COT4210 Final Paper

by Anthony Reyes

Topic

Should COT4210 be retained as a required course for computer science majors?

Justify your answer philosophically and via concrete examples.

08/03/2011
Introduction

Colleges and universities are living, breathing entities that throughout time experience changes in funding, curriculum, pedagogy and even philosophy. It is increasingly common for these institutions to modify the plans of study for the majors they offer or to add a course at the expense of another. In this paper, we will discuss why the addition of a capstone course to UCF’s Computer Science plan of study should not result in the loss of its existing *Discrete Structures II* (COT4210) course and what implications this exchange might have.

COT4210 is very much a theory-based course and should more appropriately be called *Computer Theory* because of the nature of the material. From the early stages of grade school through the pinnacles of higher education, many students carry with them, a philosophy that every course they take should have a direct and practical application in the real world and often feel discouraged when taking highly theoretical or abstract courses like the course in question due to a feeling of wasted effort. We will discuss the flaws in this philosophy and demonstrate via concrete examples, why courses of theoretical nature, especially in Computer Science, are highly important both to the individual taking them and the practice as a whole.

The Importance of Science

COT4210 is currently a requirement for all students studying *Computer Science* at UCF. We will begin by taking a closer look at exactly what the word science entails. In its purest form, science is the study of nature and a quest to understand the way the world (or universe) around us works. Merriam-Webster defines science as: “*knowledge or a system of knowledge covering general truths or the operation of general laws especially as obtained and tested through scientific method.*” Science in and of itself does not include the development or production of practical tools or functional products. It is merely a means to satiate mankind’s curiosity and thirst for understanding. When science is used to produce, as it so often is, the outcome is what we call technology. In special circumstances, technology that is not understood can be produced and is therefore not built upon science; but the extent of these applications is very limited. In general, we can say that science is a prerequisite for technology.

Modern times and societal structure have caused a shift in human motivation towards monetary gain and as a result, people are understandably more concerned with practical skills than in understanding. Although many universities and technical colleges offer more practical plans of study like the *Computer Programming* major at Keiser University, a fundamental approach such as that provided by the
Computer Science major at the University of Central Florida will provide students with enough fundamental understanding to learn the practical applications on their own. Additionally, such an understanding will make the student more competent and adaptable whereas a more shallow understanding may have to be completely restructured in the event of a paradigm shift, a phenomenon that occurs frequently in the realm of technology and computing.

It is somewhat difficult to pinpoint exactly where and how the knowledge and theory has its benefits because the effects of such knowledge are more subconscious and automatic than the effects of explicit, repeatable information. In general, however, such knowledge will provide for a broader point of view of the practice. Having a broader point of view and looking at what is often referred to as “the big picture” results in better planning, more efficient implementation and a higher degree of confidence during execution. This “big picture” is only achievable with a deeper understanding that includes theory.

Illustrations
To illustrate, we will examine a hypothetical scenario: Jim is a software engineer that works at Superior Software. He is highly trained in writing object oriented programs and is a dedicated worker. Unfortunately, Jim was never interested in theory and obtained to a Computer Programming degree from a technical college to preserve time. It can be easily observed in Jim’s code, that he uses the same handful of problem-solving approaches, even when they don’t fit properly. He knows which algorithms to use where but never has a full understanding of them and has trouble coming up with his own algorithms to solve unusual problems. Additionally, when Superior Software changes its programming approach and decides to start using a functional language, F# instead of C#, Jim has a significantly difficult time adapting and learning new algorithms to fit the new paradigm. Other programmers are also thrown off by the change but adapt in due time because their knowledge includes the fundamentals of computer science and they see the new paradigm as nothing more than a different way to express what they already know and are able to devise new solutions and algorithms as they proceed. Jim, on the other hand, will have to wait for a course to be offered that deals with functional programming to be effective at work again.

The importance of a deeper understanding doesn’t only involve software and Computer Science though. Our illustration including Jim and Superior Software is useful in getting the idea across but real life examples can be seen throughout history that show that having a broader perspective is critically important. Lieutenant Colonel Stanislav Petrov was a Soviet ballistics officer in charge of a secret bunker outside of Moscow that monitored the Soviet’s early missile warning system. He was trained to respond
to this system by sending a request for retaliation to his superiors. Petrov’s job was one of great importance, and his training did not merely consist of learning to push a button when he heard an alarm. He was well educated and knew a great deal about fields surrounding his responsibilities including politics, ballistics and software. In 1983, when the early missile warning system was triggered with an alert that the United States had launched five missiles towards Russia, Petrov did not act mechanically in sending the retaliation request because he had what he referred to as “a funny feeling” that convinced him there was something wrong with the system. In part, he reasoned that if the United States was going to retaliate, they would send far more than five missiles. His training and surrounding knowledge laid the groundwork for his decision that quite literally saved the world.

**Conclusion**

Our study of Turing Machines and Complexity Theory in COT4210 will likely never play a part in saving the world but knowing these principles and how they relate to software will help us make better decisions more quickly. When attempting to construct a new algorithm, students having taken COT4210 will be able to identify more easily, whether or not a perfect solution is possible and whether or not to resort to a heuristic. Additionally, when debugging applications, their understanding of Turing Machines and automata will give them more confidence because they have a deeper understanding of what is taking place before them. These are just two concrete identifiable applications for the knowledge obtained from this class, but as mentioned before, the subconscious factor plays a large role as well.

Understanding the theory behind a practice in any field, especially in software engineering, is critically important. This is not to undermine the importance of practical knowledge and experience in any way; proficiency in a skill is the primary motivation for most degree seeking students; but the practice is incomplete without the understanding. Software engineering is potentially the most volatile field in the modern world with specialized knowledge one year becoming obsolete the next.

As a University, teaching theory and letting students learn practical knowledge on their own is highly preferable to teaching practical knowledge that will eventually become obsolete. COT4210 teaches the core fundamentals of *Computer Science* and addresses *Software Engineering* problems on digestible level that will facilitate students in developing their own practical knowledge on their own time or in the workplace. COT4210 should be retained as a required course for students pursuing a *Computer Science* degree because it is the very heart of that which entails *Computer Science*, a science that finely compliments software engineering.
Bibliography
