

# Building Blocks for a Better UCR: Effects of Student Collaboration and Group Size on Learning Retention

Nelson Perez    Conley Read    Ryan Rusich

University of California, Riverside  
Computer Science  
{nperez, cread, rusich}@cs.ucr.edu

## Abstract

The Department of Computer Science at the University of California, Riverside is a quickly growing, high quality research and teaching institution. In this paper, we identify six benchmark areas that we either do well or poorly. We identify these areas through comparison with core proficiency areas at best-of-field institutions. We recommend that our findings, particularly with respect to suggested improvements in low performing core areas, be integrated into an iterative improvement process executed by the department's Instruction Committee.

During our initial study and research, we have found that high ranking institutions put a premium on student collaboration, group leadership and student projects. At UCSD, undergraduate assistants support the teaching assistant role in engineering laboratory sessions. At UCI undergraduate computer science students are required to take a lower division software engineering course which encompasses the general principles of large scale software design and production. Lab work consists of a term project that illustrates the concepts acquired. Our focus in this three week study on the relationships of student collaboration, group size, and learning retention, is on undergraduate education at the lower and upper division level in the department of computer science and engineering. Initial findings show that students working alone are less likely to complete lab work on time, learn less and lack group work skills. After being encouraged to collaborate and supervised, students were more likely to complete lab on time and continued to collaborate effectively when unsupervised.

## 1 Introduction

The University of California, Riverside is a diverse and growing university with strong programs in

Computer Science and Engineering. The Computer Science and Engineering Department at UCR has a drive for excellence and a work ethic that puts us on par with the best research institutions in the world. UCR is also one of the youngest institutions within the University of California. Despite the high quality of our research and work ethic of our faculty and students, our institution is overshadowed, at least domestically, by other Universities in the United States and within the University of California system.

We have identified a subset of six benchmarks that we believe indicate the required growth potential for an institution of our expected quality. These benchmarks are as follows: curriculum, funding, student labs and teaching techniques, student projects, entrepreneurship opportunities, and campus or college marketing. We noted immediately a few important factors that influence the benchmarks we have listed above. Undergraduate curriculum is almost always influenced by the presence of professional schools on the same campus. For example, the medical school at the University of California Los Angeles influences the bioinformatics and medical informatics courses. Campus and college marketing can also help guide an institution to higher reputation and stature. Marketing to a population of lesser reputation seems to be bad, while "marketing up" increases reputation. In short, picking the right competition is important. Finally, funding is as important as curriculum for a research university. Without faculty that retain important funding, student research opportunities and projects suffer. Research announcements are key to marketing and institution stature.

## 2 Inspiration

As graduate students within the Department of Computer Science, we see promise now and in the future for the Department to rise in stature and reputation to succeed in competition with the highest ranked institutions of engineering for the best students,

whom we can offer the best opportunities and education. We draw our knowledge for comparison from institutions that we have identified as the best in the field. These institutions are University of California Los Angeles, University of California San Diego and other regional and same-system universities. These institutions have excelled to different degrees in gaining status as top flight engineering schools and campuses as a whole. We investigate the efforts they have engaged in and the metrics that they have applied to themselves to reach this pinnacle that we may surpass them in this effort.

## **3 What UCR Does Well**

### **3.1 Undergraduate Curriculum**

After careful consideration of the curriculums at UCI and UCLA, a more thorough understanding of UC Riverside's strengths became evident. First all three universities instill in their computer science students a strong background curriculum in mathematics and the physical sciences. However, UCR actually required their students to participate in an upper division linear algebra and discrete math course where as UCLA and UCI incorporate lower division offerings of these courses. In fact UCR is the only one of the four campuses that has as a requirement for degree the Bachelor's, completion of any upper division mathematics courses.

UCSD does cross list their Algorithm Analysis and Intro to Theory of Computation classes as math courses, this we do not consider as traditional upper division math offerings. Additionally UCR and UCSD requires that their students take an upper division statistics course, as is common in most computer science curriculums, tailored for engineering and computer science students whereas UCI a UCLA require a lower division course. UCR, UCLA, and UCSD require a year in the study of Physics with laboratories designed for engineering students. UCI allows their students to make a choice of either the Physics series designed for Life Science majors, a year of Chemistry, Biology, or the equivalent of the Physics series required by Los Angeles and Riverside.

A second and more fundamental area of strength for UCR is the advent of labs for every computer science course offered at UCR. Students are given the opportunity in a lab environment complete with workstations and staff, to hone their programming

skills, and practice new techniques as they are being taught them. At UCLA, only the hardware and architecture class have labs, discussions are required of all CS classes. At UCI there labs are present at the lower division level, and become more abbreviated as the curriculum advances with many upper division core courses such as Operating Systems having no lab, and Compiler Design have one hour per week. These two campuses seem to provide additional instruction outside of lecture via discussions. UCSD is unique in that they appear to have fairly consistent lab inclusion, but in some instances lecture is supplemented by discussions, review sessions, and further instructional meetings in conjunction with traditional lecture.

Lastly only UCR and UCSD allow their undergraduates the choice of six technical electives; we believe that this gives these two universities marked advantage over their Irvine and Los Angeles counterparts. This freedom in the selection process for an individual's curriculum will better allow them to shape their education into a from which they can leverage for the types of positions in industry that is most in tune with their personal interests in the field. UCLA's program is fairly strict in that the students are only allowed to select two technical electives of their choice. UCI allows five picks, but only three of those are project classes. This reality puts UCR's students in a more experienced position upon graduation.

### **3.2 Competition & Community**

Ryan Rusich in his CS12 course hosted a laboratory session in which the class was divided into two teams of eight and 9 students respectively. A project manager for each group was chosen by Mr. Rusich based on insider knowledge of the two leaders' abilities. These two leaders were charged with the task of dividing their individual team members into three sub-groups that would develop modules of a Tic-Tac-Toe game with a graphical user interface. The programming exercise would incorporate certain core concepts covered in the class such as object-oriented programming including abstract data types, and inheritance, as well as a simple graphics package API.

The members of each subgroup would work collectively on a portion of the larger program with the prior knowledge that after some development time, an attempt to incorporate their work with the other team members offering would be undertaken with the ultimate goal of have a working game that would allow a user to play the computer and have a winner

determined as per the rules of the game. An added caveat was that the project managers were not allowed to personally manipulate any code or machine. As an added incentive, a case of soda was offered to the team that was able to complete the game on deadline, which in this case was the end of three hour lab period. Out of fairness pizza was offered as a reward should both teams succeed.

The first third of the exercise was spent letting the individual teams get use to each and their newly appointed project managers. Some pitfalls that might contribute to a failed attempt were pointed out to both teams by Mr. Rusich. After sixty minutes transpired a detailed code review of all work done so far was initiated by Mr. Rusich at which point he ascertained whether or not significant progress was being made. At ninety minutes into the event each of the two teams were instructed to select a single workstation to which all the code would be collected to be integrated into a single software package for further development. At this point the direct instruction from lab staff was discontinued to allow the teams to better agree on their particular strategy for completion of the task.

### **3.3 Student Projects**

Student Research projects open a multitude of opportunities for students at the university. Mr. Perez reports on the students projects native to the UCR CS 153 Operating Systems Course. Expanded research can increase student morale.

CS153 Operating Systems is one of the key undergraduate courses in computer science. Projects in this class are usually challenging and may be to demanding for a single student. This is the reason students are allowed to work in pairs. Due to the nature of the projects, I argue that pair programming may not be as effective in this course as it can be in other courses.

As an example, I will discuss how useful pair programming can be on the context of two representative programming assignments of CS153: the shell and the scheduler. Note that I did both projects when I was an undergraduate and presently as a TA as well. CS153 is a course in which the benefit of pair programming varies according to the nature of the assignment. It can be useful in some cases and so much in others.

#### **3.3.1 Project 1 – The Shell Program**

This assignment consists on developing from scratch a Unix-like command line interpreter. This assignment can be done in an incremental fashion in which features are added one after the other. Pair programming is a nice strategy to develop it as students can take turns on implementing distinct features. Furthermore, one student can implement a feature and the other can focus on testing the new feature.

#### **3.3.2 Project 2 - The Scheduler**

This assignment is possibly one of the most challenging projects in the undergraduate course. It involves a few more than a hundred lines of code. However, the challenge comes from understanding code that was written by fellow students. For the most part students write this project from scratch but add their code to the existing Linux kernel files. In addition, students must write a test program to verify that the new scheduling is working properly.

The best strategy to attack this assignment is for students to split the work. One student should focus on implementing the policy and the other one in designing the architecture of the test program. These two tasks are not completely independent as the developer of the test program needs to know a few details about the scheduler. Two students alternate implementing the schedule policy seems not to be the best approach since it requires few lines of code. Similarly, the test program requires a few more than a hundred lines and so taking turns is not the best approach either. In conclusion, for this project pair programming is not the best approach. The best way to do it is just by dividing the work.

## **4 Improvement Areas for UCR**

### **4.1 Student Collaboration in Lab**

Student collaboration in laboratory work at UCR has been largely obstructed by a mislead belief at the lower division undergraduate level that any collaboration is academic dishonesty. Students trained in this method avoid collaboration to remain safe from possibility of cheating accusations and other infractions. When group collaboration and leadership skills are not exercised, students appear not to use them even in the environment where they are acceptable.

Collaboration and group problem solving are highly valued skills at the top ranking institutions that we studied. At the Massachusetts Institute of Technology, collaborative research and laboratory work are encouraged at the undergraduate level. We believe that this early, important focus on collaborative skills increases the quality and success rate of student projects, and eases the transition from research to entrepreneurship.

## **4.2 Marketing of UCR CS**

We see the opportunity for UCR to differentiate itself in the public eye from its regional counterparts in the area of media relations. None of the four Southern California University of California campuses have what could be called a dominant presence on local media, whether it is in print, broadcast, or online form. More importantly Irvine, Los Angeles and Riverside share the same primary local television broadcast areas. Often perception becomes reality. Conveying to the community at large the significant and exciting events that are occurring at an institution such as UCR, and particularly in should be a top priority.

These events should strive to convince the observer of segments detailing our unique and progressive approach to the fields of Computer Science and Engineering. Whether a tour of a new facility such as Engineering II or drawing attention to current research that attempts to better understand current technology trends and phenomenon, like Karagiannis and Faloutsos work on P2P file sharing, a coordinated relationship with the media could benefit both the public reputation of UCR, and our ability to recruit students, and to some extent faculty talent. It is entirely possible that via committee interaction, the department might be able to coordinate and plan events that would show our campus in the most positive light to as many individuals as possible.

## **4.3 Multidisciplinary Research**

At UCLA, the computer science department and the medical school have a fruitful relationship. People from both departments engage in multidisciplinary research. From this cooperation, researchers in the CS department can not only be funded by traditional sources like NSF or DARPA but also by the NIH (National Institute of Health). Furthermore, new, hybrid research laboratories are the offspring of the interaction. The best example of this is the Medical Imaging Informatics Group. This group consists of researchers from both, computer science and the

medical sciences. The group is submerged into multidisciplinary research. UCR could learn from such the UCLA model. Multidisciplinary research also increases student entrepreneurship. This keeps students local non-academic scheduling period and build opportunity for new graduates.

## **5 Problem Area Identified**

As we have identified earlier in this paper, group collaboration has not been an important part of lower division undergraduate education in the computer science department at UCR. We believe that the laboratory subset of each course or elective is the ideal place to exercise these important skills. Recently, so-called XP, or extreme programming, where student programmers work in pairs on programming projects and assignments, has been implemented in undergraduate courses for group project work. While this is a definite start, the limited implementation and continued departure from group collaboration at the undergraduate level problematic. We suggest remedy and evaluate the effect of different group sizes on learning in collaboration.

A second and more insipient threat from an institution's predominant focus on individual measurement is academic dishonesty. Regularly at UCR computer science students are encountered sharing code and submitting an obvious collaborative effort as their own work. University and departmental policy allows for punitive action to be taken, but in the extreme case where the malicious activity is rampant; professors, instructors, and teaching assistants find a disproportioned amount of their time being spent investigating, identifying, and prosecuting academic fraud rather than preparing curriculum, or offering assistance to well intentioned and diligent students. Our suggestion is that much of this could be alleviated by an increased emphasis on teamwork. In other words allowing some sanctioned collaboration on homework or programming assignments, mixed in with individual metrics such as exams, quizzes.

## **6 Results of Team Approach**

Allied with our discovery that high ranking institutions for computer science focus on student collaboration skills at the undergraduate level, we used each of our lab sections to evaluate the effect of different group sizes on learning in collaboration.

First, we describe our methodology. Ryan Rusich tracked the effects of student collaboration in groups

of five or more. Conley Read evaluated student performance and learning retention in groups of three. Nelson Perez followed the effect of traditional XP pair programming in his lab. The tracking methods applied in each case were standardized as much as possible. Unfortunately, in some cases we had to work with the requirements of the class as already defined, disallowing some flexibility that may have improved the quality of results or the tracking methods. For example, we would have liked to implement a variety of collaboration types in each lab, but we were limited by the requirement that systems programming in Computer Science 153 occur in the traditional extreme programming groups.

An interesting set of events occurred in Ryan Rusich's CS12 lab. First without explicit direction from staff the workstations selected by the two teams happened to have an orientation where the two keyboard operators from each team were directly facing each other while their respective teams stared intently over their shoulders occasionally glancing directly ahead at their opponents. Initial testing of both teams programs exposed an onslaught of software bugs and integration issues. After another 30 minutes to allow for the teams to settle down into a collective calm, Mr. Rusich investigated individually with each team overt the problems that they were having. He suggested to both teams that whenever a new problem area of their code was discovered ask within the group which of their members was responsible for that section and immediately get that individual or set of individuals' input on a strategy to resolve their predicament.

Ultimately the resolution over the next and final hour of the competition, we advise the reader that the contest was designed to allow for both teams to be successful. Results were strikingly different for the two teams. One team seemed to stand idle why the keyboardist frantically tried to make headway on the slew of bugs and difficulties that arose. The leader became distant and idle, while two of the more prodigious members of that same group chose to acquire seating at the back of the group and socialize, or primarily address non-task issues. The second team's demeanor was markedly different. At times their confidence was shaken as their responses to a particular dilemma were met with sustained or even increased adversity.

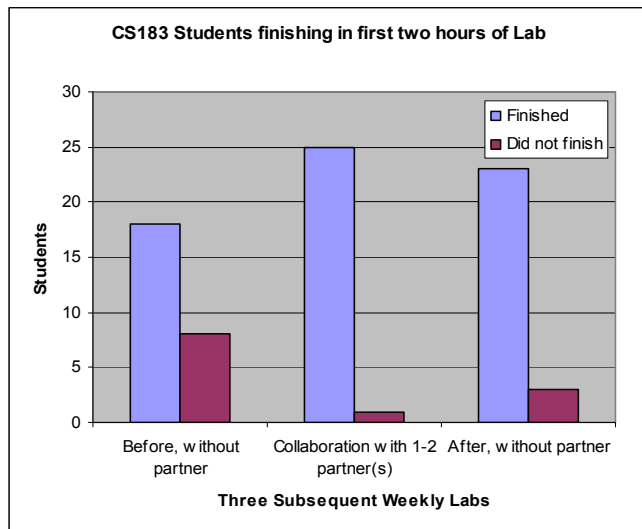
However with each small victory resulting in a more general optimistic outlook, their confidence was beginning to gain momentum. At one point the entire

team hovered over a single terminal and roared out loud, causing Mr. Rusich to investigate whether or not they had completed the task. They had not, but with only ten minutes remaining it was entirely possible that with one final push and an ample amount of good fortune they might just make it. Let the record show that Mr. Rusich had not made special effort to not prejudge either of the two teams ability to achieve the ultimate goal, of a working software game program, but with time running out he had substantial doubt that either team would finish.

Five minutes passed, and as the team that was experiencing difficulties was performing a premature postmortem of their misfortunes with Mr. Rusich, a second and more decibels driven cheer erupted from the team with better fortune. Immediately students were seen giving high fives and beaming with joy as they had gotten their program to run. To be official, lab staff tested the winning program, and upon success congratulated them on a job well done. The ecstatic and proud looks on their faces were some what dampened by the mood of the team that had failed in their attempt, but more importantly the students seemed to take overall pride in the accomplishment of this particular lab section in being the first of six lab sessions over two quarters to complete the goal in the allotted time.

In conclusion, the experience was outwardly a success, but inwardly the students gained the most. The winning team exuded all of the traits and qualities of true teamwork: passion, willpower, care for the collective good, and selflessness, traits that are important to software professionals and people of all walks of life. It is doubtful that the lessons learned from such a short intense experience can be duplicated through traditional pedagogical methods. With this in mind we would like to offer that some consideration to events such as these be given in the future by the department when possible. Of course we understand that this should never interfere or impede on more important or necessarily rigorous academic activity.

## 7 Graph of Results



**Figure 1. : We compare lab completion rates during the first two hours of three subsequent laboratory sessions of Computer Science 183, an introductory systems administration class, which confronts upper division students with technical, security, and ethics challenges.**

## 8 Conclusions

Through our research, evaluation, and examination of common standards and their implementation at the University of California, Riverside, we believe that our campus is on the right track. A continued, iterative improvement process including oversight and implementation will insure that we stay headed in the right direction.

As we have noted before, the University is a young campus and does not yet enjoy the stature that it's older, possibly less innovative and agile siblings in the UC system have the benefit of, currently.

It is key that we as an institution track and identify the mistakes and missteps of other institutions, so that we may avoid common pitfalls and integrate into our improvement process the knowledge of learning institutions from every tier of education. It is obvious that one unique drive toward excellence is the requirement of competition and cooperation that collaboration requires at even the departmental level within a University. For example, at UCLA, the presence of a medical informatics degree in the school of medicine drives the department of computer science to support the required courses, increasing knowledge capital and curriculum depth within the college. This, in turn, gives capable students more opportunities and

pays significant dividends on the initial investment as students matriculate.

## References

- [1] "CoVis Philosophy. Learning through Collaboration and Visualization: Communities of Practice." <http://www.covis.nwu.edu/info/philosophy/communities-of-practice.html> 1998.
- [2] "CSE @ UCR: Home Page." <http://www.cs.ucr.edu/index.php/main/main/> 2005.
- [3] "UCSD Computer Science & Engineering Courses." <http://www.ucsd.edu/catalog/CSEcour.html> 2005.
- [4] "UCLA Computer Science Courses." <http://www.registrar.ucla.edu/catalog/catalog-200.htm> 2005.
- [5] "UCI Information and Computer Science Courses." <http://www.editor.uci.edu/04-05/ics/ics.2.htm> 2005.