Software Engineering Frameworks: Textbooks vs. Student Perceptions

Kirby McMaster
kmcmaster@weber.edu
CSIS Dept, Fort Lewis College
Durango, CO 81301, USA

Steven Hadfield
steven.hadfield@usafa.edu
CS Dept, U.S. Air Force Academy
Colorado Springs, CO 80840, USA

Stuart Wolthuis
stuartlw@byuh.edu
CIS Dept, Brigham Young University-Hawaii
Laie, HI 96762, USA

Samuel Sambasivam
ssambasivam@apu.edu
CS Dept, Azusa Pacific University
Azusa, CA 91702, USA

Abstract

This research examines the frameworks used by Computer Science and Information Systems students at the conclusion of their first semester of study of Software Engineering topics. A questionnaire listing 64 Software Engineering concepts was given to students upon completion of their first Software Engineering course. This survey was given to samples of students at three universities. 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 this survey for the three student samples. We then compare the student ratings with word frequencies exhibited by authors of Software Engineering textbooks. In this way, we show how student frameworks relate to frameworks presented by Software Engineering authors.

Keywords: Software Engineering, framework, gestalt, schema, concept, rating.

1. INTRODUCTION

Learning is more effective if course topics and concepts are organized within an overall mental *framework*, or *gestalt*. By gestalt, we mean "a configuration or pattern of elements so unified as a whole that it cannot be described merely as a sum of its parts" (www.thefreedictionary.com). 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*, *paradigm*, and *mental model*.

According to Donald (2002), a course needs a schema to enable and 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 order and how closely concepts are linked and the character of the linkage.

Bain (2004) describes why instructors should provide frameworks for courses, rather than rely on students to form their own.

> 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 are common in virtually all Computer Science and Information Systems (CSIS) courses. Often, primary concepts are organized 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 a blend of the OSI Model and the Internet Protocol Suite (Peterson & Davie, 2011). Operating Systems courses include topics from the hardware, kernel, system services, and application layers (Silberschatz, Galvin, & Gagne, 2011). Computer Hardware has layers from simple digital logic up to VLSI circuits and functional components (Patterson & Hennessy, 2008). Database courses insert a DBMS software layer between application programs and operating system files (Connolly & Begg, 2009).

Not all computing frameworks are layered. The usual framework for Object-Oriented Programming (Lafore, 2001) includes sets of interrelated classes, arranged according to established design patterns (Gamma, Helm, Johnson, & Vlissides, 1994). Data Structures course topics are divided into data structure and algorithm categories, such as stacks, queues, linked lists, searching, and sorting (Drozdek, 2008). Artificial Intelligence has utilized a variety of frameworks over the years for 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? Pressman (2009) and Sommerville (2010) offer common variations (such as "waterfall" and iterative) of the classical life cycle approach to software development. Schach (2010) focuses more on object-oriented methods. Cohn (2009) encourages successful management practices to integrate agile development with Scrum.

In our previous research (McMaster, Rague, 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 words used most often in a book indicate the gestalt of the author. From each sample of books, we constructed a framework (or scale) as an ordered list of most frequent words.

In this research, we sought to determine what mental frameworks *students* had developed at the completion of their first SE course. We examined whether their frameworks were consistent across courses taught by different instructors at different schools. We also compared the student frameworks with those of authors of commonly used SE textbooks.

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 rating pattern variations for courses taught by different instructors. Finally, we compare student ratings with word frequencies in SE textbooks.

2. METHODOLOGY

In this section, we describe the methodology used in our study. A questionnaire listing 64 Software Engineering concepts (see Appendix B) was given to CSIS students upon completion of their first 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 ranked high on a Software Engineering *gestalt* scale that we previously developed from frequently used words in SE books. We supplemented this word list with topics we felt were important, utilizing

input from other instructors that teach SE courses. 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 list was compiled, it was randomized so that there would be no implied significance to the order in which the concepts were presented to students.

The SE concept list was included in a survey given to samples of students at three schools. School-1 consisted of 9 SE students at a state university, School-2 consisted of 27 SE students at a national university, and School-3 consisted of 19 SE students at a private university. Almost all students were juniors or seniors, and had completed courses in programming and data structures. Some students had also taken a database course. The course sections had different instructors and textbooks, but each sample of students received a fairly traditional first semester SE course, with an emphasis on systems analysis and design.

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 within each sample. We calculated *trimmed means*, removing approximately the top and bottom 11% (1/9 or 2/19 or 3/27) of the individual ratings, so that extreme responses would not unduly influence the concept ratings.

We found that the trimmed means for the 64 concepts differed in a biased way between the three schools. To make the data for the samples comparable, we *standardized* (rescaled) the concept means within each school, so that the three sets of 64 scores had the same average (7.20) and standard deviation (1.00). This rescaling kept the combined mean at 7.20, but changed the standard deviations slightly. Note that we did not rescale individual student ratings. We rescaled the trimmed means in a way that preserved the ordering of concepts within each school. We could have achieved a similar result by converting the trimmed 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 types of data to analyze and compare: student ratings for the three schools, and textbook word frequencies from our prior research. We first examine the concept ratings for the three schools, both separately and combined. Next, we look at the ratings variation for each concept within schools and between schools. Then we compare the combined student ratings with word frequencies in SE textbooks.

3. CONCEPT RATINGS

In this section, we analyze the concept ratings for the three student samples. Table 1 presents the 32 top-rated Software Engineering concepts (out of 64), along with the rescaled trimmed means for School-1, School-2, and School-3.

Table 1. Top 32 concept ratings for schools.

SE Concept	School-1 N = 9	School-2 N = 27	School-3 N = 19	Combined Rating
design	8.71	9.19	8.71	8.87
quality	9.15	8.72	8.00	8.62
requirement	8.13	9.21	8.47	8.60
test	8.56	8.96	8.24	8.59
implementation	8.27	8.67	8.00	8.32
user	7.98	8.88	8.00	8.29
development	8.13	7.97	8.40	8.16
software	8.56	7.72	8.00	8.10
interface	8.42	8.30	7.38	8.03
information	7.98	7.76	8.24	7.99
analysis	7.83	7.35	8.79	7.99
solution	7.98	7.76	8.08	7.94
prototype	7.98	8.18	7.38	7.84
performance	7.83	7.68	7.85	7.79
customer	6.96	9.25	7.14	7.79
project	7.83	7.31	8.08	7.74
team	7.54	7.89	7.69	7.71
application	8.42	7.26	7.38	7.69
method	8.27	7.06	7.69	7.67
model	8.42	7.55	6.99	7.65
product	7.98	8.34	6.59	7.64
management	6.96	8.34	7.61	7.64
diagram	7.69	7.43	7.77	7.63
engineering	7.40	7.01	8.47	7.63
organization	7.54	8.38	6.83	7.59
program	7.83	7.10	7.69	7.54
system	7.40	6.97	8.08	7.48
data	7.98	6.56	7.77	7.44
function	7.83	6.85	7.61	7.43
code	7.69	7.10	7.46	7.41
process	7.40	6.52	8.32	7.41
architecture	6.96	7.72	6.91	7.20

We include a column showing the average rating of each concept for the combined sample.

The combined ratings are unweighted to prevent the larger School-2 sample from dominating the results. The concepts are listed in decreasing order, based on average rating.

A quick visual inspection of the three schools in Table 1 reveals substantial rating similarities for the concepts. In this table, the top five rated concepts, all with combined ratings above 8.30, are *design*, *quality*, *requirement*, *test*, and *implementation* (four life cycle phase descriptors, plus an umbrella goal). These five words received a mean rating greater than 8.00 within each school. Close behind are the ratings for *user*, *development*, and *software*.

The other 24 concepts in Table 1 have average ratings at or above the mean (7.20) for all 64 concepts. The 32 concepts having average ratings below 7.20 are presented in Appendix A.

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.

Table 2. Top 10 concepts by school.

Rank	School-1	School-2	School-3
1	quality	customer	analysis
2	design	requirement	design
3	test	design	requirement
4	software	test	engineering
5	interface	user	database
6	application	quality	development
7	model	implementation	process
8	implementation	organization	test
9	method	product	information
10	algorithm	management	solution

The concepts *design* and *test* are included in the Top-10 lists for all three schools. *Quality*, *requirement*, and *implementation* are listed for two of the schools. The remaining 18 concepts in Table 2 appear only once.

We can gather the top-rated words and several of the 18 unique words from Table 2 into brief conjectural descriptions of how the three SE courses differ.

> School-1: Quality is #1. The methodology uses models and algorithms to build applications.
>
> School-2: The customer is #1. Organization and management are necessary to create a product that will satisfy users. (Students in this course worked on real-world projects.)
>
> School-3: Analysis is #1. Databases are developed to provide information and solutions. (This was a CIS course.)

Among the bottom 32 concepts, four received ratings below 6.00: *change* (5.72), *domain* (5.44), *discipline* (5.33), and *formal* (4.56). There are several possible reasons why a concept received a below-average rating. Some concepts apply to later stages in the software development life cycle, such as *construction* (7.01), *integration* (6.59), *deployment* (6.57), *validation* (7.08), *verification* (6.95), and *maintenance* (7.03). These concepts presumably would receive more emphasis in a second-semester SE course.

Other concepts relate to a narrow range of the life cycle or to a specific technology, so they are less likely to receive continual emphasis during a semester. This includes concepts such as *agile* (7.01), *formal* (4.56), *incremental* (6.36), *pattern* (6.04), *UML* (6.74), and *use case* (6.87). And, as mentioned earlier, some concepts are fairly general rather than SE-specific, such as *activity* (6.38), *change* (5.72), *discipline* (5.33), *document* (6.67), *language* (6.56), and *state* (6.05).

Over the 64 concepts, the school ratings were reasonably consistent. The correlation coefficients between pairs of schools are summarized in Table 3. The correlations range from 0.480 (School-2 vs. School-3) to 0.576 (School-1 vs. School-3). These values suggest a moderate positive relationship between the concept ratings for the separate samples. The fact that the correlations are not larger suggests that some notable differences in ratings exist between the three schools. We examine sources of this variation in the next section.

Table 3. Rating correlations between schools.

Correlations	School-1	School-2	School-3
School-1	1.000	0.568	0.576
School-2	0.568	1.000	0.480
School-3	0.576	0.480	1.000

4. RATINGS VARIATION

We collected concept ratings from students in 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 one concept at a time.

4.1 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 should better reflect the concepts that instructors emphasize in their courses. We computed within-school variation for each concept to provide a reference point for evaluating course differences.

For each of the 64 SE concepts, we calculated the (untrimmed) *standard deviation* for student ratings within each course. Rather than present individual values of these statistics, we summarize the pattern of variation by school in Table 4.

Table 4. Between-student ratings variation for concepts at each school.

Statistic	School-1	School-2	School-3
Min Std Dev	0.88	1.63	0.93
Max Std Dev	3.22	2.91	3.04
Avg Std Dev	1.86	2.25	1.92

The 192 standard deviations ranged from a low of 0.88 (School-1) to a high of 3.22 (again School-1). The average standard deviation value was slightly below 2.0 at School-1 and School-3, but was over 2.0 at School-2. So a "typical" measure of student-to-student variability for a concept is about 2.0. This is a relatively large amount of variation, considering that a "well-behaved" distribution has about 95% of the scores within two standard deviations (+/- 4.0) from the mean. On a 10-point ratings scale, this would be an interval of width 8. Many ratings distributions tended to be skewed, so the 95% rule is less relevant in these cases.

We also calculated the *range* of the ratings scores for each concept within each school. School-1 had an average range of 5.31, while the average range for School-3 was 6.39. The average range for School-2 was somewhat larger (8.05), which is consistent with the larger standard deviation for this school.

4.2 Between-School Variation

We now summarize the variation in ratings between schools in terms of patterns for concept means. For (untrimmed) means of random samples of size N, the variance of the means will vary inversely with the sample size N. So for a sample of size N = 9 (School-1), the standard deviation of the sample means would be approximately 2.0/3 = 0.67, assuming that the individual scores have a standard deviation of 2.0. For larger sample sizes, the means would vary less.

Two features of our methodology limit the strict validity of the above probability model for this study: (1) our samples were not random, and (2) we calculated trimmed means for each concept. The large within-school variation described earlier was part of the motivation for using trimmed means. Still, the above discussion provides a context for the way we interpreted differences in means between schools.

Table 5 lists the SE concepts for which the between-school ratings showed the largest differences.

Table 5. Concept rating mean differences. (highest H or lowest L for concept)

SE Concept	School-1 N = 9	School-2 N = 27	School-3 N = 19	Range= Hi - Lo
database	6.37	5.98	8.47H	2.50
algorithm	8.27H	6.85	5.89	2.38
CASE	5.64	5.73	8.00H	2.36
customer	6.96	9.25H	7.14	2.30
cost	6.08	8.05	7.22	1.97
formal	3.89	5.85H	3.93	1.96
UML	6.37	6.01	7.85H	1.84
document	5.50L	7.22	7.30	1.80
process	7.40	6.52	8.32	1.80
product	7.98	8.34	6.59L	1.75

For each concept, we calculated the standard deviation and the range of the three school means. The *ranges* are shown in the table, with concepts listed in decreasing range order. We only include concepts with a range above 1.70, which is much larger than the random variation model for means described above. Four of the concepts--*database*, *algorithm*, *CASE*, and *customer*--have ranges larger than 2.0. This suggests that the SE instructors in our study vary noticeably in how they present these topics.

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

1. One value can be much *higher* than the other two.
2. One value can be much *lower* than the other two.
3. The values can be evenly spread, with the middle value spaced about equally between the high and low values.

Looking *horizontally* at the mean ratings for each concept, we have marked a rating with an H if it is much higher than the others, and with an L if it is much lower. For example, the *database* rating for School-3 is 8.47H, and the *document* rating for School-1 is 5.50L. Note that the low *formal* rating of 5.85 for School-2 is marked with an H, as the other two schools have even lower ratings for this concept.

We can also look *vertically* at the concept ratings in Table 5 to view the distinct ratings patterns for each school. Concepts may not have been rated as important, but they were rated much higher or lower by one of the schools. From this perspective, School-1 is high for *algorithm* and low for *document*. School-2 is high for *customer* and high (less low) for *formal*. School-3 is high for *database*, *CASE*, and *UML* and low for *product*.

4.3 Ratings Profiles

In Table 5, we listed SE concepts having the largest differences in mean ratings between schools. Now we provide a visual representation of the top-24 (of 32) concepts from Table 1, where concepts are ordered by decreasing average rating. Figure 1 provides a graph of the concept ratings for each school, with a separate "line" for each school.

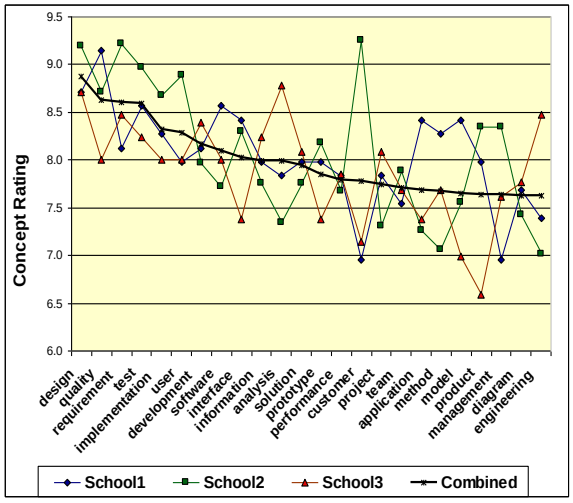

Figure 1: Top 24 concepts--profiles of 3 schools.

This figure presents the ratings pattern for each school as a *profile*. The successive differences between concept means for schools gives the illusion of random variation in most cases. Two exceptions are the concepts *customer* and *product*, where the ratings vary most widely. These concepts are included among the Table 5 concepts with large mean ratings differences.

5. STUDENTS VS. TEXTBOOKS

We now compare average concept ratings by students with two measures of word usage in Software Engineering textbooks. We exclude *use case* from this analysis, because this concept involves two words. Our textbook word counts are for single words only. For the remaining 63 concepts, we recorded how *often* and how *consistently* these words appear in a (nonrandom) sample of 36 SE books. Table 6 shows the concept ratings, word frequencies, and book counts for the top 32 student-rated concepts. Textbook results for the bottom 32 concepts are included in Appendix A.

Table 6. Top 32 concept ratings--students vs. textbooks.

SE Concept	Concept Rating	Textbook StdFreq	Books
design	8.87	158.3	**35**
quality	8.62	108.7	17
requirement	8.60	183.2	29
test (testing)	8.59	221.0	24
implementation	8.32	90.0	13
user	8.29	131.6	26
development	8.16	208.0	**36**
software	8.10	**377.8**	**36**
interface	8.03	103.5	18
information	7.99	109.4	27
analysis	7.99	92.4	26
solution	7.94	112.5	6
prototype	7.84	106.2	2
performance	7.79	61.5	7
customer	7.79	126.6	17
project	7.74	229.8	30
team	7.71	154.2	17
application	7.69	108.1	26
method	7.67	120.1	27
model	7.65	201.3	33
product	7.64	165.9	26
management	7.64	99.0	25
diagram	7.63	123.1	15
engineering	7.63	136.8	19
organization	7.59	108.3	16
program	7.54	145.6	26
system	7.48	**358.1**	**35**
data	7.44	154.9	32
function	7.43	93.1	21
code	7.41	118.8	27
process	7.41	**259.1**	**36**
architecture	7.20	117.3	13

To measure consistency of word use, the Books column gives the number of books (out of 36) that include the word in its concordance. The concordance is a list of the 100 most frequently used words in a book (excluding common English words). In Table 6, the words *software*, *development*, and *process* are listed in all 36 concordances; *design* and *system* are in 35 concordances.

To measure how often a word appears in a book, we rescaled each word frequency so that the average word frequency within a concordance was 100. This compensates for books having different total word counts. The standardized frequency (StdFreq) for a word is the average rescaled frequency across all books that include the word in its concordance. Based on this measure, the three most frequent words are *software* (StdFreq = 377.8), *system* (StdFreq = 358.1), and *process* (StdFreq = 259.1).

In Table 6, the word *model* has a StdFreq value of 201.3 for the 33 books that include this word in their concordances. The interpretation of this measure is that *model* occurs about twice as often as an average concordance word in SE books that include *model* in their concordances.

The below-average rated concepts from our questionnaire are not shown in Table 6. Three of these words--*discipline*, *incremental*, and *validation*--are not in the concordance of any of our sample books. This does not imply that these words do not appear in the books. It just means that they do not occur frequently enough to be listed in the concordances.

Of current interest, the word *agile* (not in Table 6) appears in the concordances of just two SE books. In contrast, the standardized frequency of *agile* is 194.4, suggesting that these two books utilize this word heavily.

The scatter diagram in Figure 2 displays the relationship between the combined concept ratings for the students vs. the standardized frequencies of these words in the SE textbooks.

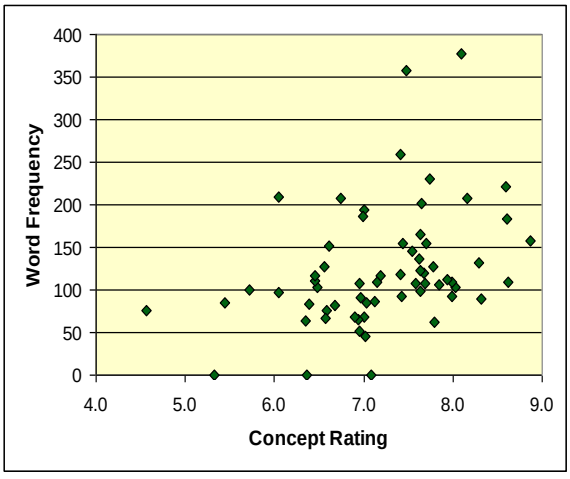

Figure 2: Concept rating vs. textbook word frequency.

Note the "diamonds" along the horizontal axis, representing the three books that were not listed in any concordance (and therefore received StdFreq values of 0.0)

In this graph, the words *software* and *system* appear as "outliers", in that the frequencies are noticeably higher for these words. One possible reason for the prevalence of these words is that they apply throughout the development cycle and are mentioned in multiple chapters in SE books. On the other hand, the highly rated word *quality* applies to every life cycle stage, but SE authors use this word less often.

The caution here is that word *frequency* does not necessarily imply *importance*. If we accept that the phrase "repetition brings conviction" applies to SE courses, perhaps we should emphasize important concepts such as *schedule* (StdFreq = 91.4, *cost* (StdFreq = 86.3), *maintenance* (StdFreq = 84.7), *document* (StdFreq = 81.8), and *performance* (StdFreq = 61.5) throughout the course, regardless of how sparingly these words appear in textbooks.

The correlation coefficient between combined concept ratings and textbook word frequencies is 0.373 (0.381 with the two high outliers removed), indicating a modest positive linear relationship. Not surprisingly, this is lower than the correlation coefficients for concept ratings between pairs of schools (which range from 0.480 to 0.576).

Thus, the students in this study agree more with each other on the relative importance of topics than they do with textbook authors, even though the students had different instructors and different textbooks. We are pleased that the correlation between concept ratings and textbook word frequencies is not negative. In the Internet era, many students do not read their textbooks consistently.

Figure 3 displays the relationship between combined concept ratings and the number of SE books containing concept words in their concordances. In this figure, no "outliers" are obvious, probably because the number of books is bounded by 36.

The correlation coefficient between concept ratings and number of textbooks is 0.415, which is slightly higher than the correlation between ratings and word frequencies. The diagram does illustrate how much "scatter" can be present in a relationship having a correlation of approximately 0.40.

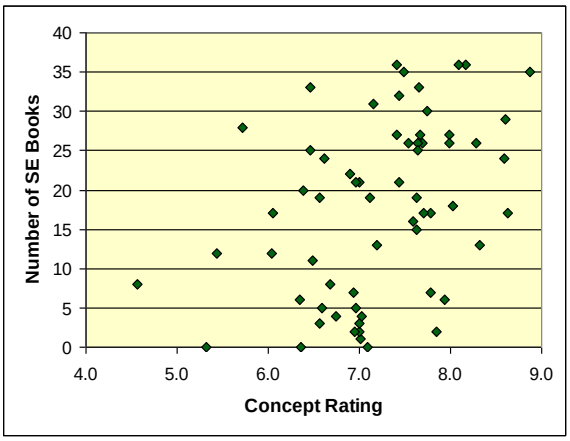

Figure 3: Concept rating vs. SE books.

To summarize, we found a modest positive relationship between student ratings of concepts and the two measures of word occurrence in textbooks. Most of the concepts with above-average student ratings appeared in the concordances of the majority of the SE books and had a standardized frequency above 100. From the textbook point of view, all three SE words that failed to appear in any concordances had below-average student ratings.

6. SUMMARY AND CONCLUSIONS

Constructing a framework for knowledge is essential for students in a Software Engineering course. A successful mental framework can help students organize course topics into a meaningful whole, which promotes learning.

In a previous study, we developed an authors' SE framework based on word frequencies in popular SE books. In this current research, we surveyed students at three schools on the relative importance of topics in an introductory SE course. We chose 64 concepts that students might use in constructing their own mental frameworks for SE. After standardizing the data from students at each school, we obtained relatively consistent concept ratings.

The five highest rated words were *design*, *quality*, *requirement*, *test*, and *implementation*, based on averages across the three schools. Concepts that apply to early states or multiple stages of the software development life cycle tended to have higher ratings. Concepts that arise late in the life cycle or involve a specific technology had lower ratings.

Within schools, variability of student ratings for concepts was quite large, with an average standard deviation of about 2.0 (for a 10-point scale). There was less ratings variation between schools, partly due to our calculating trimmed means for each concept. The largest between-school variation occurred for four concepts--*database*, *algorithm*, *CASE*, and *customer*.

Overall, the ratings profiles for the top-24 concepts were reasonably consistent for the schools, with two exceptions (*customer* and *product*). As faculty, we often agree on what is most important, but we have difficulty agreeing on what is less important. As a result, each instructor emphasizes certain extra things that make her/his course distinctive.

When student ratings for concepts were compared to frequent (concordance) words in a sample of 36 SE textbooks, only a moderate positive relationship was found. Highly rated concepts appeared more often in the sample books, but three lower-rated words were not in the concordances of any of the books.

Current Software Engineering instructors can benefit from comparing results on student ratings as summarized in this paper with their own perception of most important concepts. Where there are differences, consider how you highlight your favored SE concepts. In particular, how do emphasize important concepts that do not appear frequently in SE textbooks?

On a related note, are you certain that the frameworks of your students are consistent with the primary objectives of your SE course? Not all student learning comes from listening to lectures, reading textbooks, and doing homework assignments. You are encouraged to use the questionnaire in Appendix B to obtain feedback from your students.

6.1 Future Research

Future research will include a replication of this study with larger samples to verify our preliminary findings. With additional data, we can check how specific textbooks used in Software Engineering courses affect ratings of concepts. SE instructors could be surveyed in a similar manner to discover which concepts they feel are most important. We would then be able to assess how closely instructor ratings match those of their students.

We would also like to extend this research to examine how student frameworks evolve after taking additional SE courses, especially the SE II

course. We could study how students perceptions change as they gain more experience with the later stages of the software development life cycle.

The focus of this research has been on words that form frameworks for Software Engineering. Beyond a collection of words, a framework should provide a meaningful context that explains how the words fit together. A special challenge for future research is to examine various ways that SE words can be integrated into a unified Software Engineering framework.

7. REFERENCES

Bain, K. (2004). What the Best College Teachers Do. Harvard University Press, pp 26-27.

Cohn, Mike (2009). Succeeding with Agile: Software Development Using Scrum. Addison Wesley.

Connolly, T., and Begg, C. (2009). Database Systems: A Practical Approach to Design, Implementation and Management (5th ed). Addison Wesley.

Donald, J. (2002). Learning to Think. Jossey-Bass, p 15.

Drozdek, A. (2008). Data Structures and Algorithms in Java (3rd ed). Cengage Learning.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley.

Lafore, R. (2001). Object-Oriented Programming in C++ (4th ed). Sams.

McMaster, K., Rague, B., Hadfield, S., and Anderson, N. (2008), Three Software Development Gestalts. In *The Proceedings of the Information Systems Education Conference 2008*, v 25 (Phoenix).

Patterson, D., and Hennessy, J. (2008). Computer Organization and Design (4th ed). Morgan Kaufmann.

Peterson, L., and Davie, B. (2011). Computer Networks: A Systems Approach (5th ed). Morgan Kaufmann.

Pressman, R. (2009). Software Engineering: A Practitioner's Approach (7th ed). McGraw-Hill.

Russell, S., and Norvig, P. (2009). Artificial Intelligence: A Modern Approach (3rd ed). Prentice Hall.

Schach, S. (2010). Object-Oriented and Classical Software Engineering (8th ed). McGraw-Hill.

Silberschatz , A, Galvin, P., and Gagne, G. (2011). Operating System Concepts (8th ed). Wiley.

Sommerville, I. (2010). Software Engineering (9th ed). Addison Wesley.

APPENDIX A

Bottom 32 Concept Ratings--Students vs. Textbooks.

SE Concept	School-1 N = 9	School-2 N = 27	School-3 N = 19	Combined Rating	Textbook StdFreq	Books
problem	6.52	7.72	7.22	7.15	108.8	31
cost	6.08	8.05	7.22	7.12	86.3	19
validation	6.37	7.97	6.91	7.08	--	0
maintenance	6.96	7.14	6.99	7.03	84.7	4
construction	7.69	6.60	6.75	7.01	45.2	1
agile	7.98	6.60	6.44	7.01	194.4	2
algorithm	8.27	6.85	5.89	7.00	68.4	3
class	7.40	6.14	7.46	7.00	186.7	21
schedule	6.37	8.01	6.52	6.97	91.4	5
specification	6.52	6.97	7.38	6.96	107.8	21
verification	6.37	7.35	7.14	6.95	51.0	2
database	6.37	5.98	8.47	6.94	65.9	7
control	7.25	6.48	6.99	6.90	68.8	22
use (case)	6.52	6.72	7.38	6.87	--	--
UML	6.37	6.01	7.85	6.74	207.3	4
document	5.50	7.22	7.30	6.67	81.8	8
component	7.10	5.89	6.83	6.61	152.0	24
integration	6.52	6.97	6.28	6.59	75.9	5
deployment	7.25	6.72	5.73	6.57	67.3	3
language	6.52	6.10	7.06	6.56	127.9	19
module	6.23	6.56	6.67	6.49	103.2	11
tool	5.94	6.14	7.30	6.46	110.9	25
CASE	5.64	5.73	8.00	6.46	117.4	33
activity	6.96	5.52	6.67	6.38	83.5	20
incremental	5.94	7.01	6.12	6.36	--	0
framework	6.52	6.64	5.89	6.35	63.4	6
state	5.79	6.01	6.36	6.05	97.6	17
pattern	6.81	5.81	5.50	6.04	209.5	12
change	6.23	6.06	4.87	5.72	100.2	28
domain	5.35	5.40	5.58	5.44	84.3	12
discipline	5.94	5.56	4.48	5.33	--	0
formal	3.89	5.85	3.93	4.56	75.9	8

APPENDIX B

Software Engineering Topic Identification Name _____

For each topic/concept listed below, please rate on a scale from 1 to 10 the importance of the topic in this Software Engineering course. On this scale, 1 represents "least important" and 10 represents "most important".

	Topic/Concept
_____	implementation
_____	algorithm
_____	model
_____	test
_____	activity
_____	domain
_____	deployment
_____	formal
_____	problem
_____	design
_____	interface
_____	data
_____	maintenance
_____	diagram
_____	discipline
_____	change
_____	customer
_____	cost
_____	agile
_____	schedule
_____	program
_____	UML
_____	CASE
_____	language
_____	code
_____	project
_____	engineering
_____	tool
_____	use case
_____	integration
_____	verification
_____	software

	Topic/Concept
_____	product
_____	construction
_____	performance
_____	pattern
_____	framework
_____	state
_____	system
_____	process
_____	development
_____	database
_____	class
_____	application
_____	requirement
_____	management
_____	organization
_____	architecture
_____	user
_____	control
_____	document
_____	incremental
_____	prototype
_____	quality
_____	validation
_____	module
_____	team
_____	solution
_____	information
_____	method
_____	function
_____	component
_____	specification
_____	analysis