7.14	0.58

The concept that showed the largest ratings increase is *integration* (+1.47). Other concepts with large increases include *schedule* (+0.99), *change* (+0.86), *algorithm* (+0.84), *team* (+0.83), and *architecture* (+0.82). Most of these concepts are clearly relevant for SE-II students who have been working as teams on a semester-long project. Interestingly, four mostly SE-I concepts (*architecture*, *framework*, *specification*, and *class*) went up in SE-II ratings. One might optimistically think that the students realized the importance of these concepts as they moved further along in their development efforts.

Regarding concept ratings that dropped, Table 8 lists the 10 concepts having a rating *decrease* from SE-I to SE-II of 0.50 or greater (in magnitude).

Table 8. Largest concept rating *decreases* from SE-I to SE-II.

SE Concept	SE-I Rating	SE-II Rating	Change SE-II - SE-I
diagram	7.63	6.27	-1.36
problem	7.15	6.00	-1.15
user	8.29	7.33	-0.96
solution	7.94	7.10	-0.84
language	6.56	5.92	-0.64
requirement	8.60	8.00	-0.60
test	8.59	8.02	-0.57
design	8.87	8.32	-0.55
code	7.41	6.87	-0.54
product	7.64	7.14	-0.50

Two of these concepts have ratings decreases of magnitude 1.0 or greater: *diagram* (-1.36) and *problem* (-1.15). Two other concepts with large ratings decreases include *user* (-0.96), and *solution* (-0.84).

One could argue that concepts such as *problem*, *requirement*, *design*, and *diagram* relate more to analysis and design phase activities. As a result they might receive less emphasis at the end of the SE-II course, when students are under pressure to deliver working software. However, *code* and *test* should be of greater importance in the SE-II course. Another note of interest is that SE-II students see *users* as less important. This might be due to a lack of "real users" in their SE-II projects. On the other hand, real world projects often fail to keep users actively involved in the later stages of the development process.

A closer look at Table 7 and Table 8 reveals an interesting pattern. For the 12 concepts with the largest rating *increases*, 10 have a SE-I rating below the mean of 7.20. On the other hand, only 2 of the 10 concepts with the largest rating *decreases* have an SE-I rating below 7.20. A partial explanation for this pattern is that a concept with a high SE-I rating has less room for an increase, and more room for a decrease.

A more detailed description of this phenomenon is presented in Table 9. This table shows the number of concept rating increases and decreases for the 64 SE-I ratings values grouped into five intervals.

Table 9. Concept rating changes from SE-I to SE-II.

Change to SE-II	SE-I <6.50	6.50 - 6.99	7.00 - 7.49	7.50 - 7.99	SE-I >=8.00
Increase	7	10	8	4	2
Decrease	5	2	6	13	7
Avg Change	0.21	0.36	0.02	-0.26	-0.30

Note that higher SE-I ratings lead to more rating decreases. This pattern is less likely to apply to concepts having an initial low SE-I rating. It appears that concepts at extreme ends of the SE-I scale tend to move (regress) toward the center of the SE-II scale.

Summary and Conclusions

The primary purpose of this research was to identify concepts that are considered most important to students after a second-semester Software Engineering course. These concepts presumably would be used by students to construct mental frameworks for Software Engineering. A suitable framework can help SE students integrate course topics a meaningful way to promote learning and understanding.

In this study, we asked students in second-semester SE courses at three schools to rate the relative importance of 64 concepts. After standardizing the data at each school, we obtained relatively consistent SE concept ratings. The five top-rated concepts, based on averages across the three schools, are *implementation*, *team*, *development*, *design*, and *quality*. The first four concepts represent middle life-cycle activities, performed while working on team projects. *Quality* is important during all stages of software development. Concepts that apply primarily to early stages, involve a specific technology, or are not SE-specific tend to have lower ratings.

We calculated within-school variation to see how consistently the students rated the SE concepts. We also examined the variation between schools to measure the effect of course differences on the ratings. The top-rated concept *implementation* had the smallest variation within courses, indicating substantial agreement among the students. The concepts with the largest differences between schools were *maintenance*, *software*, and *algorithm*, suggesting varying emphasis on these topics at the three schools.

The concept ratings for second-semester SE students, when compared to ratings by first-semester students (obtained in an earlier study), showed a strong positive relationship. The concepts having the largest rating increases during the second semester were *integration* and *schedule*. The largest decreases were for *diagram*, *problem*, and *user*. Rating increases and decreases indicate changes in relevance as different life-cycle activities are performed during the two-course SE sequence. One overall change pattern was a tendency for ratings to regress toward the mean in the second semester. That is, concepts with high first-semester ratings often dropped, while low first-semester ratings tended to increase.

This study focused on student ratings for individual Software Engineering concepts. Subsequent research is planned to examine how students mentally assemble these concepts into effective frameworks.

What can Software Engineering instructors do with this research? They can compare the concept ratings reported in this paper with the concepts they feel are most relevant for their students. SE instructors can also reflect on how they integrate their preferred concepts into course frameworks.

Future Research

Future research includes a replication of the first-semester and second-semester studies with larger samples to confirm our preliminary findings. SE instructors will also be questioned to discover which concepts they believe are most important. We will can then better assess how closely student ratings match those of their instructors.

We especially want to study how students organize the individual SE concepts into meaningful frameworks. Which mental dimensions do they apply in linking and grouping the concepts? Are life-cycle phases or software architecture levels important components of their frameworks? What other relevant organizing criteria are used? Obtaining visual representations of student frameworks would be of particular interest.

References

Bain, K. (2004). *What the best college teachers do*. Harvard University Press.

Beck, K. (2002). *Test-driven development: By example*. Addison-Wesley.

Beck, K., & Andres, C. (2004). *Extreme programming explained: Embrace change* (2nd ed). Addison-Wesley.

Cockburn, A. (2006). *Agile software development: The cooperative game* (2nd ed). Addison-Wesley.

Connolly, T., & Begg, C. (2009). *Database systems: A practical approach to design, implementation and management* (5th ed). Addison-Wesley.

Davis, P., & Hersh, R. (1981). *The mathematical experience*. Birkhauser.

Donald, J. (2002). *Learning to think*. Jossey-Bass.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Jacobson, I, Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Addison-Wesley.

Lafore, R. (2001). *Object-oriented programming in C++* (4th ed). Sams.

Lafore, R. (2002). *Data structures and algorithms in Java* (2nd ed). Sams.

McConnell, S. (2004). *Code complete: A practical handbook of software construction* (2nd ed). Microsoft Press.

McMaster, K., Rague, B, Hadfield, S., & Anderson, N. (2008). Three software development gestalts. In *The Proceedings of the Information Systems Education Conference 2008*, v 25 (Phoenix).

McMaster, K., Hadfield, S., Wolthuis, S., & Sambasivam, S. (2011). Software engineering frameworks: Textbooks vs. student perceptions. In *The Proceedings of ISECON 2011*, v28 (Wilmington, NC).

Pressman, R. (2009). *Software engineering: A practitioner's approach* (7th ed). McGraw-Hill.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed). Prentice Hall.

Silberschatz , A., Galvin, T., & Gagne, G. (2008). *Operating system concepts*. Wiley.

Sommerville, I. (2004). *Software engineering* (7th ed). Addison-Wesley.

Tanenbaum, A. & Wetherall, D. (2011). *Computer networks* (5th ed). Prentice Hall.

Biographies

Dr. Kirby McMaster recently retired from the Computer Science Department at Weber State University. To remain active, he is currently a visiting professor in Computer Science and Information Systems at Fort Lewis College in Durango, Colorado. His primary research interests are in database systems, software engineering, and frameworks for Computer Science and Mathematics.

Dr. Samuel Sambasivam is Chairman and Professor of the Computer Science Department at Azusa Pacific University. His research interests include optimization methods, expert systems, client/server applications, database systems, and genetic algorithms. He served as a Distinguished Visiting Professor of Computer Science at the United States Air Force Academy in Colorado Springs, Colorado for a year. He has conducted extensive research, written for publications, and delivered presentations in Computer Science, data structures, and Mathematics. He is a voting member of the ACM and is a member of the Institute of Electrical and Electronics Engineers (IEEE).

Dr. Steve Hadfield is an Associate Professor of Computer Science at the United States Air Force Academy in Colorado Springs, Colorado. He has taught at the Academy for nineteen years in both the departments of Computer Science and Mathematical Sciences and currently serves as the curriculum chair for Computer Science. He earned his Ph.D. in Computer Science from the University of Florida in 1994. His research interests include software engineering, software assurance, and Computer Science education.

Stuart L. Wolthuis is Assistant Professor in the Computer & Information Sciences Department at Brigham Young University--Hawaii. His teaching focus includes software engineering, HCI and information assurance. He brings almost 24 years of service in the USAF to the classroom with real world experiences as a program manager and software engineer. When not enjoying Hawaii's great outdoors, his research interests include melding together information systems and marine biology. His current project, Ocean View, will link land-locked educators and students to live underwater ocean views via an educational website.