Using Cloud Services To Improve Software Engineering Education for Distributed Application Development

Jorge Edison Lascano[1,2], Stephen W. Clyde[1]
[1]Computer Science Department, Utah State University, Logan, Utah, USA
[2]Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE, Sangolquí, Ecuador
email: edison_lascano@yahoo.com, Stephen.Clyde@usu.edu

***Abstract*—Software-engineering education should help students improve other development skills besides design and coding. These skills, referred to here as A2R (Analysis to Reuse), include analysis, technology evaluation, prototyping, testing, and reuse. The need for improved A2R skills is particularly pronounced in advanced areas like distributed application development. Hands-on programming assignments can be an important means for improving A2R skills, but only if they focus on the right details. This paper presents a case study of programming assignments offered in a graduate-level class on distributed application development, where the assignments required the students to use cloud services and programming tools that were heretofore unfamiliar to the students. Direct observation by the instructor and a post-class survey provided evidence that the assignments did in fact help students improve their A2R skills. The post-class survey also yielded some interesting insights about the potential impact of well-designed programming assignments, which in term led to ideas for future research.**

Keywords-computer science education; software-engineering education; cloud computing; virtual environments; distributed systems.

I. INTRODUCTION

Imagine yourself at a worktable with four or five of your peers. In the center of the table is a pile of seemingly random objects, including two dozen sheets of paper, paper clips, a small roll of tape, pins, and several small wooden sticks. A quick glance around the room reveals a dozen other groups just like yours with similar piles in front of them. An individual, who is introduced as your customer, stands at the front of the room and says that you have 30 minutes to build a "great" tower. What do you do first? How do you put all that you know about paper, clips, tape, wooden sticks, etc. into practice to satisfy the customer's request for a tower and do so within 30 minutes?

Such is the typical scene on the first day of class in the undergraduate introductory course on software engineering at Utah State University (USU). In general, all the students have a good working knowledge of objects at their disposal and even some inkling on how they may combine several of them to create new more structural useful objects. Most groups succeed in creating something that stands on its own and roughly resembles a tower within 30 minutes. However, at the end of that time, the customer surprises the students by giving them a few more objects, e.g., more paper and tape, and asks them to take 15 more minutes to make their towers taller or stronger. Many groups fail to do so in the limited allotted time. In fact, about half of them end up destroying their original towers in the attempt.

Afterwards the instructors and students discuss the experience in terms of what worked well for the group, particular difficulties that hindered progress, how the group organized itself, and how they decided on an overall approach. The discussion usually leads to some very interesting comparisons with common aspects of software engineering, such as group work, tool evaluation, prototyping, design patterns, testing, extensibility, reuse, and more. Over the years, one of the authors, who is a long-time instructor for this introductory software engineering course, has observed the following:

1. Virtually no student or group ever asks the customer what a "great" tower means. Most assume that they already know and proceed to build without each researching the requirements.
2. Virtually no student or group ever looks around to see what other groups have done or are doing, evaluates the ideas they see, and then tries to adapt or improve on them.
3. Only a small percentage of the groups try prototyping an idea to explore its characteristics.
4. Only rarely does a group test the properties (e.g., stability or strength) of a component or the whole tower and then try to make modifications to improve those properties.
5. Only a few groups try to establish patterns or "best practices" either in their building processes or the components they create, and then reuse those ideas.

Each of these observations represents a potential engineering pitfall or negative practice that can lead to inefficiency or failure. Software-engineering education needs to help students avoid these and other related pitfalls by connecting theory with best practices in the context of real non-trivial problems [1]. Doing so goes well beyond teaching the "How To's" of a specific technology, like a programming environment. Instead, it requires educators and students alike to address the "How To's" of the overall development process, including:

1. How do we know when we understand the customer's problem sufficiently?
2. How can we benefit from existing technology or from what others have tried in the past?
3. How can we prototype part of a problem or alternative solutions to answer critical questions?

 ISBN: 978-1-61208-438-1

4. How can we test what we build?
5. How can we find good solutions to reoccurring problems and reuse that knowledge?

More concisely, software-engineering education needs to help students make analysis, technology evaluation, prototyping, testing, and reuse an effective and integral part of their development activities [1]. Here, we'll refer to these as Analysis to Reuse (A2R) skills.

The need for better A2R skills is prevalent in every software-engineering domain, but is pronounced in the development of distributed applications. Distributed-application development, or distributed-system development at large, has all of the challenges of traditional software development, plus the complexities introduced by inter-process communications, concurrency, the potential of partial failure, and replication that exist for performance improvements or fault tolerance [2].

Now let us roll our classroom scene forward several years to a graduate software-engineering class that focuses on distributed applications. Students entering in this class have solid foundations in software-engineering fundamentals, programming languages, inter-process communications using sockets, and many other areas of computer science. Yet, they still need to strengthen their A2R skills, especially in the context of distributed applications, and the best way to do that is through hands-on experience [3]. So, from an education perspective, the challenge is to provide realistic and engaging assignments that will strengthen the A2R skills and are doable within the allotted time.

Because distributed applications are relatively complex [4] by their very nature, there are two negative tendencies for program assignments in this area: a) abstracting away too many interesting aspects of the problem and b) getting bogged down with unnecessary application-domain details.

The first tendency is very common in advanced CS courses, because simpler assignments are more manageable, teachable, and easier to fit within a given allotted time. Advanced courses usually have to operate within same time constraints as introductory courses. Even though, they are more complex, it is essential that advanced assignments include reasonable limits on the expected time and effort [4]. Simplicity in their design is a necessity and by itself is not a problem. Focusing on the wrong details and abstracting away all interesting parts of the problem, however, is a serious real pitfall. For example, scalability is a real and very common aspect of most distributed applications [2]. Even though removing scalability requirements would simplify an assignment, it would rob the students of a valuable opportunity to improve A2R skills in a relevant area.

The second tendency is to allow an assignment to get bogged down in application-domain details, shifting focus away from the learning objectives. Assignments in advanced courses, like distributed-application development, work best if they are grounded in a meaningful real-world domain. However, most distributed applications and their domains are relatively complex. If not careful, an instructor could easily use all available time explaining the sample application domain, instead about the core course topics. Keeping assignments focused on a small set of functional requirements that require minimal application-domain knowledge, is essential to making sure that they are doable within time limits and achieve the learning objectives.

This paper describes a case study of programming assignments conducted in an advanced software-engineering class on distributed-application development, where all of the assignments required students to use cloud resources for their execution environment. The hoped-for result was that the assignments provided students significant opportunities to improve their A2R skills, while introducing them to new concepts and development tools. Section 2 describes the course's programming assignments in terms of their learning objectives, the application domains that act as backdrops, and their requirements. Section 2 also explains the tools and technologies introduced for each assignment. Section 3 summarizes the instructor's observations made throughout the semester and assignment design learnings. To evaluate the effectiveness of the assignments, we conducted a post-class survey. Section 4 describes this survey and presents the resulting raw data. Since the class was a second-year graduate class, the enrollment was small. So, we cannot make many generalizations from the survey data. Nevertheless, they do lead us to some interesting insights. We share those insights in Section 4.B. Section 5 explores related work in software-engineering education using cloud resources and hands-on learning. Finally, Section 6 provides conclusions, along with ideas for future research that could further advance software-engineering education relative to A2R skill development.

II. Programming Assignment For a Distributed Application Development Course

CS 6200 at USU is a second-year graduate course in software engineering that focuses on the development of distributed applications. Its prerequisite, CS 5200, provides students with a strong foundation in inter-process communication, protocols, concurrency, and communication subsystems. CS 5200 is also a programming intensive course, which means that students who successfully complete it have confidence in their ability to implement non-trivial software systems. The overall learning objectives for CS 6200 are as follows:

- Master theoretical elements of distributed computing, including: models of computation and state, logical time, vector timestamps, concurrency controls, and deadlock;
- Become familiar with the provisioning and use of virtual computational and storage resources in a cloud environment;
- Become familiar with cloud-based tools for processing large amounts of data efficiently; and
- Become familiar with distributed transactions and resource replication.

For the Spring 2015 semester, the homework was broken down into five assignments, each lasting two to three weeks.

 ISBN: 978-1-61208-438-1

A. Assignments 1 & 2 – Disease Tracking System

For the first two assignments, the student implemented a set of processes that worked together to form a disease tracking and outbreak monitoring system. They had to deploy multiple processes on EC2 instances within Amazon Web Services (AWS) cloud. The first type of processes were simulations of Electronic Medical Records Systems (EMR's) that randomly generated notifications of diagnoses for infectious diseases, like influenza. The EMR's sent these disease notifications to Health District Systems (HDS's), which collated diagnoses and then sent periodic disease counts to Disease Outbreak Analyzers (DOA's). Each DOA monitored outbreaks for a single type of disease. See Figure 1. The specific learning objectives for these two assignments included:

- Review inter-process communications;
- Become familiar with vector timestamps and how they behave in a distributed system under varied conditions;
- Become familiar with setting up and using computational resources in a cloud, e.g., AWS; and
- Become familiar with setting up a simple name service.

The students were asked to learn and use Node.js as the primary programming framework [5][6]. Because Node.js was new to all the students, some class time was dedicated to teaching Node.js, but only enough to get them started. Their unfamiliarity with Node.js was also the reason this first system was split into two assignments. They built and tested approximately half of the functionality in the first assignment and the remainder in the second.

To deploy their systems to EC2 instances on AWS, the students had to learn about security on AWS, create security keys, and setup their own user accounts using Amazon's Identity and Access Management (IAM). They also had to setup and learn the AWS's command-line language interface (AWSCLI), so they could automate the deployment and launching of their systems.

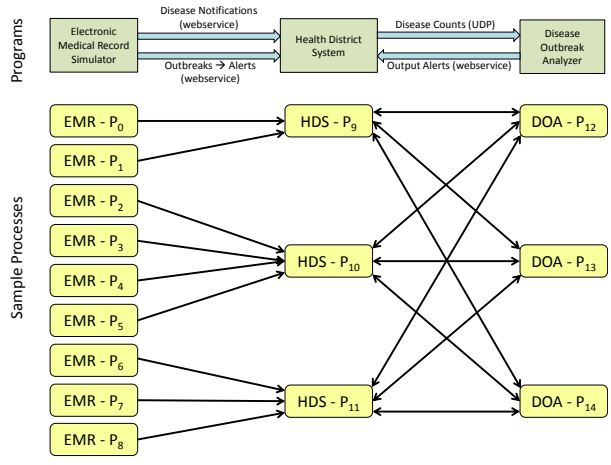

Figure 1. Programs built as part of Assignments 1 & 2, plus an illustration of sample processes.

B. Assignment 3 – Twitter Feed Analysis

In this assignment, the students explored how to process big data using MapReduce on AWS and how to configure cloud resources using AWS's Cloud Formation tools. Specifically, they were to capture tweets through Twitter's API and then analyze them for positive or negative sentiment relative to some key phrase, like "health care". The learning objectives for this assignment were as follows:

- Become familiar with setting up and using MapReduce with a cloud-based distributed file system;
- Become familiar with tools for provisioning collections of resources that are needed for a distributed system; and
- Explore the types of problems that are well suited for a MapReduce solution

To complete this assignment, students setup and learned how to use AWS's S3, MapReduce, and Cloud Formation services. Some also used this assignment to learn about a Node.js module for working directly with AWS; while others strengthened their knowledge of AWSCLI.

C. Assignment 4 – Distributed Election

In this assignment, the students implemented a distributed system consisting of dozens of processes that shared access to common data files, which were collectively treated as one large shared resource, like a database. One of the processes played the role of Resource Manager (RM) and accessed the common data files in response to requests from the other processes. If RM died, then the other processes had to detect that failure and elect one of the remaining processes to be the new RM seamlessly. The learning objectives for this assignment were:

- Master at least one distributed election algorithm;
- Master the concept of resource managers for controlling access to share resources; and
- Become more familiar with tools for provisioning collections of resources in a cloud.

To complete this assignment, we allowed students to use any of the tools they had learned thus far, but they had to deploy their systems to multiple EC2 instances and demonstrate that the system would elect a new RM if the current one was stopped. They had to show that the system has as a whole, lost no work.

D. Assignment 5 – Distributed Transactions

In this assignment, the students had to build a simple transaction management system with locking capabilities. Like Assignment 4, this system had to support multiple concurrent worker processes, but went a step further in requiring multiple shared resources and multiple concurrent RM's. Each RM had to keep track of a single resource and support lock, read, write, and unlock operations on that resource. The system also had to include a transaction manager that supported starting, committing, and aborting of transactions. Assignment 5's learning objectives included:

- Become familiar with locking; and

 ISBN: 978-1-61208-438-1

- Become familiar with transaction management in a distributed system.

Like Assignment 4, the students could use any of the tools that they learned to this point in completing Assignment 5.

III. INSTRUCTOR OBSERVATIONS

Seven students took CS 6200 in the Spring 2015 semester: 5 who were registered for credit and 2 who audited the class. It's impossible to recap all that took place during the semester, but we summarize a few observations prior to presenting the post-class survey to help set the stage for the survey and our conclusions.

First, we observed that all of seven students started the class with roughly equivalent software-engineering backgrounds and programming skills, even through they were not all seeking the same degree nor did they have the same programs of study. None of the students had used Node.js before and only one had any exposure to cloud computing, and that was only a light exposure.

Second, we observed that requiring students to setup and managing their own cloud resources not only helped them with core concepts and development skills, but it also allowed them to improve their A2R skills relative to figuring out what the most important requirements were, tool evaluation, and testing. For example, in the first two assignments, the students had to deploy their system to EC2 instances. For most of the students, this was the first time deploying something that they built to an execution environment different from their development environment, along an execution environment consisting of multiple virtual machines. It opened their eyes to new challenges, such as firewall issues, file permissions, and missing dependencies. Time was made available in every class period for them to talk about the challenges that they were facing and get ideas from other students or the instructor about how to address those challenges. Similar discussion also took place on an online forum. By the end of Assignment 2, the classroom and online discussions showed that the students had stepped up their efforts to understand the assignment requirements, evaluate the tools available to them, and test their work.

Even though the purposes of Assignments 4 and 5 were considerably different from the first three, they possessed some of the same challenges, like resource name resolution and deployment into a cloud environment. It was encouraging to see that the students solved these problems by adapting techniques used in the earlier assignments and improving upon them – evidence of them practicing A2R skills.

We were happy to see that the students learned some unexpected, but very relevant lessons. For example, one student stored his access keys in a text file and committed that file to a public Git repository. It wasn't long before someone hacked his AWS account. Amazon and the student caught the problem relatively quickly and simultaneously, but not before the hacker had used over $600 of resources. He ended up taking extra time learning more about security from unauthorized use. Thankful, Amazon worked with him to recover the expenses, so he did not have to pay for the lost out of pocket. Still, it proved to be a valuable learning experience that he will not forget.

With respect to the selected cloud AWS, we observed that it provided a mature and full-featured set of services for the students to learn from. In some areas, AWS's learning curve was steeper than necessary, but with supplementary examples and good discussions, it was manageable. From an education perspective, a good thing about AWS is that it has features in three main categories: Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS) [2].

One negative experience with AWS occurred during Assignment 3, which depended on an Amazon-provided template for setting up a MapReduce cluster. That template was changed by its authors in the middle of assignment, causing several of the students not to complete all of the requirements. To avoid this problem in the future, the instructors will make private copies of public or external resources, so changes to them will not affect assignments in progress.

A. Assignment Design Learning

When instructors design assignments oriented to networking or distributed applications, they need to consider distribution concepts, but at the same time bear in mind the limitations for the students' capabilities and hardware environment. Before cloud resources became available, this typically consisted of one computer [7] or small number of computers on a local area network (LAN) in a school lab. Assignments that work well on one computer or a LAN may not allow the students to gain appreciation for more realistic networking challenges, performance issues, and reliability problems [8]. With cloud resources, assignments can now be designed having a broad range of resources in mind, while still considering good software-engineering practices for analysis, technology evaluation, testing, deployment and even reuse.

IV. POST-CLASS SURVEY

To assess the value of the programming assignments for CS 6200, we designed a post-class survey and conducted that survey with two populations: students registered in CS 6200 for credit and students who just audited the class. Those registered for credit had to complete all of the assignments to receive a grade; those just auditing the class did not. In fact, it is important to note that none of the second group completed any assignment.

A. Survey design

We organized the survey into two parts. The first part asked students to rate their knowledge and skills in areas related to the course and the assignments, as they were before the class started, using a 1-to-5 scale. The second part asked them to do the same relative to the end of class. *Table*

 ISBN: 978-1-61208-438-1

and *Table* list the concepts (knowledge areas) and skills respectively covered in both parts of the survey. The survey used a *Matrix Table* format, with the concepts and skills as rows and possible ratings as columns. See Figure 2 for partial view of the survey instrument for Part 1.

The difference between each individual's answers to corresponding questions from two parts provides a glimpse of that person's perceived change in knowledge or skill levels as a consequence of the course.

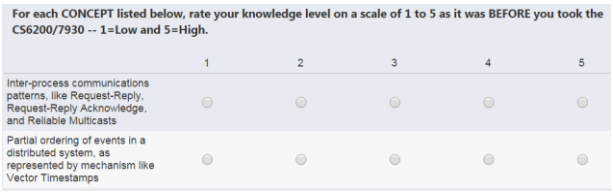

Figure 2. Partial view of the survey.

We could have administered a pre-class survey similar to the first part, but considerable differences in each student's personal rating scheme would likely have evolved over the semester, making it difficult to assess perceived change. We could have also administered pre and post exams to measure their proficiency objectively, but there was no common knowledge basis for a pre exam. So, the study would have degenerated into the interpretation of just post exam results.

TABLE I. KNOWLEDGE SURVEY QUESTIONS.

No	Knowledge/Concept	Acronym
Q1.1	Inter-process communications patterns, like Request-Reply, Request-Reply Acknowledge, and Reliable Multicasts	IPC
Q1.2	Partial ordering of events in a distributed system, as represented by mechanism like Vector Timestamps	VTS
Q1.3	Message serialization/deserialization	S/D
Q1.4	Intra-process concurrency	IntraPC
Q1.5	Computation resources in a cloud-computing environment, such as AWS	AWS
Q1.6	Namespaces, name services, and name resolution	NS
Q1.7	Deployment, execution and testing techniques in a distributed environment	Deploy
Q1.8	Deployment, execution and testing techniques in the cloud.	Testing
Q1.9	Distributed election algorithms	DEA
Q1.10	Resource managers	RM
Q1.11	Fault tolerance in a distributed environment.	FT
Q1.12	Tools for provisioning collection of resources needed for a distributed system.	Tools
Q1.13	Cloud Computing resources	CCR
Q1.14	Infrastructure as a Service (IaaS)	IaaS
Q1.15	Platform as a Service (PaaS)	PaaS
Q1.16	Inter-process concurrency	InterPC

TABLE II. SKILLS SURVEY QUESTIONS.

No	Skill	Acronym
Q2.1	AWS Users and key pairs (Identity and Access management -IAM)	AWS-IAM
Q2.2	AWS Virtual PC Instances (EC2)	AWS-EC2
Q2.3	AWS Storage (S3, EBS)	AWS-S3,EBS
Q2.4	AWS-CLI (Command Line Interface)	AWS-CLI
Q2.5	AWS SDK (Software Development Kit)	AWS-SDK
Q2.6	Managing instances in AWS: creating/launching, starting, stopping, terminating	EC2-Instances
Q2.7	AWS Billing	AWS-Billing
Q2.8	Using Node.js to Develop Distributed Systems	Node.js_DS
Q2.9	Using Node.js to deploy and run Distributed Systems in the cloud	Node.js_Cloud
Q2.10	Designing and developing TCP/UDP/Web Services-based systems with Node.js	Node.js_C/S
Q2.11	Writing scripts to Deploy/execute applications in distributed environments	DS_Scripts
Q2.12	Designing and Developing Resource Managers	RM_DD
Q2.13	Designing and Developing Distributed Election Algorithms	DEA_DD

B. Survey Results

All seven students completed both parts of the survey. Figures 3 and 4 show averages of the students' raw estimates of their knowledge and skill levels for before and after class. The blue lines represent the levels before and the red lines after. The (a) graphs are for the first population, namely the students who registered for credit and the (b) graphs are for the auditing students. Figure 5 shows the average net change in the levels, broken down by the two populations.

One interesting result that is worth pointing out immediately, is that the first group of students, in general, rated their before-classes level lower than the second population. We believe that this can be contributed to the common adage, "You don't know what you don't know". The first group of students did the assignments and soon discovered how much they really didn't know, whereas the second group did not come to the same realization. For example, the auditors' perception about their AWS and Node.js skills was that they knew those technologies relatively well before starting the class; meanwhile the first group of students came to realize that their skills were almost nil.

Next, notice that the estimated pre-class knowledge levels are higher than the estimated pre-class skill levels. In general, the students felt they had a conceptual understanding of the course concepts, including AWS, which only one student had exposure to before class. From this, we can see that students (and perhaps all people) tend to believe that they are able to generalize conceptual knowledge into new areas that they have not seen before.

 ISBN: 978-1-61208-438-1

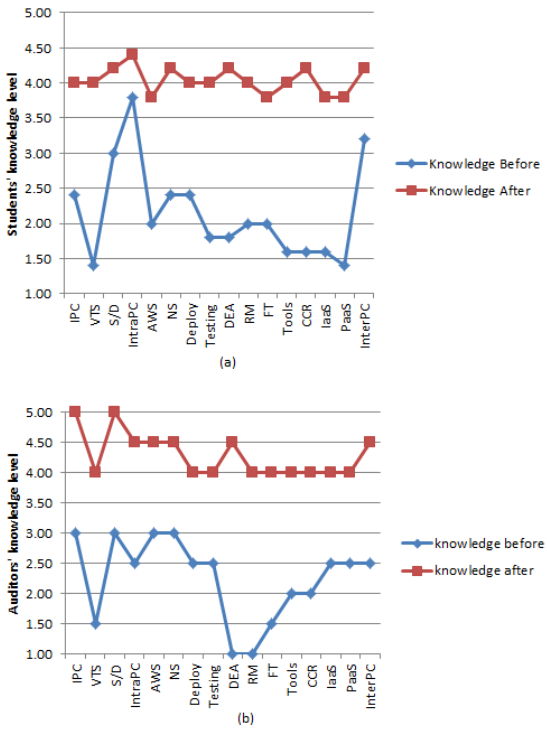

Figure 3. Before and After Knowledge Levels.

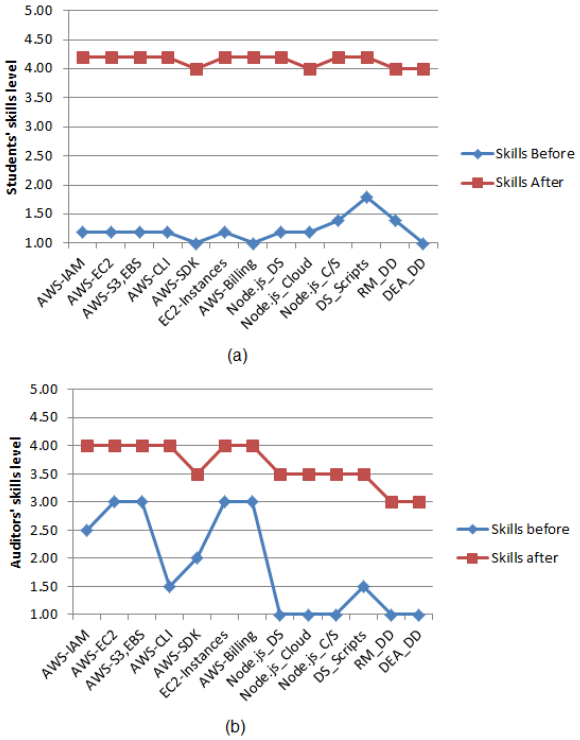

Figure 4. Before and After Skills Levels.

Figure 5 shows evidence that the first group of students truly improved their skills. Their net change for every skill was higher than the net change for the second group. Interestingly, the same is not true in the knowledge area. At first glance, this might seem odd, but considering the timing and relative nature of the self-made estimates, there is a possible explanation. Specifically, the students who didn't do the assignments naturally felt that their biggest growth was in increase of conceptual knowledge.

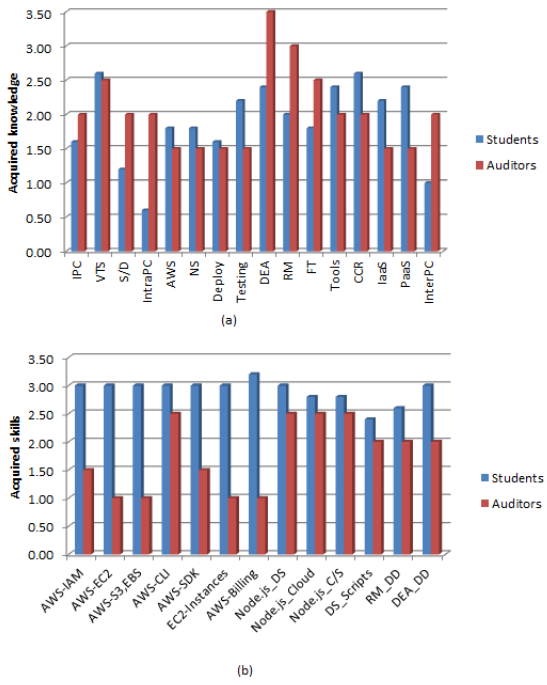

Figure 5. Perception of acquired knowledge: differences between the after and the before.

V. RELATED WORK

Other higher-education institutions are using cloud computing resources in courses that focus on distributed applications or network programming. Clearly, these platforms allow the students to use realistic testing and production environments. Moreover, there are large research universities that have implemented private clouds on their campuses and use them in the classroom. For example, Syracuse University provides a local virtual machine lab used to form virtual networks for security projects [9]. North Carolina State University supplies computing resources over the Internet with their Virtual Computing Lab [10], Arizona State University developed V-lab for Networking Courses [3], and Okanagan College and King's university College talk about using a cloud for educational collaboration [11]. Nevertheless, these private solutions are often not economically viable for many universities [7], and therefore they can only consider public cloud solutions.

Programming assignments that use public or private clouds can add value to the learning experience and increase students' skills directly related to possible professional careers [4] in network programming [7], distributed systems [11], systems administration [4], security [4][9], data processing [12], among others. Furthermore, a major benefit is that students do not need to simulate network

communications over a localhost interface [7]; instead, they can use multiple virtual machines and real network communications to better understanding the distributed system components, their roles, and the related concepts.

Using a public cloud for hands-on activities offers benefits such as scalability, flexibility, security, cost-efficiency and accessibility [7], which all are key characteristics of distributed systems [2]. Public clouds also add an interesting and valuable dimension to the execution and debugging of distributed applications [12], without needing huge budgets for private-cloud or physical-machines infrastructure. Most of the public cloud providers, e.g., Amazon, Google, Microsoft, IBM, offer grants for academic institutions that want to use their resources for educational purposes. For example, at the time of this study, Amazon offered grants up to $100 per students [13]. Other benefits to public clouds include ready access to different operating-system platforms, communication protocols, development tools, open-source code, public forums, and more.

VI. CONCLUSIONS AND FUTURE WORK

For this small case study, we conclude that programming assignments with requirements to use cloud resources were successful in helping the CS 6200 students to improve their A2R skills, as well as their core distributed-application development skills. Both the instructor's observations and the post-class survey provide anecdotal evidence of their improvement.

We also found some evidence that students are willing and even excited to learn new tools and skills, especially if they can see how it lets them put theory into practice. Even though the assignments were based on carefully crafted and sanitized requirements, they were realistic enough for the students to experience real problems and see how theoretical concepts, like vector timestamps and distributed election, could be used to solve those real problems.

Some important design criteria for assignments included: a) hiding unnecessary details, like all the other capabilities of an EMR beside the generation of disease notifications, b) focusing on requirements that put theory into practice, like the election of an RM in Assignment 4, c) including non-trivial non-functional requirements, like scalability, and d) wherever possible allow students to reuse components or knowledge acquired in previous assignments.

The survey data also opened some doors to possible future research. Specifically, we would like to conduct a broader experiment across multiple software-engineering classes of various kinds and at different levels, to explore specific ways that the design of assignments can improve A2R skills in general. From that, we hope to publish more concrete guidelines for programming-assignment design for software-engineering classes at all levels.

ACKNOWLEDGMENT

We would like to thank Amazon for providing funding for the students to use AWS resources for this class.

REFERENCES

[1] C. Ramamoorthy, "Computer Science and Engineering Education," IEEE Transactions on Computers, Vols. C-25, no. NO. 12, December 1976, pp. 1200-1206.

[2] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, Distributed Systems: Concepts and Design, 5th ed., Boston, MA: Addison-Wesley Publishing Company, 2011, p. 1008.

[3] L. Xu, D. Huang, and W.-T. Tsai, "A Cloudbased Virtual Laboratory Platform for Hands-On Networking Courses," in ITiCSE '12 the 17th ACM annual conference on Innovation and technology in computer science education, New York, 2012, pp. 256 - 261.

[4] C. Leopold, Parallel and distributed Computing, New York: John wiley & Sons, Inc., 2001.

[5] C. Gonzalez, C. Border, and T. Oh, "Teaching in Amazon EC2," in SIGITE'13, Special Interest Group for Information Technology Education, Orlando, Florida, 2013, pp. 149-150.

[6] D. Howard, Node.js for PHP Developers, S. S. L. a. M. Blanchette, Ed., Sebastopol, CA: O'Reilly Media, Inc, 2012.

[7] node.js, "node.js," Joyent, 2015. [Online]. Available: https://nodejs.org/. [Accessed 28 04 2015].

[8] W. Zhu, "Hands-On Network Programming Projects in the Cloud," in SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education, New York, 2015, pp. 326-331.

[9] j. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Seattle: A Platform for Educational Cloud Computing," in 40th ACM technical symposium on Computer science education SIGCSE'09, New York, 2009, pp. 111-115.

[10] W. Du and R. Wang, "SEED: A Suite of Instructional Laboratories for Computer Security Education," Journal on Educational Resources in Computing (JERIC), vol. 8, no. 1, March 2008, pp. 3:1-3:24.

[11] H. E. Schaffer, S. F. Averitt, M. I. Hoit, A. Peeler, E. D. Sills, and M. A. Vouk, "NCSU's Virtual Computing Lab: A Cloud Computing Solution," Computer, vol. 42, no. 7, July 2009, pp. 94 - 97.

[12] Y. Khmelevsky and V. Voytenko, "Cloud computing infrastructure prototype for university education and research," in WCCCE '10, 15th Western Canadian Conference on Computing Education, New York, 2010.

[13] A. Rabkin, C. Reiss, R. Katz, and D. Patterson, "Using clouds for MapReduce measurement assignments," ACM Transactions on Computing Education (TOCE), vol. 13, no. 1, January 2013, pp. 2:1-2:18.

[14] AWS, "AWS in Education Grants," Amazon, [Online]. Available: http://aws.amazon.com/grants/. [Accessed 01 07 2015].

 ISBN: 978-1-61208-438-1