IDENTIFYING PROGRAMMER ABILITY USING PEER EVALUATION:

AN EMPIRICAL INVESTIGATION

by

JASON ROBERT OSLIN

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the Graduate School of
the University of Alabama

TUSCALOOSA, ALABAMA

2011

ABSTRACT

The ability of students to accurately rate the programming ability of their peers was investigated. Two studies were performed in the context of undergraduate Computer Science courses to measure how closely student peer ratings matched class grades given by course instructors. The results showed that peer ratings did correlate with instructor grades. Additional data was collected regarding how the students related to each other through previous projects and social network connections. These relations were treated as measures of familiarity. Networks of familiarity were created and clusters within these networks were identified to show which students were more familiar with each other. Analyzing these clusters showed that familiarity plays a role in the accuracy of the peer ratings. Students who were more familiar with each other generally provided more accurate ratings of their peers.

Further testing is warranted to validate these results, since the data gathered was more sparse than anticipated at the beginning of the study. Also, further analysis of the effect of familiarity may yield more compelling results if different network clustering methods are applied.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank the friends and faculty members who have helped me prepare this thesis. I owe the most gratitude to Jeffrey Carver, the chairman of my thesis committee, for introducing me to human aspects in software engineering. His knowledge, experience, and patience were invaluable in helping me achieve my goals as a student. I would also like to thank my committee members, Nicholas Kraft, Cecil Robinson, and Randy Smith for their input on the project. I would like to thank Brandon Dixon and Susan Vrbsky for allowing me to use their classes to gather data. Additionally, I wish to thank the computer science undergraduates who participated in the study. I would especially like to thank my fellow graduate students in the University of Alabama Software Engineering Group who kept me going through trying times and impossible deadlines.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Traditional software engineering research focuses on the development of tools and processes that empower software developers to improve their productivity and the quality of their software. However, more important than the specific technologies used is the underlying ability of each software developer. For example, according to the COCOMO II model, "programmer capability" is the third most significant adjustment factor when estimating schedule, with a potential scheduled influence of 1.76, bested only by "product complexity" (2.38) and "requirements analyst capability" (2.00). By contrast, "use of software tools" has a potential schedule influence of 1.5 and "process maturity" has an influence of 1.43 (Boehm et al. 2000).

In an industrial setting, the ability to accurately identify programming ability plays an important role in tasks such as hiring, determining appropriate compensation, and allocating developers to projects and tasks. From a hiring perspective, this knowledge is particularly important because of the large variation in productivity from one programmer to another. An early estimate by Sackman and Grant indicated that the highest-performing programmer could be up to 28 times better than the lowest-performing programmer (Sackman et al. 1968). A more recent estimate by Prechelt argued that typical range between the highest and lowest performing

programmers is closer to 4:1 (Prechelt 1999). This large variation creates similar challenges for developing compensation systems that appropriately reflect these large variations. This variation appears to be well-known in the software engineering practitioner community; my impression is that much more care goes into selection of programmers than, say, programming language or development process, because the quality of the programmers is believed to have a much larger impact on project outcomes.

From a project management perspective, allocating developers to various projects and tasks is particularly important. To make the most effective use of the human resources, this allocation process needs to be informed by an awareness of the abilities of the development team members. For example, a manager may want to place a disproportionately large amount of expertise on a particularly difficult or particularly important project. Conversely, in typical situations, that manager may wish to spread the developer expertise evenly across a set of tasks or projects.

Programming ability also plays an important role in software engineering research. In particular, this large variation in programmer ability is one of the reasons why so few human-based empirical studies achieve the recommended statistical power of 0.8 (i.e., 80% probability of obtaining a statistically significant result assuming there is a genuine difference among the treatments) (Dyba et al. 2006).

Unfortunately, there are no easy methods for quantifying developers' ability to support the project manager or the software engineering researcher. In empirical studies that require programmer ability as a variable, researchers are often forced to use simplistic measures such as "years of experience" or "professional vs. student" as a proxy for programmer ability. These measures are clearly incomplete and inadequate at best, but no other viable measures are

available. In industry, where the stakes of correctly recognizing programming ability are much higher, some companies resort to elaborate puzzle-based interviews to try to identify potentially high performing software developers (Poundston 2004).

The main difficulty in identifying the best programmers is that there are no good metrics for operationalizing programming ability. While some broad generalizations seem reasonable, e.g. one expects a professional software developer to perform better than a first year computer science student, what the research and professional communities really need are finer-grained measures to differentiate between professionals. For example, there is no good evidence that a particular professional with ten years of experience will perform better than another professional with five years of experience. Also, simple productivity metrics are not adequate for judging programming ability. For example, I claim that the number of lines of code developed per week cannot be used to identify the better programmers despite its proposed use as a productivity metric (IEEE 1992).

The central hypothesis of my research is that even if programming ability cannot be operationalized into an objective metric, developers know it when they see it. If one considers some of the better developers they have worked with in the past, he or she is likely to be able to name them based on personal judgment and experience.

The studies and analyses presented here seek to address the relationship between peer evaluations and programmer ability. The existing work in software engineering literature addresses what quantifies expertise and some of the proxy measures used to control for it. Psychology addresses the assessment of peers, including methods and factors that affect accuracy. Combining these two lines of thought propels this work in addressing the first research question:

Q1: Can developers accurately predict the programming ability of their peers?

Using data from the studies outlined in this proposal, the research provides insight into the efficacy of peer ratings as predictors of developer ability. Establishing a link motivates further investigation into how peer rating information can be used to quantify ability and better describe the differences between lower and higher performers. An additional research question emerged from the studies which seeks to better explain the relationship between peer assessments and developer ability:

Q2: How does familiarity between peers affect the accuracy of those predictions of ability?

The ability of developers to predict the ability of their peers likely will not be uniform for every rater, peer, or group. The motivation for this question is the suspicion that being more familiar with one's peer makes one better able to give an accurate judgment of that peer. Familiarity in this context will refer to interpersonal relationships between developers who assign ratings. These relationships can extend from development-related tasks or social ties between developers.

The benefit to the software engineering community of analyzing these research questions is the possible emergence of a new means of determining developer ability. Such a development would impact the software engineering discipline at the teaching, research, and industrial levels.

Starting from the research questions stated above, I explored these questions with data gathered from two empirical studies I conducted. This work describes my research process. First, I catalogue the related literature to coalesce the software engineering and psychology literature as they relate to this study. Next, I describe the design and execution of the two empirical studies. Then, I explore a post-hoc analysis on the data to see how peer familiarity may affect the

results of the studies. Finally, I synthesize the analysis of the two studies to form more cogent conclusions regarding the nature of the data.

CHAPTER 2

RELATED WORK

There has been extensive study comparing how experts and novices perform software tasks. Sonnentag provides a full literature review (Sonnentag, et al. 2006, von Mayrhauser, et al. 1995). Such studies typically rely on programmer experience as a proxy for expertise. However, as Sonnentag et al., note, most studies use years of performance as a proxy for expertise, when the underlying construct of interest is actually high performance (Sonnentag, et al. 2006). Host et al. tested this hypothesis by comparing software professionals with students. Their study, involving a software engineering judgment task, found no statistically significant difference between students and professionals (Höst, et al. 2000).

Host et al.'s result suggests the importance of identifying factors other than student/professional for predicting performance. Carver et al. sought to identify behavior in programmers' backgrounds that explain variation in performance on finding defects in inspections. (Carver, et al. 2003, Carver, et al. 2008). Dehnadi and Bornat claimed to have identified a test that can identify programming aptitude among novices without any programming experience (Dehnadi, et al. 2006), but a subsequent replication produced a negative result (Lung, et al. 2008).

This paper focuses on peer evaluations and self-evaluations, Bryant conducted a closely related study of expertise ratings in the context of pair programming. The participants each rated their own ability and were rated by their peers. The results of this study indicated that 79% of the high ability programmers underrated their own experience, while 50% of the low ability programmers overrated their own ability (Bryant. 2005). However, as in previous studies, Bryant used "length of tenure" (i.e. years of experience) as an objective metric for ability.

Psychologists have also studied the accuracy of assessments. In terms of the accuracy of self-assessments, Kruger and Dunning report that novices tend to overestimate their own ability because they lack the experience to recognize their own limitations. In addition, experts tend to overrate the ability of their peers because an expert assumes that their peers are as qualified as they are (Kruger, et al. 1999). In terms of the accuracy of peer assessments, Holzbach found that peer ratings were better correlated to ratings of superiors than to self-ratings, suggesting that self-rating and peer-rating are actually measuring different things (Holzbach. 1978). Finally, in a study of peer evaluations of police officers, peer rankings (i.e. ordering peers from 1..n) were very reliable. They were more accurate than peer ratings (i.e. scoring based on some predefined scale) (Love. 1981). The superiority of ratings over rankings is consistent with results from survey research in other areas (Krosnick. 1999).

The existing literature motivates a thorough inspection of how developers assess their peers. Most existing research relies on coarse measures to control for expertise even though developers seem able to make better judgments of ability than these coarse measurements. Peer ratings seem to be a valid method of evaluation. In addition, people may be better able to determine the relative ability of their peers rather than providing an absolute rating. Beginning from existing research on peer assessment, I investigate how students perform a peer assessment

task. Instead of showing the effects of experience or professional status on accuracy, I examine peer ratings from the perspective of familiarity to see if familiarity is a suitable predictor of peer assessment accuracy.

CHAPTER 3

EMPIRICAL STUDIES

I performed two studies to explore my research questions. In this section I describe these studies beginning with the design and execution of the first study. I then give the results of the analyses performed on the data. I describe how lessons learned from the first study affected the study design of the second. I also present the analyses performed on the data gathered from the second study. Using the results of the analyses to motivate a post-hoc analysis, I delve into the effects that familiarity between peers may have had on the initial results of the studies.

The common goal of both studies was to characterize peer assessment of programmer ability in the context of an undergraduate computer science course. The courses that provided the data were computer science classes at the University of Alabama in which programming assignments played a significant role in the students' final grades. The courses were far enough into the curriculum that participants would be familiar with each other. Students received roll sheets with the names of their classmates. Those students who chose to participate rated classmates on a scale. At the conclusion of the semester the instructors of these courses provided their course grades to me. The instructors were not involved in the study in any other way. Using the grades as a measure of 'true' ability, I compiled the ratings of the students and evaluated them against this base measure of 'true' ability. I collected other data regarding the relationships

between classmates, including an indication of whether or not they had worked previously with their peers on software development tasks. Each study implemented this overarching study protocol in slightly different ways, and the following sections discuss those differences.

First Study

The course chosen for the first study was a senior-level programming languages course. Students worked on five programming assignments over the course of the semester: an SSH log parser in Perl, two sets of exercises in F#, a program interpreter in Java, and an RSS reader in Java. The goal of this study, as stated before, was to determine whether or not students could accurately rate the programming ability of their classmates, leading to the first hypothesis of this study:

H1: Developers are able to accurately predict the programming ability of their peers.

The literature suggests that novices overestimate their ability because they lack the experience to recognize their own limitations (Kruger and Dunning 1999), an observation which motivated another hypothesis that could be evaluated in this study:

H2: Low-performers tend to overestimate their ability.

To evaluate these hypotheses, the participants needed to give two pieces of information: an evaluation rating made by themselves and by their classmates and some measure of their true programming ability. For the latter, as mentioned previously, I used the programming assignment grades provided by the class's instructor as a proxy. For the former, I devised a form (Appendix 1) for all the participants to fill out at the beginning of the semester. The form consisted of a class roll with the names of the students in the first column. In the adjacent

columns there were spaces for participants to provide a rating. The form instructed participants to use the following scale to assign ratings:

A = student is in the top 20% of the class

B = student is in the top 21% - 40% of the class

C = student is in the top 41% - 60% of the class

D = student is in the top 61% - 80% of the class

E = student is in the bottom 20% of the class

In addition to this rating, the form also had spaces for participants to provide an estimation of their confidence in that rating on a 3-point scale (C-Low, B-Medium, A-High). Finally, the form instructed students to indicate if they had worked on a project with their peers and if they had attended a class with those peers.

At the end of the semester, I divided the grades from the instructor (on a 0-100 scale) into quintile groupings of the participants. The grouping of grades by quintile matched the scale that the participants used to rate each other, enabling a side-by-side comparison of the peer ratings and the measure of actual ability. Figure 1 shows the instructor-provided grades divided into quintiles.

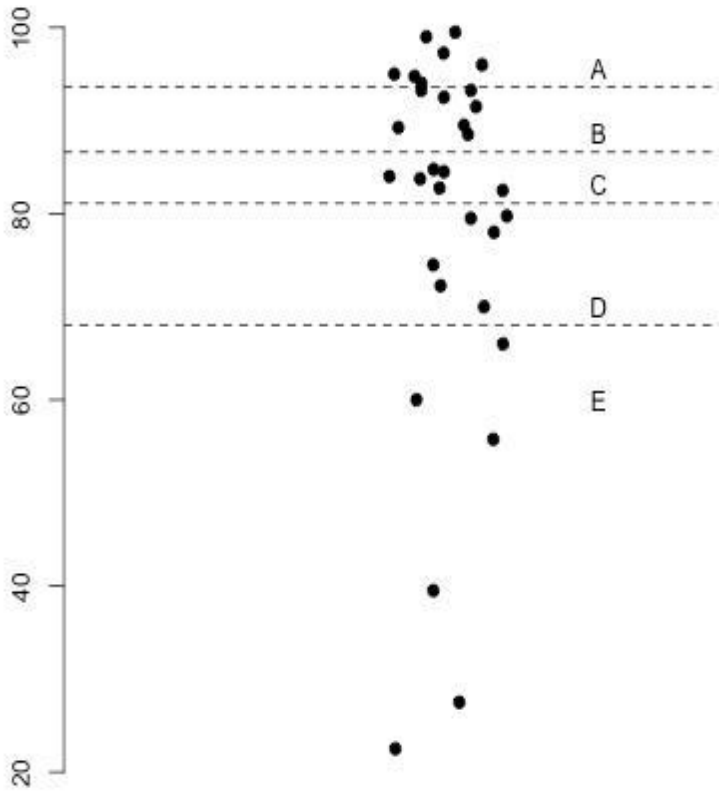

Figure 1 – Division of Quintiles based on actual grade

The students in the top quintile received a 'true' score of 'A', and students in the bottom quintile received a 'true' score of 'E' to match the A-E scale used on the peer evaluation forms. The data collected at the end of the experiment showed that 18 of the 36 students enrolled in the course participated in the peer rating exercise at the beginning of the semester. Four students dropped the course during the semester, so programming project grades from 32 students remained.

A trend that stood out just by inspecting the data set was that classmates were not as familiar with each other as assumed, making the rating coverage sparse. Figure 2 shows this distribution, with shaded cells indicating a rating (dark borders for self-ratings), rows representing raters, and columns representing those being rated. I expected to see many more shaded cells indicating denser rating coverage.

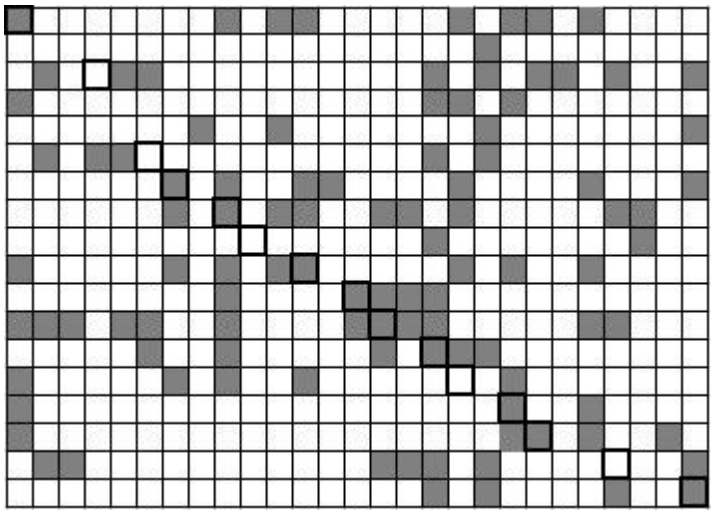

Figure 2 – Distribution of ratings

Another trend immediately noticeable in the data set is that ratings were skewed heavily toward positive ratings. If everyone provided ratings for everyone else and adhered to the quintile rating structure, one would expect to see as many low ratings as high ratings, but the data do not show this trend.

Hypothesis 1 – Developers are able to accurately predict the programming ability of their peers

Recall that the first hypothesis was that developers are able to accurately predict the ability of their peers. For this hypothesis, I compared the grade data from the instructor against

Central Tendency	Spearman	p-value
Average	.432	.039
Weighted Average	.432	.04
Median	.489	.018
Mode	.684	.002

Table 1 – Comparison of Ratings to Actual Scores

the rating information from the participants. For each participant, I aggregated the ratings given by peers by four measures of central tendency: mean, weighted average, median, and mode. Weighted average was a measure that used the peer rating in conjunction with confidence to give more weight to ratings associated with higher confidence. The formula for computing the weighted average was to multiply each rating by the confidence and then divide by the sum of all of the confidence values. This formula ensures that the peer rating value remained in the given 5-point scale, but it provided extra weight to those ratings that were given with more confidence. A Spearman Rho correlation of grades to peer ratings was computed for each of these measures. The results are shown in Table 1.

The mode was the most accurate predictor, so I performed a linear regression of the mode against the actual grades, yielding a slope of 1.23 (not far off from 1.0) which is

Central Tendency	*Spearman*	*p-value*
Average	.283	.19
Weighted Average	.283	.19
Median	.358	.093
Mode	.466	.052

Table 2 – Comparison of Ratings to Modified Actual Scores (Quintile based on grading scale)

statistically significant. The adjusted R-squared value of the linear regression was 0.3997, indicating that peer rating accounted for nearly 40% of the total variance in the data set.

Recall in the description of the data that both the grades and the ratings were skewed toward the higher end of their respective scales. One possible explanation was that students may have a hard time thinking of their classmates in terms of quintiles and rated them on letter grade instead. To test this hypothesis, I altered the grade scale for the 'true' score. Originally, the 'true' scale was a division of students into quintiles based on grade. I modified this division so that the

Central Tendency	Spearman	p-value
Average	.482	.02
Weighted Average	.467	.025
Median	.547	.007
Mode	.808	< .001

Table 3 – Comparison of Ratings to Modified Actual Scores (Based on traditional grading scales)

grade scale itself was divided into quintiles, i.e. grades between 80 and 100 were scored into the A group, 60-80 in the B group, and so on. I performed the same correlations (Table 2), but the correlations were weaker than those from dividing students into quintiles.

I ran the correlations again on one more translation of the grade scale into the 5-part division. In this case, I applied the rating scale in a traditional letter-grade manner, such that 90-100 was an A, 80-90 was a B, etc. As shown in Table 3, the correlations this time were overall stronger than for the initial analysis.

Hypothesis 2–Low performers over-estimate their ability

Recall that the literature motivated the second hypothesis for the first study, which was that lower performers would tend to over-estimate their ability. However, only eight participants

	Estimate	p-value
Slope	-0.41	0.25
Intercept	5.6	0.0085
Adjusted R-squared	.078	.25

Table 4 – Linear Model of Self Ratings

provided self-ratings, and all but two fell into either the A or B category on the 'true' scale. Given these trends in the self-ratings, I developed an alternate hypothesis stating that an inverse

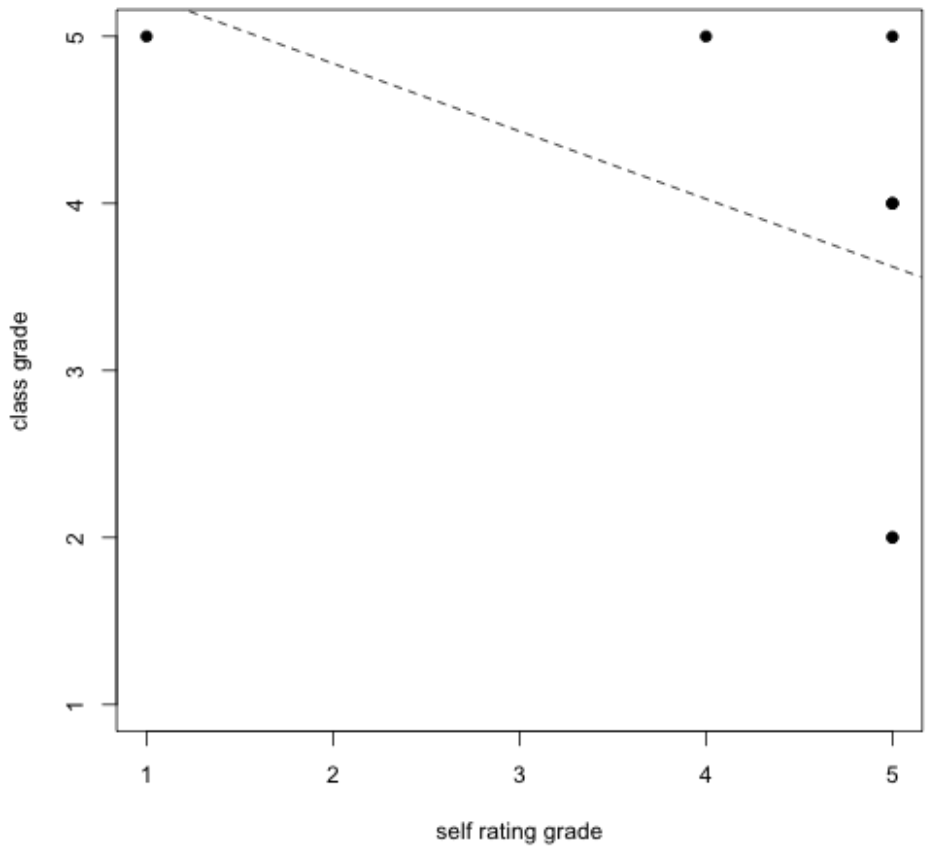

Figure 3 -- Grades vs. Self Rating

relationship would exist between programmer ability and that programmer's perception of their ability. I performed a linear regression to evaluate this hypothesis. The trend line (Figure 3) was downwards, but the small sample size did not yield enough statistical weight to give significant results. The regression data is shown Table 4.

Lessons Learned

The first study was an exploratory study, thus the data did not lead to firm conclusions about the nature of peer evaluation. The most significant impact of the first study was to motivate changes to implement for another study. There were several lessons learned from the first iteration of the investigation. The biggest issue to address was the low number of rating

scores given by the participants. I learned later from some of the students that they did not recognize the names on the rating sheets, knowing their peers only by face or by nickname. Also, this study brought into question the A-E scale. Students may have a difficult time rating into quintiles, and it may be more relevant to ask them what grade they would expect a peer to make. Additionally, there were other interacting variables of interest, specifically those relating to peer familiarity that should also be accounted for. The lessons learned motivated the changes made in the second study, which the next section describes.

Second Study

I performed the second study on two classes instead of one. Again, I studied a senior-level class. This time it was an algorithms course (CS470). The other course was a sophomore-level data structures course (CS357). The reason for pulling in the sophomore-level course was two-fold: one, I wanted more data to analyze; two, I wanted to have data sets in which I could expect a reasonable difference in the amount of familiarity the groups had with each other. I expected participants in the senior-level course to be more familiar with their peers than those in the sophomore-level course, and I thought this distinction may lead to interesting results when comparing the two classes to each other. Table 5 describes the programming activities undertaken by both groups.

Course	Assignment	Description
470	1	Devise an experiment comparing and evaluating either a pair of sorting or order statistics algorithms.
470	2	Implement the Bellman-Ford shortest paths algorithm and Johnson's modification for sparse graphs
357	1	Implement a queue
357	2	Read in traces of a binary tree, perform some processing and output the resulting tree
357	3	Implement a dictionary using a hash table
357	4	Implement an AVL tree

Table 5 – Programming Assignments

In this study, as in the previous study, the primary focus was to evaluate peer judgment of expertise. This experiment showcased a few new features in response to the lessons learned from the first study. The data collection form (Appendix 2) contained more fields, including spaces to indicate connection in a social network (Facebook, MySpace, etc.), whether or not the peer is considered a better or worse programmer than the rater, and how likely the rater would choose the peer for work on a team project.

For this study, I chose numerical scales instead of letter scales in an attempt to divorce the peer rating activity from a grading activity that, as students, they may associate with evaluation. The programming ability scale changed from an A-E scale to a 1-5 scale, with 1 indicating low ability and 5 indicating high ability. The three-point rating confidence scale also changed to a numeric scale, again having 1 indicating low confidence in the rating and 3 indicating high confidence. The 'worked on project' field remained a yes-no field as was the social network field, and the better or worse field was similarly binary. The 'choose for team' field scale was 1-5, with 1 meaning the rater would probably not choose the peer for a team and 5 meaning the rater would very likely choose a peer for a team assignment.

Another major change to the data collection form was the addition of space for *ranking* peers from best to worst. Ranking peers seems also to be a valid method of evaluation. Whereas before I was looking for agreement between peer and superior rating scales in an absolute manner, in the second study I wanted to evaluate how much they agree in a relative sense. It may be better to ask, "Do you agree on who is better?" than to ask "Would you rate these people similarly?" Krosnick's research indicated that rankings may be superior to ratings in determining relative skill (Krosnick 1999). The participants ranked up to 10 peers with whom they were familiar from best to worst.

Finally, given my assumption that the two samples had a different level of familiarity with one another, I sought to explore how this difference would affect the two populations. I focused particularly on familiarity and confidence. I assumed that familiarity would inspire confidence in ratings, and that ratings given more confidently were likely more accurate.

Motivation from the first study and these new concerns inspired five hypotheses for this study:

H1: Developers can accurately judge the programming ability of their peers

H2: Relative rankings are more accurate that absolute ratings

H3: Developers are more accurate assessing others than themselves

H4: Connection in a social network is a good predictor of accuracy and confidence

H5: Confidence is a good predictor of accuracy

The changes in data collection were not limited to the changes in the data collection form. As noted from the previous study, some students could not make peer assessments because they could not identify their peers by name alone. To address this concern, class roll was called upon handing out the data collection forms. As each student's name was called, he or she stood up. This method allowed people in the class to make name to face associations and provide a more complete set of evaluations.

Data Overview

In CS357, 11 out of the 20 students who completed the class consented to participate in the study and filled out the data collection forms. In CS470, 20 out of the 45 students who completed the class consented to participate in the study and filled out the data collection forms. In CS357, there were 59 peer ratings assigned by participants out of a maximum possible 121

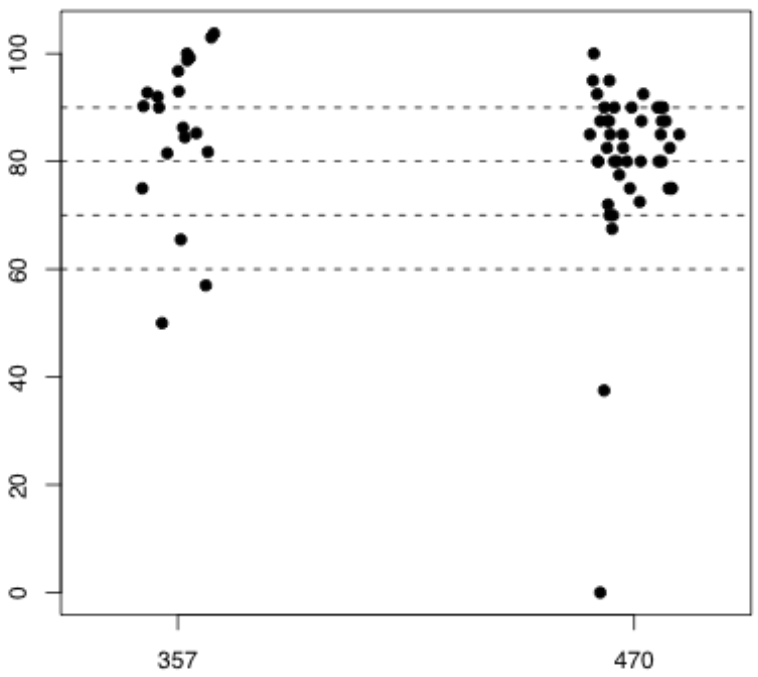

Figure 4 -- Grade Distribution

(49% coverage), and in CS470, there were 128 peer ratings assigned by participants out of a maximum of 780 (16% coverage). In my previous study, I obtained 95 out of a maximum possible 471 ratings (20%); thus, I saw an improvement in coverage for CS357 but a reduction for CS470.

In each class, the professor reported the average programming grade on a 100-point scale. To enable comparison to the 5-level ordinal scale used by the raters, I converted the programming grade to the standard letter grading system used at the University of Alabama (A: ≥90, B: 80-89, C: 70-79, D: 60-69, F: <60). Figure 4 shows how the grades were distributed. Note that in CS357 it was possible to score above 100.

Class	*Mean*	*Weighted Mean*	*Median*	*Mode*
357	.602*	.558	.719*	.480
470	.137	.093	.145	-.84

Table 6 – Pearson Correlation of Grade to Peer Rating

On the data collection forms, some participants responded with invalid data. Some of the students reported confidence on a 1-5 scale instead of a 1-3 scale. For these students (three in CS357, two in CS470), I mapped the data from a 1-5 scale to a 1-3 scale with [4,5] => 3, [3] =>2, and [2,1] => 1. When asked "Is this person better or worse than me", several students responded "same". I discarded this value and treated it as if the student had not responded.

Hypothesis 1 – Developers can accurately judge the programming ability of their peers

Just as in the first study, I computed four measures of central tendency: mean, weighted average, median, and mode. Table 6 shows the Pearson correlation of grade to peer rating. In this study, the median provided the best results, whereas in the last study the best results came from mode. Also, weighting the average with the confidence actually decreases the strength of the correlation.

	Raw Scores (Pearson's r)		*Ranked grades (1-N) (Pearson's rho)*		*Letter grades (Pearson's rho)*	
Class	Mean	Median	Mean	Median	Mean	Median
357	.602*	.719*	.618*	.631*	.463	.525
470	.137	.145	.132	.145	.181	.233

Table 7 – Correlations of Raw and Transformed Scores

In addition to correlations against the raw score, I calculated correlations between a ranking of the grades (1-N) and a translation of grades into a standard letter grade scale (A-F). These correlations were made on the strongest two correlating factors from the first analysis, mean and median. The results of these correlations are shown in Table 7.

Finally, I performed a linear regression on the raw score using the mean and median results. Both measures were statistically significant for CS357, while neither was significant for CS470. Therefore, the first hypothesis is supported by data from CS357, but not for CS470.

Class	*Pearson's r*	*Adjusted R^2*
357	.719	.467
470	.145	.001

Table 8 – Effect sizes for absolute ratings using median as central tendency

	Correlation
Pearson's r	.438
Kendall's tau_b	.470
Spearman's rho	.621

Table 9 – Correlation for Relative Rankings (CS357)

Hypothesis 2 – Relative rankings are more accurate that absolute ratings

From the previous section, I observed the effect sizes shown in Table 8 for the median for the absolute ratings.

To make a comparison of this data against the absolute ratings, I computed correlations between the rankings given from the peer evaluations and the rankings based on the actual grade data. Table 9 shows the correlations, all of which were significant at the $p < .05$ level.

When looking at CS357, I can directly compare the Pearson's r from before and see that the hypothesis is not supported by the data. The ratings students gave correlated more strongly with instructor scores than student rankings correlated with rankings derived from the instructors gave.

For CS470, many students received the same grade, which introduced some complications to the analysis. I analyzed three strategies for breaking ties:

1. Repeated tie scores: If several students had tie scores, I assigned each the top rank. For example, if two students tied for second place, they would each be assigned a ranking of 2.

2. Averaged tie scores: If several students had tie scores, I assigned each of them the mean of the tied rankings. For example, if two students tied for second place, they would be assigned a ranking of mean(2,3) = 2.5.

3. Maximize correlation: If several students had tie scores, I broke the tie in the way that was consistent with the student ranking. For example, if two students tied for second place, the rankings of 2,3 would be assigned to them using the same relative ordering as the student who rated them.

The results of running correlations with each data correction method are shown in Table 10.

	Repeated tie score	*Average tie score*	*Max correlation*
Pearson's r	.249*	.286*	.368*
Kendall's tau_b	.199*	.227*	.291*
Spearman's rho	.254*	.295*	.366*

Table 10 – Correlation for Relative Rankings (CS470)

In this case, CS470 did slightly better by ranking peers than rating them, but the correlations in both cases are low.

Hypothesis 3 – Developers are more accurate assessing others than themselves

Previous research which indicates that self-assessments can be very inaccurate motivated this hypothesis. Students in the previous study indicated more confidence in their self-judgments than those they received from their peers, a notion not supported by the data.

To evaluate the hypothesis, I compared the error in assessment between a student's self-rating and an aggregation of that student's error in judging others. Error in this case means the absolute magnitude of the difference in rating from the actual grade on the 5 point scale. If a student earned a 4 in the class but was rated a 5, the error score would be 1. If a student earned a 5 in the class but was rated a 4, the error score would also be 1. The aggregate of error in the peer judgments is the mean and the median of these errors.

For CS357, calculating the peer judgment error by taking the mean of each student's errors shows a significant pairwise correlation with the self-judgment error, indicating the two measures are not different ($r = .767$, $p = .01$). Additionally, a paired samples t-test reveals that there is not a significant difference between the self-error and peer error ($p = 0.915$), also indicating the error measures are not different. Using the median peer error value to compare against self-judgment error shows a strong but not significant correlation ($r = .587$, $p = .074$). In the paired samples t-test, the significance value is lower (0.591) and does not indicate a significant difference between self-judgment error and peer judgment error.

For CS470, the correlations between the self-judgment error and peer judgment error are smaller for both measures of central tendency. A paired samples t-test reveals significance values

of 0.978 and 0.756 for mean and median, respectively. These values do not approach the significance needed to declare these measures significantly skewed from each other. This result supports the previous figure showing values scattered on both sides of the diagonal.

Given these results, the data does not support the hypothesis that developers are better able to judge peers than themselves.

Hypothesis 4 – Connection in a social network is a good predictor of accuracy and confidence

The idea that familiarity increased accuracy and confidence inspired this hypothesis. I used connection in a social network to operationalize familiarity for this analysis. To test this hypothesis, every peer rating given fit into one of two groups. The first group consisted of ratings between students connected in a social network, and the other group was made up of ratings between students not connected in a social network. I compared the accuracy and confidence between these groups.

For confidence, an independent sample t-test over all the rating data showed a significant positive correlation between confidence and connection in a social network ($p < 0.001$). However, the positive correlation did not emerge in both classes. The members of CS470 showed a strong correlation between confidence and social network connection, whereas the members of CS357 showed no significant correlation. The results of this analysis are shown in Table 11.

Class	t	df	p-value
357	.542	57	.59
470	4.106	118	<.001*
Both	3.611	177	<.001*

Table 11 – T-test for Social Network and Confidence

For accuracy, I performed the same test to investigate correlation between rating accuracy and social network connection. Overall, there seemed to be no correlation at all between the two (p = .962). As shown in Table 12, neither class proved to be significantly better than the other at predicting accuracy regardless of social network connection.

Class	t	df	p-value
357	.894	57	.375
470	.952	118	.343
Both	.048	177	.962

Table 12 – T-test for Social Network and Accuracy

According to the data, connection in a social network raised confidence in raters, especially in CS470. However, any familiarity assumed by connection in a social network did not lead to significant changes in the accuracy of the ratings. In other words, raters were more confident but were not more accurate.

Hypothesis 5 – Confidence is a good predictor of accuracy

Recall from Hypothesis 1 that I performed the correlations to the actual grades against both the mean and the confidence-weighted mean. In both classes, the correlations actually went *down* when the ratings were weighted by confidence. Reduced correlation here suggests that

confidence does not have a positive effect on accuracy. Another analysis performed looked at the variance of the accuracy based on the confidence of the rater. The results are shown in Table 13 and Table 14.

The analysis reveals no statistically significant effect due to confidence, which reinforces the suspicion from the data in Hypothesis 1 that confidence is not a good predictor of accuracy.

	Df	*Sum sq.*	*Mean sq.*	*F-value*	*p-value*
Confidence	1	.44	.44	.58	.4511
Residuals	57	43.19	.76		

Table 13 – ANOVA (CS357)

	Df	*Sum sq.*	*Mean sq.*	*F-value*	*p-value*
Confidence	1	.02	.023	.04	.8393
Residuals	126	62.70	.50		

Table 14 – ANOVA (CS470)

Discussion

Going into this study, I expected to find support for all five hypotheses, given the results from the previous study and from the published literature. I was therefore surprised to find that H2, H3, and H5 were not supported by the data from the study.

I was also surprised by how different the results of the study were between CS357 and CS470. Even more surprising was that the peer evaluation performance was better in CS357, a sophomore level class, than it was in CS470, a senior-level class. I assumed that the senior-level students would know each other better and therefore would be better judges of programming ability.

The lack of support for H5 is particularly surprising, because it suggests that the participants in the course did not have an accurate sense about how well they were able to do

peer assessment. Combined with the results from H1, this leads to an important conclusion: programmers may sometimes be good at assessing the ability of peers, but not always, and their own personal level of confidence in their assessment is not a good guide to accuracy.

A Post-hoc Analysis of Familiarity

At the beginning of the second study, I made the assumption that students in CS470 would be better predictors of peer ability than the CS357 students because I assumed that the senior level CS470 students would be more familiar with their peers than would their sophomore level CS357 counterparts. Analysis of the data showed the opposite. The sophomore class as a whole gave more accurate ratings than the senior class. This observation may seem to indicate that familiarity was not a significant factor at all, or that it introduced some bias that made ratings inaccurate. More likely, seniority is a poor proxy for familiarity. A review of peer studies by Falchikov and Goldfinch concludes that seniority does not significantly affect student-teacher agreement in performance assessment (Falchikov et al. 2000). However, this only discredits seniority, not familiarity.

During the second study, I observed that some of the ratings were distributed in a way such that certain sub-groups within the class seemed to provide most of their ratings within that group. I proposed that the efficacy of peer ratings should be broken down to a finer granularity than compiling statistics over the entire class. Instead, it may be more interesting to evaluate how well peer evaluations predict ability amongst members of these groups. A good way to visualize these groups is to draw graphs, where each node represents a student and the connections between nodes represent some relationship between students. Figure 5 shows a graph of who rated whom amongst students in the data structures class evaluated in the second study. The

nodes represent students, and each line between the nodes represents a rating given from one student to another. This graph in itself serves only to show the density of the ratings. However, with further analysis, using graphs potentially provides insights into how a large group behaves as a bundle of smaller groups, or clusters.

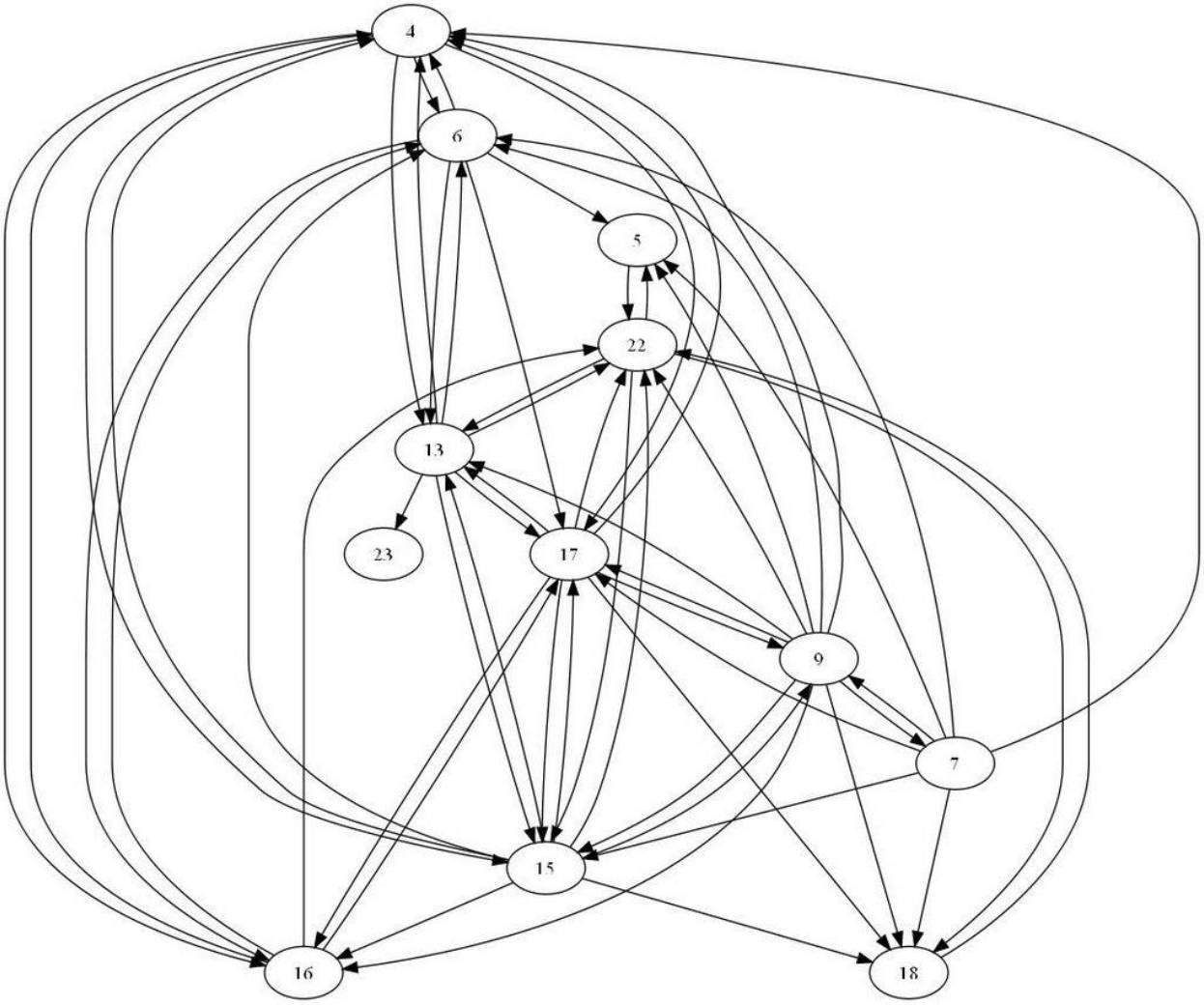

Figure 5 – Graph of Ratings in CS357

A graph-based analysis of peer network clusters

The most obvious way to form these clusters for analysis would be to break out groups of nodes that are completely connected to each other and analyze them independently of the rest of the data set. For example, if students 13, 17, and 22 all rated each other, they would form a *complete graph* if broken out from the rest of the data. Being a complete graph on their own, these students form a *clique* in the context of the graph over all the data. Ideally, the data would be dense enough to yield several cliques to be measured and compared against each other and classes as a whole.

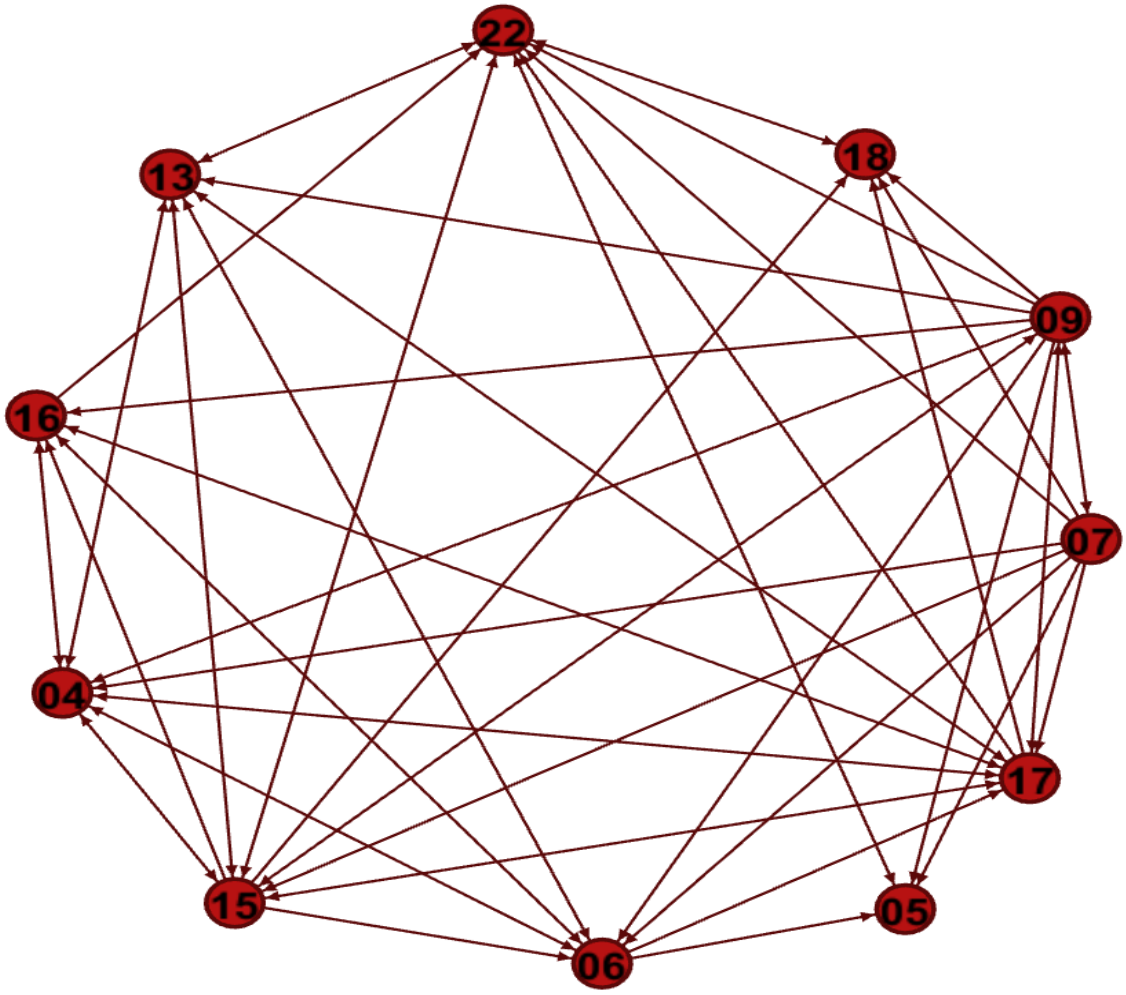

Figure 6 – Graph of ratings in CS357 with Louvain clustering applied. Colors represent clusters. This graph contains only one cluster.

Upon inspecting some initial graphs, however, I found the data to be too sparse (as noted previously) to yield significant cliques, so I employed a different clustering method for this analysis. To try to identify communities within each class, I applied the Louvain method of community detection (Blondel, et al. 2008), as implemented by the Gephi visualization tool (Bastian, et al. 2009). This method repeatedly passes over a graph, pulling nodes into communities based on whether or not adding a node to a community increases the modularity of the graph. A modular graph has dense connections within clusters and sparse connections between clusters.

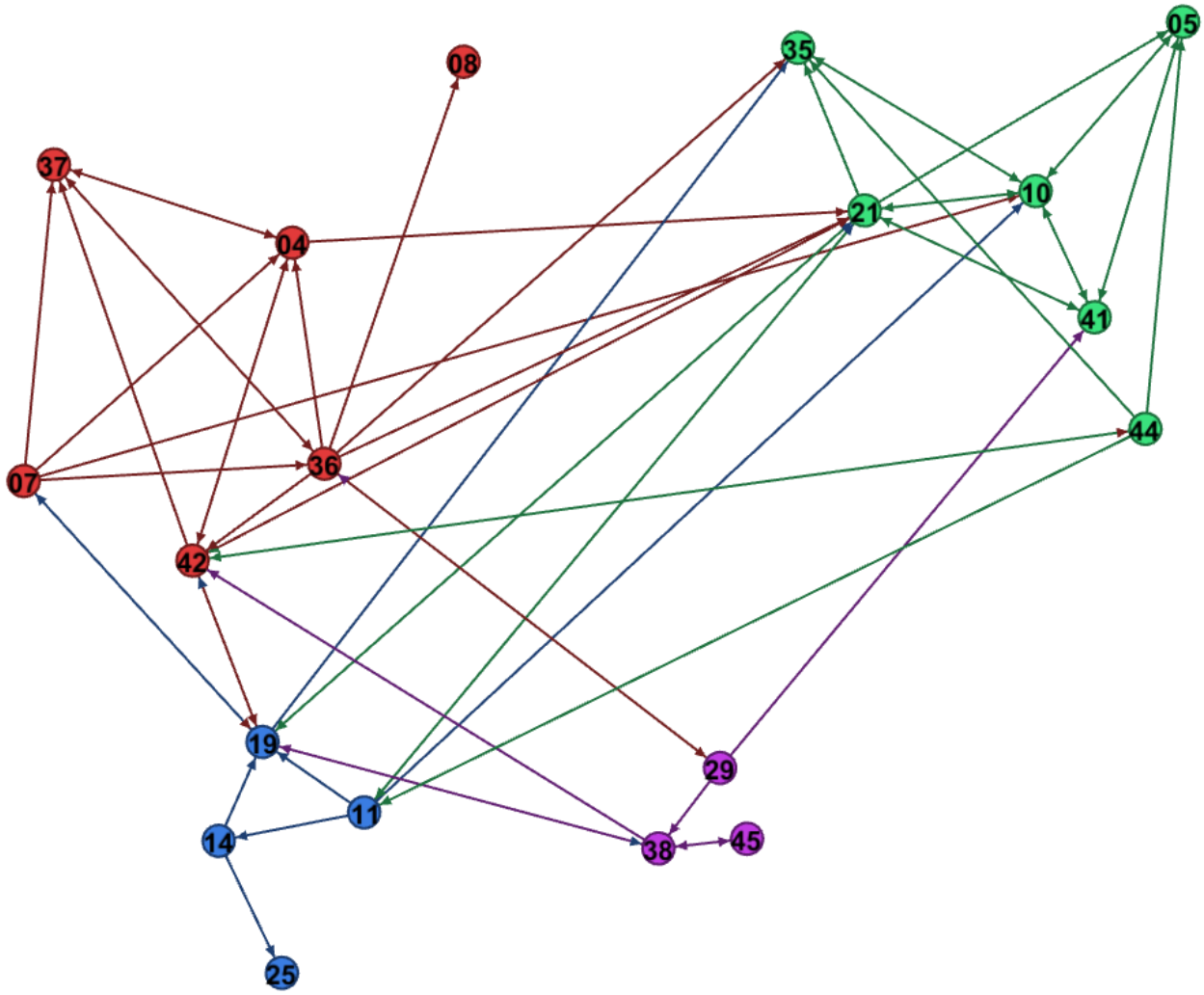

Figure 7 – Graph of ratings in CS470 with Louvain clustering applied. Colors represent clusters.

With a method in place to define clusters, I performed a preliminary assessment of the communities within both classes from study 2. I created a graph for each class of all peer ratings given in the class. The clustering method yielded one large community for CS357 (Figure 6) and four smaller communities within CS470 (Figure 7). What does the cohesiveness of the CS357 class mean in regards to rating accuracy? Familiarity does not seem to hinder accuracy. Sonnentag concluded that likeability does not affect the validity of a performance evaluation (Sonnentag 1998). In addition, Magin conducted a study that showed that reciprocity does not affect the validity of peer marks, and student relationships do not compromise the peer measurements (Magin 2001).

From the graphs and the literature, I developed the following hypotheses for testing graphs of student relationships:

H1: Student ratings within clusters are more accurate than ratings between clusters.

H2: The more tightly connected a cluster, the more accurate the ratings within it are.

Building the graphs

I built a graph for each connecting factor in each class. For CS357 and CS470 from study 2, this means I created a graph for ratings, who worked on a project with whom, and social network. For study 1, I created a graph for ratings and who worked on a project with whom. A graph of ratings captures every rating given; therefore its edges comprise a superset of the edges in the other graphs. Participants from study 1 also provided info about whether or not they had taken a class with their peers, but nearly every rating response was ‘yes’, making the resulting graph nearly identical to the ratings graph.

I translated the rating data from the students into graphs by creating a node for each student and an edge for every rating data point. Each rating data point contained information identifying the rater, the ratee, the rating score from the rater, and the actual score given by the professor. For the graphs showing who rated whom, the edges are directed, meaning that a connection from node A to node B did not imply a connection from node B and node A. I made this distinction because there were many cases where one student would rate another without reciprocation. For graphs of social network and previous work on a project together, I used undirected edges. An undirected edge shows that a connection exists between two nodes and

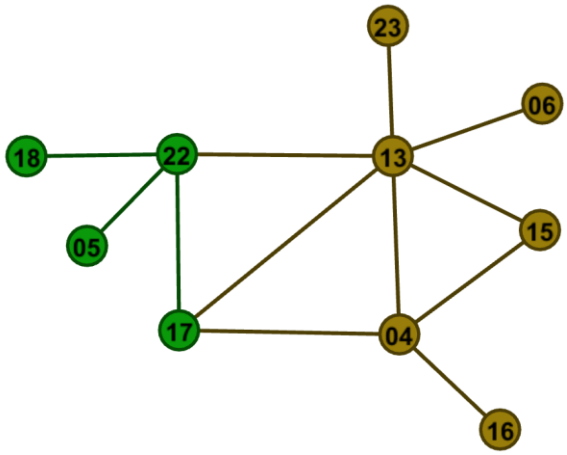

Figure 8 – Graph of social network clusters in CS357 with Louvain clustering applied. Colors represent clusters.

nothing about who reported the connection. I assume that these factors are implicitly reciprocal, even if a data edge only existed in one direction.

Once these graphs were built, I used the Louvain community detection method to identify clusters of students in the data. The method categorized the nodes into separate communities. I used these clusters in the analysis as a proxy for familiarity. I assumed that nodes within a cluster represent students that are more familiar with each other than with those outside the cluster. I

also classified each edge (i.e. peer rating). Edges that connect nodes in the same cluster are intra-cluster edges. Edges that connect nodes from different clusters are inter-cluster edges. In Figure 8 for example, the edge between node 13 and node 15 is an intra-cluster edge. The edge between node 13 and node 22 is an inter-cluster edge. Every edge falls into one of these two categories. With these distinctions made, I analyzed how familiarity between students affects the accuracy of their ratings.

Hypothesis 1 - Student ratings within clusters are more accurate than ratings between clusters.

I hypothesized that ratings of students within clusters is more accurate than rating students between clusters. In terms of the graph, that means I expect intra-cluster edges to be more accurate than inter-cluster edges. Each edge corresponds to a single rating and contains information about the score given by the peer and the grade given by the instructor. To analyze the first hypothesis, I calculated Spearman's rho to show the correlation between the instructor grade and the peer rating across the intra-cluster edges and the inter-cluster edges.

	Rating	*Social Network*	*Worked on Project*
Intra-cluster	.347*	.343	.630*
Inter-cluster	*	.775	*

Table 15 – Spearman's Rho for CS357 Graphs with Louvain Clustering

The results of the correlations for CS357 are shown in Table 15. Asterisks indicate that there was no data to analyze. For CS357 I noted before that the class forms one large cluster for ratings. Since there is only one cluster, no inter-cluster edges exist. The correlations for the intra-cluster ratings show significance for the ratings graph and the project collaboration graph. However, since there are no inter-cluster ratings in these graphs, these results cannot be used to support or refute the hypothesis. They do reinforce the notion that clustering produces communities that can perform the rating task well.

The CS357 social network graph, on the other hand, shows no statistically significant correlation for either set of edges, but the inter-cluster edges show slightly higher correlation between peer ratings and instructor grade. This result casts doubt on the hypothesis, but supports the result from the second study that social network does not significantly correlate with rating accuracy.

CS470 provides a more interesting study for this hypothesis, since the rating graph for the class breaks down into four clusters. The results of the correlations for CS470 are given in Table 16. Like CS357, CS470 shows a statistically significant correlation between the peer rating and instructor grade on intra-cluster edges when clustering on the rating graph. This correlation is the only significant one for the class, but intra-cluster ratings show better correlations than inter-cluster ratings for the social network graph and the project collaboration graph. The results from CS470 support the hypothesis.

	Rating	*Social Network*	*Worked on Project*
Intra-cluster	.341*	.136	.193
Inter-cluster	.138	-.110	-.256

Table 16 – Spearman's Rho for CS470 Graphs with Louvain Clustering

In the case of study 1, the data collected was narrower and yielded only two graphs for clustering: a ratings graph and a project collaboration graph. Table 17 shows the results of the intra-cluster and inter-cluster correlations on this data. Again, the intra-cluster edges of the rating graph show a significant correlation between peer ratings and actual grade. The inter-cluster

	Rating	*Worked on Project*
Intra-cluster	.371*	-.131
Inter-cluster	-.101	*

Table 17 – Spearman's Rho for Study 1 Graphs with Louvain Clustering

ratings show a slightly negative correlation. For project collaboration, intra-cluster ratings show a slight but insignificant negative correlation. There exists only one inter-cluster edge, so it does not make sense to try to establish any correlation to compare against the intra-cluster rating. The significant correlation in the rating graph for intra-cluster ratings supports the hypothesis.

Analysis of hypothesis 1 shows that for all classes clusters based on who rated whom yielded significant correlations between peer ratings and actual grades for intra-cluster ratings. Project collaboration showed a significant correlation for CS357. Most of the results showed stronger correlations for intra-cluster ratings which supports the hypothesis. Social network connection did not uphold this trend. While contrary to the hypothesis, this result is consistent with the data gathered from study 2. The rating graphs, which showed the strongest support of the hypothesis, were also the most dense. This observation implies a link between cluster density and rating accuracy. The analysis for hypothesis 2 further explores this link.

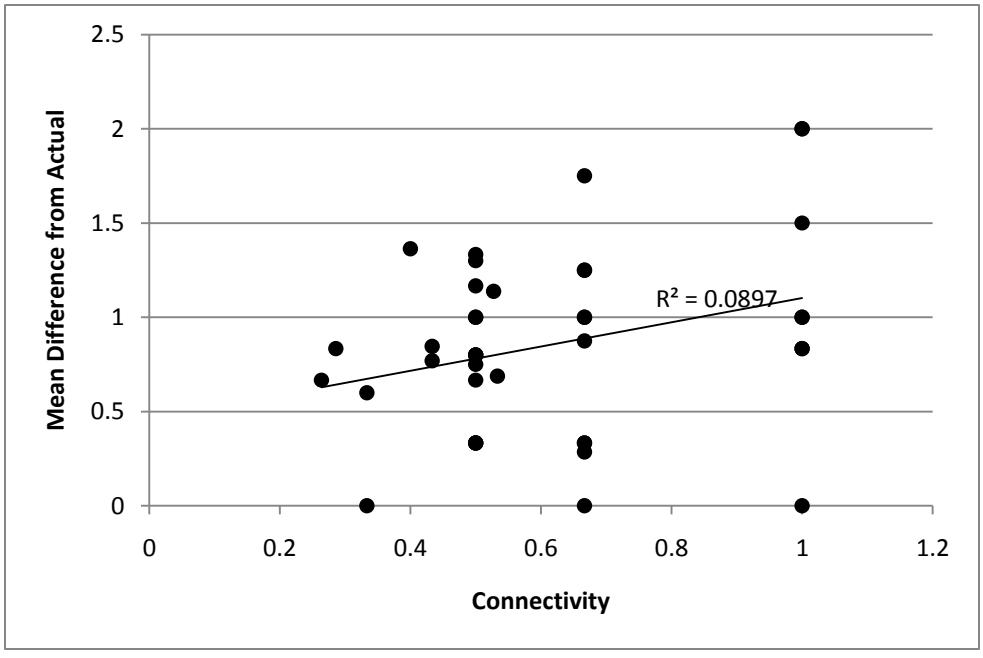

Figure 9 – Cluster connectivity against the average error of its ratings. Each point represents one specific cluster from each of the relationship graphs.

Hypothesis 2 - The more tightly connected a cluster, the more accurate the ratings within it are.

The first hypothesis represents an expectation that familiarity coincides with accuracy. The second hypothesis emerges from trying to quantify degrees of familiarity. To quantify the degree of familiarity of students within a cluster, the "connectedness" of a graph is used as a proxy. Connectedness of a cluster for my analysis is simply the ratio of edges within a cluster to total possible edges in a cluster. For the ratings graphs, where edges are directed, a cluster of size *k* can have *k(k-1)* edges. For the undirected graphs, a cluster of size *k* can have $\frac{k(k-1)}{2}$ edges.

When analyzing the accuracy of the clusters, I aggregated the error of each cluster by taking the average margin between the peer rating and the actual score for each rating in the cluster. This method is similar to the aggregation of ratings for students in the analysis of the class studies from before. I also calculated the connectedness of each cluster using the method described above. Using the aggregate error and the connectedness of each cluster I created the plot in Figure 9.

I expected to see an inverse relationship between connectivity and error. The plot instead shows a trend line with a slope of nearly zero and an R-squared value less than 0.1. The connectedness of the clusters shows practically no correlation to the accuracy of the peer ratings. The data does not support my hypothesis.

Revisiting the clustering method

For the previous analysis I used the Louvain method for finding communities within graphs. This method worked well for the dense rating graphs in particular. However, for the sparse project collaboration and social network graphs I noticed that the Louvain method emphasized network inclusion over cluster density. Take the CS357 social network graph from

Figure 8, for example. The Louvain method divided these nodes down the middle, leaving many edges between clusters. Intuitively, these central nodes seem to form a cluster in themselves. The outliers were pulling dense, centralized clusters apart as a consequence of creating modular clusters.

For CS470's project collaboration graph (the left side of Figure 10), two trends emerge that may weaken the clusters: chaining and fan-out. Chaining (seen in the relationship of the blue nodes) results from poorly connected nodes that get grouped together despite some nodes in the cluster being more than two edges away from others in the same cluster. Fan-out (seen in the emerald and purple clusters) results from one rater rating many people that did not themselves turn in ratings. The only connection these nodes have to the class is one particular student who rated them. Thus, the connection of these otherwise disconnected nodes pulls the fanning node away from more centralized node clusters.

Applying the Louvain method resulted in clusters that did not appear well-connected compared to some nodes that connected across clusters. I wanted to explore this effect by creating new clusters in a manner that appeared more intuitive upon inspection. To do this, I developed a heuristic for trimming off the offending outlier nodes that pulled other nodes away from fuller clusters:

1. Start with a raw graph with edges and nodes but no cluster data.
2. Prune off all nodes with only one edge.
3. Remove any two-edged node that would leave two disconnected subgraphs.
4. Find any node connecting two other nodes with only two edges and remove it.
5. Repeat until no more nodes are pruned from the graph.

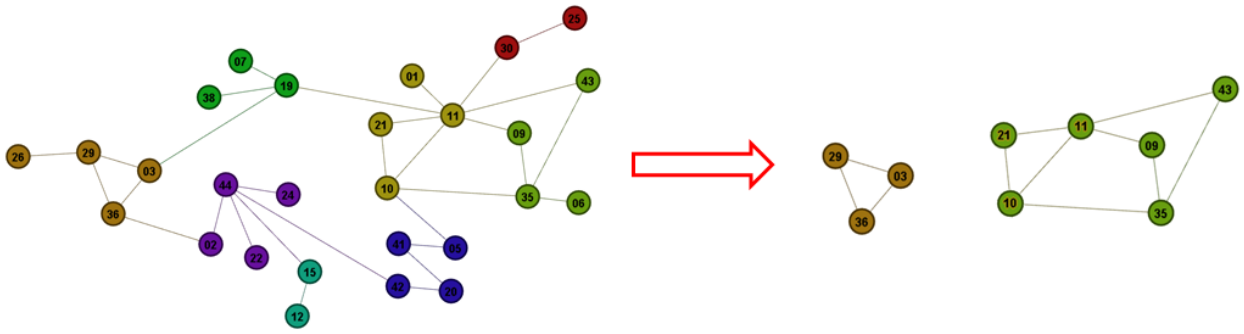

Figure 10 – Applying alternate cluster method to CS470 project collaboration graph. Louvain clustering is shown on the left. Alternate clusters are on the right.

This heuristic is straightforward, except for step 4. Step 4 works to eliminate chains by removing any node that connected two other nodes that each only had two edges. For example, if a node A had an edge to node B and an edge to node C, I looked at the number of edges connected to nodes B and C. If B and C had only two edges (one to node A, another to some other part of the graph), then I eliminated A for being a chaining node. I made an exception for triplets, where nodes A, B, and C were all connected to each other. By eliminating edges in this way I created new clusters. The remaining edges served as intra-cluster edges. Every other edge that was thrown out during the new clustering process became an inter-cluster rating. Figure 10 shows CS470's project collaboration graph before and after pruning. The edges in the new graph became the new intra-cluster edges.

I performed the same analysis on these new clusters as I did on the Louvain clusters. The analysis yielded new correlations, shown in Table 18 and Table 19 for CS357 and CS470, respectively. Study 1's new project collaboration graph yielded no clusters (i.e. no edges satisfied the heuristic) and is omitted from this analysis.

Comparing the alternative clustering method to the Louvain method, I observed that the correlations for both social network clusters and project collaboration clusters increased in both classes. In fact, for CS357 both intra-cluster correlations were statistically significant. In CS470,

only project collaboration showed a statistically significant correlation, but both measures showed stronger correlations for intra-cluster ratings compared to inter-cluster ratings. The alternative clustering method showed stronger support for hypothesis 1 than the original Louvain method.

	Social Network	*Worked on Project*
Intra-cluster	.682*	.777*
Inter-cluster	.310	*

Table 18 – Spearman's Rho for CS357 Graphs with Alternate Clustering

	Social Network	*Worked on Project*
Intra-cluster	.624	.586*
Inter-cluster	.054	-.323

Table 19 – Spearman's Rho for CS470 Graphs with Alternate Clustering

Discussion

Clustering the graphs based on factors that increase familiarity shows value in identifying groups that may yield more accurate ratings of their peers. Analysis of the data for hypothesis 1 showed that ratings given to peers within communities based on some relationship are more accurate than those given between communities. An important implication of this result is that inter-cluster ratings may be holding back intra-cluster ratings when observations are made over the class as a whole. Familiarity does seem to play a role in the accuracy of peer ratings.

Hypothesis 2 pushed further, expecting greater connectivity to associate with higher rating accuracy. Analysis showed, however, that the connectivity of a cluster does not have an effect on the average error of that cluster. The connectivity measure used in this analysis favored smaller clusters, since reaching full connectivity is more feasible in a small cluster. However,

larger clusters may, by their size, indicate greater community and familiarity. This hypothesis should be tested with different measures of connectivity.

An additional analysis showed that clustering method plays a significant role in selecting clusters that give accurate ratings. Louvain clustering works well on dense graphs, but sparse graphs benefit from a more common-sense approach to clustering. The problem with common-sense clustering is scalability, and other clustering methods should be explored and tested.

This analysis addresses the disparity between the sophomores and seniors to an extent. However, the sophomores still generally showed higher accuracy than the seniors. I believe that there are other factors at work that hinder the ability of seniors to rate each other. First, the computer science curriculum at the University of Alabama tends to keep groups of students together for early courses and scatters those students when they begin taking senior-level classes. This phenomenon breaks down student cohesion that shows in the rating clusters of the two classes. Another contributing factor is that senior-level students have had more exposure to the discipline. With this exposure comes knowledge of more subtle aspects of developing. Perhaps these additional aspects made assessing their peers more difficult, whereas the sophomores considered fewer aspects. Both classes showed good predictive ability, but comparing them to each other is difficult.

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

Expertise among software developers is a trait that acknowledges the skill of a developer in performing development tasks. Understanding expertise is important to many people. Researchers would like to be able to control for expertise when evaluating software tools, artifacts, or processes. On the industry side, managers would like to be able to assign their personnel to different tasks by properly allocating the expertise of the developers they have. For educators, understanding expertise would better enable them to identify traits of high performers and transfer those traits to lower performers.

However, software developer expertise is difficult to quantify. Coarse measures such as years of experience are often used to approximate expertise, but these may not be particularly accurate. The central thesis of this research is that quantification and recognition are two different things. Programmers may not be able to identify what traits make one programmer better than another, but are generally confident they can judge a good one from a bad one.

Two studies were undertaken to test this belief. Undergraduate computer science students were asked to rate their peers, and these ratings were compared against grades on programming assignments to determine their accuracy. The results were promising, but it was difficult to draw firm conclusions from the initial analyses performed on the two studies.

With unsatisfactory answers to my research questions after performing the studies, I revisited the data with the goal of finding patterns at a finer granularity than simply observing trends at the class level. My second research question regarding the effect of familiarity guided my efforts to quantify familiarity and analyze it. I decided on graphs to show relationships and used clustering to show what effect these relationships had on the accuracy of peer review. I conclude that familiarity has a significant effect on the accuracy of peer rating data. This conclusion folds back neatly to the first research question which asks whether or not developers can accurately judge their peers. The data shows that peers can evaluate their peers accurately, if attention is paid to how the peer groups are formed.

When identifying these peer groups, the method in which familiarity is established must be chosen with care. The analysis showed that using Louvain clustering did not perform as well as applying more intuitive clustering. Using Louvain clustering is advantageous for large and dense graphs, as intuitive clustering becomes more difficult with larger data sets. Familiarity appears to be a powerful tool for predicting accuracy, so care must be taken when quantifying it.

This work lays the foundation for more work in the area of peer evaluation of software development expertise. A study like this in the future would benefit from larger volumes of data. Targeting large classes is a first step. The study would also benefit from being in an environment where students typically have the same classmates throughout the entire curriculum. Additional factors should be measured. Do gender biases exist in software-related peer rating? How do student perceptions of the learning environment change self, peer, and superior ratings?

Alternative methods should also be explored. How can the rating scales be improved? Is it beneficial to rate on specific development activities instead of general performance? For

familiarity analysis, what clustering methods can be employed and which ones are most effective? These questions drive future research in this area.

When conducting future work from this research, however, the most important consideration to make for this data is the population that generated it. I used undergraduate students. I am confident enough to draw the conclusion that peers can accurately rate each other, but to what extent can students really provide measures of expertise? The sample is limited in that students do not approximate well the full extent to which developers obtain expertise. I posited that experience is not the same as expertise, but it is foolish to assume that experience does not play a role in enhancing one's expertise.

The results shown between the sophomores and seniors in the study deserve particular consideration. The computer science curriculum at the University of Alabama lends itself to greater student cohesion at lower levels. Students often take the earlier courses together, but as they proceed through the senior level courses they define their own course of study by choosing the order in which they take classes. In the algorithms class, some students may have already taken programming languages while others have not. The level of expertise in programming for students is dictated mostly by the courses they have taken. In this sense, cohesion may break down for the seniors in a way that it does not for the sophomores.

Another consideration to make for this sample is that classes are not the same. The analysis assumes some normal distribution of expertise holds for all of the classes, but some students may perform better in certain classes, and some classes overall may consist of better students. In this case, how meaningful are comparisons between classes?

I conducted this study in an academic environment. What do the results mean for industry? There are serious questions about study design just to run this same study in industry.

Particularly, what is a measure of ‘true’ ability in industry? The undergraduate environment is a volatile setting to evaluate expertise. Students are gaining new knowledge and experience at a quick pace. Industry practitioners generally have a strategy for evaluating previous work to select developers. Universities on the other hand are generally more interested in a student’s potential. This is a significant difference when it comes to building populations. This study shows that peer evaluation can be a useful tool, but one must take care when adapting the results to other settings.

REFERENCES

Bastian, M., Heymann, S. and Jacomy, M. 2009. Gephi: An Open Source Software for Exploring and Manipulating Networks. In 3rd International AAAI Conference on Weblogs and Social Media, Anonymous , 361-362.

Blondel, V., Guillaume, J., Lambiotte, R. and Mech, E. 2008. Fast Unfolding of Communities in Large Networks. Journal of Statistical Mechanics: Theory and Experiment P10008.

Boehm, B., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J. and Steece, B. 2000. Software Cost Estimation with COCOMO II. - Prentice Hall.

Bryant, S. 2005. Rating Expertise in Collaborative Software Development. In 17th Workshop on the Psychology of Programming Interest Group, June, Anonymous Sussex University, , 19-29.

Carver, J. and Basili, V. 2003. Identifying Implicit Process Variables to Support Future Empirical Work. Journal of the Brazilian Computer Society .

Carver, J., Hochstein, L. and Oslin, J. 2009. Identifying Programmer Ability Using Peer Evaluation: An Exploratory Study. - .

Carver, J.C., Nagappan, N. and Page, A. 2008. The Impact of Educational Background on the Effectiveness of Requirements Inspections: An Empirical Study. IEEE Transactions on Software Engineering 34, 800-812.

Dehnadi, S. and Bornat, R. 2006. The Camel has Two Humps (working title). http://www.cs.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf.

Dyba, T., Kampenes, V. B. and Sjoberg, D. I. K. 2006. A systematic review of statistical power in software engineering experiments. - Information and Software Technology 48: 745-755.

Flachikov, N. and Goldfinch, J. 2000. Student Peer Assessment in Higher Education: A Meta-Analysis Comparing Peer and Teacher Marks. Review of Educational Research, Vol. 70, No. 3, 287-322.

Holzbach, R.L. 1978. Rater bias in performance ratings: Superior, self-, and peer ratings. Journal of Applied Psychology 63, 579-588.

Höst, M., Regnell, B. and Wohlin, C. 2000. Using students as subjects-a comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering 5, 201-14.

IEEE 1992. IEEE Std 1045-1992, IEEE standard for software productivity metrics. - IEEE.

Krosnick, J. A. 1999. Survey Research. - Annual Review of Psychology 50: 537-67.

Kruger, J. and Dunning, D. 1999. Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments. - Journal of Personality and Social Psychology 77: 1121-1134.

Love, K.G. 1981. Comparison of peer assessment methods: Reliability, validity, friendship bias, and user reaction. Journal of Applied Psychology 66, 451-457.

Lung, J., Aranda, J., Easterbrook, S.M. and Wilson, G.V. 2008. On the difficulty of replicating human subjects studies in software engineering. In 30th International Conference on Software Engineering (ICSE), Leipzig, Germany, Anonymous ACM, New York, NY, USA, 191-200.

Magin, D. 2001. Reciprocity as a Source of Bias in Multiple Peer Assessment of Group Work. Studies in Higher Education 26. 53-63.

Poundston, W. 2004. How Would You Move Mount Fuji? Microsoft's Cult of the Puzzle - How the Word's Smartest Companies Select the Most Creative Thinkers. - Little, Brown and Company.

Prechelt, L. 1999. The 28:1 Grant/Sackman Legend is Misleading, or: How Large is Interpersonal Variation Really? - .

Sackman, H., Erikson, W.J. and Grant, E.E. 1968. Exploratory experimental studies comparing online and offline programming performance. Communications of the ACM 11.

Sonnentag, S. 1998. Identifying High Performers: Do Peer Nominations Suffer From a Likeability Bias? European Journal of Work and Organizational Psychology, Vol. 7, No. 4, 501-515.

Sonnentag, S., Niessen, C. and Volmer, J. 2006. Expertise in Software Design. - In: Ericsson, K. A., Charness, N., Feltovich, P. J. and Hoffman, R. R. (eds.), The Cambridge Handbook of Expertise and Expert Performance. Cambridge University Press, pp. 373-387.

Von Mayrhauser, A. and Vans, A.M. 1995. Program Understanding: Models and Experiments. Advances in Computers 40, 1-38.

APPENDIX 1

Instructions
Attached is a list of the names of all of the students in this class.
For each student (including yourself), please do the following:

1. Rate the **programming ability** of that student using the following scale:
   - A - Very High
   - B - High
   - C - Average
   - D - Low
   - E - Very Low

2. Rate your **confidence** in your estimate:
   - A - High
   - B - Medium
   - C - Low

3. Have you **taken a class** with this student?
Enter 'Y' for yes or 'N' for no

4. Have you **worked on a project** with this student?
Enter 'Y' for yes or 'N' for no

5. Indicate who you are by placing an "x" next to your name

NOTE: If you do not know the student at all (i.e. never took a class with them and never worked on a project with them), please leave the line blank.

Example:

Me	Student	Programming ability	Confidence Estimate	Taken class with?	Worked on project with?
	Joe Blow	C	C	Y	Y
	Jane Doe	A	A	Y	N
	Bob Smith				
X	Stacy Hancock	B	A		
	Michael Wolton	E	B	N	Y

APPENDIX 2

Instructions
Attached is a list of the names of all of the students in this class.
For each student (including yourself), please do the following:

1. Rate the **programming ability** of that student using the following scale:
    1 - Very Low
    2 - Low
    3 - Average
    4 - High
    5 - Very High

2. Rate your **confidence** in your estimate:
    1 - Low
    2 - Medium
    3 - High

3. Have you **worked on a project** with this student?
Enter 'Y' for yes or 'N' for no

4. In your opinion is this student a **better or worse** programmer than you?

5. Do you follow this student on any type of **social network**?
Enter 'Y' for yes or 'N' for no

6. If you were creating a project team, how likely is it that you would **choose this student for the team?**
    1 - Very Low
    2 - Low
    3 - Average
    4 - High
    5 - Very High

7. Indicate who you are by placing an "x" next to your name

8. Identify the 10 classmates that you know best. At the end of the survey, there are some blank lines, please put these ten students in order from the best programmer to the worst, with 1 being the best and 10 being the worst.

NOTE: If you do not know the student at all (i.e. never took a class with them and never worked on a project with them), please leave the line blank.
Example:

Me	Student	Programming ability	Confidence	Worked on project	Better or Worse than me	Social Network?	Choose for Team?
	Joe Blow	3	3	Y	worse	yes	4
	Jane Doe	5	1	N	better	No	2
	Bob Smith						
X	Stacy Hancock	4	1				
	Michael Wolton	5	2	Y	better	No	5