Observations on Designing a Computer Science Curriculum Focusing on Game Programming Using Testimonials from Industry Leaders

Graham R. Smallwood and Don V. Black
California State University Long Beach

***Abstract*--The 2008 ACM/IEEE Curriculum Report section 5.4.2 mentions a "Focus on games or entertainment software" as one method of organizing a computer science curriculum. But outside of using games as a methodology for teaching standard computer science, it is also worth considering whether the techniques taught actually transition to the games industry. This paper will explain the IEEE standard for a computer science curriculum, and then compare those milestones with what the games industry wants using interviews with game professionals who are responsible for hiring decisions at top companies.**

INTRODUCTION

CS2008, as the *2008 ACM/IEEE Curriculum Standard* is referred to, is the latest document in a series published by those two professional organizations aimed at helping structure educational programs for computer science. The development process is open to all suggestions, and the final document was created and revised with successively smaller groups of leaders from both industry and academia.

While studies have shown the benefits of using game topics as a background for teaching topics straight from the standard [Sung], this paper concerns discovering if the curriculum standard itself is of value to game development companies directly. Questionnaires followed by interviews with hiring managers at Zynga, Valve, Quicksilver, and Microsoft Games were used to gauge what was truly valued in a student and potential new hire. Just as how a student won't graduate without learning what the standard dictates, they won't get a job without learning what the game companies want them to learn.

ACADEMIC STANDARD

The CS2008 document partitions all of computer science into 14 large categories, and then details 5-10 topics that fall under each one. In addition, it specifies which topics are "core". Core topics should be covered regardless of what other topics are chosen as electives. Before any analysis can be done using the standard, it is important to fully understand it. Any measurement using "hours" in the document will be converted to the easier to understand unit of "classes" where 30 hours equals one class.

A. Discrete Structures (DS)

This entire discipline is flagged as Core due to how the concepts learned will be applied in so many of the other categories. This section includes basic abstract principles such as sets, graphs, trees, and probability. Classes are likely to overlap with a math department, and it takes 1.5 classes to finish all of the Core material.

B. Programming Fundamentals (PF)

Another discipline whose entirety is marked as Core, programming fundamentals are topics that underlie any programming language. This category involves describing the concept of programming before the more advanced Programming Languages and Software Engineering categories below. Topics include data structures, recursion, event driven programming, object oriented concepts, and basic security, and 1.5 classes are designated for the category.

C. Algorithms and Complexity (AL)

These topics are concerned with the efficiency of software solutions. Picking the correct algorithm is the plan that starts the program on the right track, so topics covering the basic algorithms and strategies for picking them are marked Core, and should have one class devoted to them. The elective topics cover the more advanced choices, like automata theory, cryptography, and NP study. The topic of parallel algorithms is singled out as one of the topics that hastened the release of the 2008 review ahead of its planned 2011 publication data due to how fast it was rising as an important topic.

D. Architecture and Organization (AR)

Understanding the architecture of a computer is the first step away from the theoretical. These classes teach why programming works as it does. Memory, I/O, and multiprocessing are among the Core topics that should be given one class. Further electives cover performance and distributed systems.

E. Operating Systems (OS)

One class's worth of Core topics is dedicated to teaching how concurrency, scheduling, and memory actually work, and this is the first category to recommend required lab work. Electives cover the file system, performance evaluation, and security methods.

F. Net-Centric Computing (NC)

Only half a class worth of basic material is marked Core, but the rising importance of network-capable code is pointed out. Electives cover web, mobile, and multimedia needs on today's networks.

G. Programming Languages (PL)

A key takeaway from the study that went in to the standard is that students should be able to learn more than one language. This category aims to teach what the definition of a language truly is so that the best choice of language can be made for any one task. The one class of core topics covers abstraction, basic types, and object oriented mechanics. The electives go on to cover the semantics and design of a language itself.

H. Human-Computer Interaction (HC)

Only half a class is Core here, and it just covers the basics of UI. But the electives go on to study how designing an interface early in a project can help define and improve the structure of the underlying code. Other topics cover distributed communities and understanding what makes good UI different from working UI.

I. Graphics and Visual Computing (GV)

Graphics is the first category to be filled with elective topics after just one Core week of describing what graphics are. Four sub-categories are laid out: Graphics, Visualization of data that has no form, Virtual Reality, and Computer Vision. Graphics is the only sub-category further split into topics though, and covers modeling, rendering, and animation.

Of particular note among the topics in GV is the one on Game Engine Programming, as it is the only topic devoted to game programming in the document. It only covers using an existing engine, however, and how to use a system that has rendering, physics, collision, sound, AI, and terrain already implemented.

J. Intelligent Systems (IS)

Half a class worth of Core covers basic concepts. But hidden inside the topic covering search strategies is one subject of particular interest to games. The topic covers the idea of an exhaustive search of future possibilities being used in game AI. This is one kind of AI with a direct connection to games possessing an AI opponent, such as an abstract game. The rest of the electives cover more mainstream topics in robotics, learning, language processing, and perception.

K. Information Management (IM)

This is another category using half a class to cover the Core ideas of data modeling. This category devotes many topics to the concepts behind databases. There are almost enough topics to cover a whole major, and all are very focused.

L. Social and Professional Issues (SP)

This is an interesting category in that the standard recommends interspersing each topic in to the technical class to which it is related. The Core topics add up to half a class, but each topic is only one or two hours long. Security and privacy, intellectual property, and professional ethics are some specific topics, while "Social Context" is a Core topic pointed out to be part of every class.

M. Software Engineering (SE)

This category brings all of the others together in to the ability to make a final product, and is one full Core class. From designing the system to writing and using tools to software processes, the "how" of making a program is covered. Additionally, topics in requirements, validation, and management ensure that it is the right program. Electives are more specific and focus on reliability concerns, security, and risk assessment.

N. Computational Science (CN)

This is the only topic with no core topics at all, and is far to the end of the "science" side of computer science. Modeling and simulation of fluid dynamics, the structural analysis of materials, and other hard core topics that game programming does not use are here. One notable exception is the topic of parallel computing, which was singled out in the industry responses.

INDUSTRY PROFESSIONAL OPINIONS

To find out what skills were the most useful for students to learn to get in to the games industry, one must ask the senior engineers and development directors in the games industry responsible for the hiring of new graduates. The questionnaire covered topics from each discipline in the curriculum standard as well as several areas outside of pure academic learning.

A. Opinions on the CS2008

1) DS – Discrete Structures

All respondents considered this an obvious and automatic set of topics, as understanding discrete structures is a basic requirement of computer science.

2) PF – Programming Fundamentals

Like DS, this category is taken as a given and didn't garner much response.

3) AL – Algorithms and Complexity

Highly valued across responses. In addition to simply knowing enough algorithms so that intelligent solutions can be made, the ability to do cost analysis of algorithms was also called out as important. In a real time environment, doing something quickly is as important as doing it right.

4) AR – Architecture and Organization

Two of the four interviewed said that knowledge of how the CPU and memory functioned was important, with one respondent connecting the subject to the ability to analyze algorithms.

5) OS – Operating Systems

Not considered relevant by any of the respondents.

6) NC – Net-Centric Computing

Rated highly with the Zynga respondent, who pointed out that all the growth in the game market is in on-line products. But even those from more balanced companies pointed out its importance as a specialty, not as something that all computer

science students need to know.

7) PL – Programming Languages

All respondents agreed that more important than knowing any one language is the ability to learn a new language quickly. Even within a company that is primarily C++, a situation might arise that requires another language

8) HC – Human Computer Interaction

Responses on this were split, with some saying it was a specialization and some saying that everything needs to be written with usability in mind.

9) GV – Graphics and Visual Computing

All respondents agreed that everyone needs some level of graphics experience. Writing shaders and creating new special effects are in the realm of a specialization. But every piece of code for a visible object needs to be written with the understanding of 3D transforms, models, and animations.

10) IS – Intelligent Systems

High level AI was soundly panned as a subject. At a simple level, it is important for tasks like pathfinding and making state machines. But the higher level thoughts and plans in a game are not driven by AI. They are driven by scripts written by the game designer. The AI is not supposed to be smart or good at the game; it needs only to be fun. "I want them to understand path-finding options. All the rest of it is useless," stated the Zynga interviewee.

11) IM – Information Management

Only one respondent mentioned databases even in passing. They are used in social and massively multiplayer games.

12) SP – Social and Professional Issues

Not mentioned.

13) SE – Software Engineering

Parts of this category were deemed essential. The software design topic is a game programmer's entire job. Two responses mentioned the source control part of the Tools topic specifically, and one singled out the making of requirements documents.

14) CN – Computational Science

Only one topic from here was mentioned as desirable, but it was stressed highly. Parallel computing is a very difficult topic and is becoming more and more common as hardware improves. It is rarely to be found on current incoming resumes as a skill. This sentiment was echoed in the standard's AL section above. Additionally, not finding the subject in the listed curriculums points to schools being behind on this subject.

Another topic from this category was stressed, but in the negative. Knowledge of physics simulations was deemed entirely unnecessary. The reasons given range from either because the engine is already made, physics can be done in purchased middleware, or the game just doesn't need anything complex. The respondent from Valve stated, "Physics engines are a commodity. Being able to write one or innovate in this area is a very specific skill that's not of that much use, frankly."

B. Opinions on General Topics

1) Game Design

Many university programs across the country have classes outside of computer science entirely and cover gaming in general but are seldom about programming. A detailed analysis of these programs are outside the scope of this paper, but, in short, while these non-programming game classes clearly don't fit in to the IEEE standard, the industry responses held them in high regard. A strong passion for gaming outside of the school setting is the preferred choice, but these game-centric classes are accepted as an alternative. To be able to implement a system common to a particular genre of games, it is extremely helpful for the programmer to be familiar with that kind of system already. For example, trying to explain concepts like "attack-move" or "item sockets" to a lay person can be difficult, but a game aficionado would take those terms as a given.

2) Group Projects

All respondents made the importance of this subject perfectly clear. "I wish curriculums would focus on more large scale projects," mentioned the Valve respondent. All of those interviewed wanted graduates who had experience in group projects, as co-operation and communication are essential qualities on a production team. This ranked even higher than having completed and shipped a game project on one's own. One respondent responsible for performing engineering interviews even admitted that once satisfied that the basic engineering skills were present in a candidate, the in-person interview is primarily a personality and compatibility test to ensure that the new hire would work well with the team.

This makes the classes that appear to just be narrowly covering the SE category much more important than just a one-category class may seem. However, a semester-long class with a single project as its goal can't be run in too open-ended a fashion or it can encounter some difficulties.

On a game programming team, the person managing the team is most often the most senior. A Lead Programmer or Development Director will assign and monitor all tasks. But in an undergraduate college class everyone on the team will likely have the same level of limited experience. As a solution, a fledgling local game programming class had everyone write up a game idea and submit it. The most well thought out plans were picked as the projects the groups of students would develop, and the owner of that idea was made Lead based on the reasoning that they were the most passionate about completing the project. This worked out well in more than half of the groups, but two of the teams had leadership problems which resulted in disorganized groups with no direction.

This observation has been studied before [Cliburn]. When offered the choice between a very dry but structured programming assignment and the chance to make an open-ended game, over three quarters of students choose the task with the details chosen for them. This agrees with the observations of that game class as there all the students were forced to choose the open-ended project and most of them could not handle it.

Better results were achieved by another semester-long project that used a more structured approach at Penn State. [Ryoo] Even though it was only a 200 level class and was teaching simpler topics, its method of keeping structure on an open creative process is very interesting. The projects were kept in discrete phases that gave complete freedom but enforced regular checkpoints.

3) Specialist vs Generalist

All industry respondents came down on the side of generalists here. Only one even mentioned having a concept of specialists, as they hired network specialists and 3D graphics specialists with experience. "Focused specialty is somewhat useful for specialized areas like graphics engineering and network programming, but even then it is career limiting if a person only knows one area," noted the respondent from Microsoft. As pertains to college students, this point is very important to note as it doesn't always match up with academic opinion. The author of UCI's program, for example, made a point of saying how broad their program was when asked. But at USC their author states of their program, "The industry demands an increasing supply of graduates trained not as generic programmers, artists, or producers, but as specialists in the particular technologies and techniques that drive the latest best sellers." [Zyda] This is a dramatic disconnect in educational philosophies between schools and the core of what this paper is examining.

4) Miscellaneous

There are some other interesting differences in opinion between the industry respondents and academia to point out. Teaching the math behind 3D transformations is one topic that varied greatly, with academia valuing it highly and the industry finding it irrelevant. Understanding the transformations behind everything is essential, such as knowing how to get the hand's position from the shoulder and where to put a camera. Matrix math itself however was rated near the bottom, as every company had full math libraries to handle it.

Parallel processing was one highly desired skill that gets barely a mention in the standard, but was considered of great value by the respondents. Valve was particularly specific; "[knowledge of] threading, job systems and joblets, co-routine systems, and general best practices in implementation of engines capable of properly utilizing multi processor systems.". Fortunately, in the release notes for CS2008 it is noted that the rise of parallel computing is one of the driving forces that pushed the document to be released in 2008 instead of later, so this discrepancy will be closing.

Finally, some ancillary skills bear mentioning because of their frequent inclusion. Source Control may seem like a production skill not worth mentioning, but as every student will be working on existing code on the first day of any job, it is on the wish list of the respondents. But the respondents felt that Debugging is the one topic that is most underrepresented. If any school could teach the type of debugging that is not the documented, organized, test-case kind, that would be well received by all questioned.

CONCLUSIONS

Schools do not necessarily need classes in applied game programming techniques to make their students viable potential hires at game companies, but important game topics could be worked in to existing classes so that students can gain the necessary and desired skills.

But more important than improving the teaching of any one topic is extending the overarching requirement for group project experience. Many high-level classes are pure theory and involve no programming at all. Classes like Networking and Security would benefit from hands on work not because it would be practicing for a job, but for the act of working in a team itself.

As discovered in this paper, the extra burden placed upon a team leader can damage the success of the project for an entire team if not handled well however. For this reason the task assignments in the group should be done with the involvement of the professor. This would remove the pressure of planning a project on top of implementing a project off of one student as well as provide a much clearer measure of individual performance for grading purposes.

With the games industry being as large as many other more traditional industries, there is a constant need for good programmers. Hopefully school programs can concentrate on the broad knowledge and group project aspects of what companies want to see, and everyone will benefit.

Cliburn, D.C.; Miller, S.M.; Bowring, E.; , "Student preferences between open-ended and structured game assignments in CS1," *Frontiers in Education Conference (FIE), 2010 IEEE* , vol., no., pp.F2H-1-F2H-5, 27-30 Oct. 2010

Ryoo, J.; Fonseca, F.; Janzen, D.S.; , "Teaching Object-Oriented Software Engineering through Problem-Based Learning in the Context of Game Design," *Software Engineering Education and Training, 2008. CSEET '08. IEEE 21st Conference on* , vol., no., pp.137-144, 14-17 April 2008

Sung, K.; Hillyard, C.; Angotti, R. L.; Panitz, M. W.; Goldstein, D. S.; Nordlinger, J.; , "Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context Into Existing Introductory Programming Courses," *Education, IEEE Transactions on* , vol.54, no.3, pp.416-427, Aug. 2011

Zyda, M.; , "Educating the next generation of game developers," *Computer* , vol.39, no.6, pp.30-34, June 2006

The industry respondents from Microsoft Games, Quicksilver, Valve, and Zynga declined to be directly quoted by name.