Casey Spain

Brysen Hawkes

Mitch Patterson

Rochak Kunwar

**W11 Teach Disagreement, Discussion Q, Debate**

**Disagreement**

**Quote:** “All engineering is still fundamentally about physical systems; software engineering is not… engineers have to take courses concerned with the material world, such as chemistry and statistics”

**Response:** I understand the overall premise of what Michael Davis is trying to say with this statement. However, I disagree with this for a couple of reasons. The first reason is that up to about 40-50 years ago, all engineering had to do with the physical world because there was no digital world that we have today. As computers have gotten more and more complex and advanced, this digital world of engineering has developed along with it. Because of these developments, I would say that this definition of engineering having to do with the physical world is flawed and outdated. We should look to other definitions of engineering to decide whether or not software engineering is in fact engineering. The next reason that I disagree with the premise of this argument, is that a good chunk of software engineering requires an understanding of the physical systems that you are working with. You can have a basic knowledge and ability to write code without a knowledge of the physical systems, but you would not be able to understand how certain things work like the stack for example.

**Quote:** “engineering is a certain function, any “systematic, disciplined, quantifiable approach to the development, operation, and maintenance [of something]”. That assumption must be false. It would force us, for example, to rank accounting—a field no one supposes to be engineering—as “financial-records engineering”

**Response:** Here the author is presenting a generally excepted definition of engineering that he claims is over-simplified. I feel the author in this case is letting his zeal get the better of him. The most important part of this Quote that he misunderstands is “development, operation, and maintenance [of something]”. The keyword here is “and”, it does not say “or”. An accountant may operate a system to do their jobs, but do they do all three of those verbs? Overall, they do not while a software engineer would. Also using a very vague definition of a noun to prove whether a software engineer is an engineer is an oversimplification. Software Engineers commonly create new code and methods. An accountant does not need to create new software or methods to do their job.

**Quote:** “Software engineering has, indeed, become a profession. What it has not become is part of the engineering profession.”

**Response:** There are multiple problems with this statement. What defines an ‘Engineer’ isn’t clear or set. The second problem is that the author is assuming the role of stating who an engineer is like it’s an exclusive club. There are multiple fields of Engineering and they can be pretty unique from one another. The only real common denominator among Engineers is that the word Engineer is in the title which Software Engineers have.

**Discussion**

1. Please read the "Senses of Engineering" section of the Davis article. The author claims that none of the four senses that he identified adequately describe software engineering. Which of the four do you think most closely describes what we do?

The third sense is the one that most closely resembles what we do. Engineering-as-discipline describes having a distinctive way of carrying on an activity. That resembles software engineering in the fact it requires knowledge of a programing language and must adhere to the rules of that language.

1. The third column of the first page of the Davis article presents a formal definition of software engineering. Do you feel this definition is adequate? If not, can you propose ways to improve it or perhaps a better definition altogether?

The definition looked pretty good for the most part. It doesn’t really feel complete to use, but we couldn’t really think of how to improve it. It talks about needing a quantifiable approach to solving a problem. That is where it gets a little dicey. In SE there could be an approach that works every time, but that doesn’t mean that that are many other approaches that would be better on an instance to instance basis.

1. At the bottom of the second column of the second page of the Davis article, one aspect of the Software Engineering Code of Ethics and Professional Practice is presented. What are your thoughts on this part of the SE code?

We believe the statement is true for SE and that what he said of other engineers is false. When making any product the interests of all parties should be moderated and are. Software Engineers just work more closely to these parties so it is more apparent than that of say an electrical engineer.

1. In the third to last paragraph of his article, Davis suggests that software engineering as a discipline might be divided into sub-disciplines. What would those sub-divisions be?

There were lots of opinion about sub-disciplines. Some of my team members were against the idea of being a software engineer different than engineering. There thought was software engineer is actually an engineer and it doesn’t need to be separate. They thought we are not in 17 centuries were physical and material engineering is the only engineering; A virtual and untouchable things can be engineered too.

But, in case if it becomes a different discipline, we said **software** will be the main discipline. And the branches or sub-disciplines will be software engineering, computer science, Graphics designing, Algorithm and Data Visualization, Artificial Intelligence, Programming, Computational Theory, Big Data, etc.

The other programs which are mostly conducted by join department effort like bioinformatics, computational economics, computational history, computational physics-chemistry will continue to be joint-department program. These programs will not be under software discipline.

1. Under the "Why do we need a new type of engineer?" section of the Parnas paper, a definition of an engineer is presented. How does this definition compare to those of the Davis article? Do you feel this definition is sufficient for describing what a software engineer does? If not, can you propose ways to improve it or perhaps a better definition altogether?

Parnas describes engineering with a more traditional as engineering is the use of science and math to design a better product which is important for the safety and well-being of the public. While Davis approach is more liberal and broader; he doesn’t take one side support and defines all the possibility what software engineering can be.

We concluded that it is impossible to give a 100% accurate definition of engineering. We will never be able to define engineering until we draw a universal line which will divide non-engineering and engineering discipline.

We agreed with both people definition, but we feel like we need to draw that line to really differentiate the engineering and non-engineering field. The engineering sub-discipline needs to study heavy math at least few credits, they need to know at least one physical science like physics or any areas like geoscience, biology, chemistry, plant life, human body, etc. Engineering discipline needs to teach students about how and why of the problem so they can make a better product. It cannot be something like training where we learn to use stuff and learn techniques that are already created by engineers. Those people are technicians. People in business are also not engineers; these days people use engineer title even in business like risk engineering, financial engineer, etc. But we think that is wrong. Engineering title needs to be earned; it is not ethical to give everybody engineering title.

1. Under the "Software engineers are not just good programmers" section of the Parnas article, the author claims that software engineers need to be knowledgeable about non-technical issues as well as technical ones. Has anyone had an experience in this area? How does one prepare for this in the workplace?

No one in our team was experienced in the work area but we all agreed on this point of Parnas. We said software engineering, computer engineering, IT, Computer Science, and Programmer are all different areas. Software engineers are responsible for making better software that will lead the team of programmers. But software engineer must have programming knowledge and experience. Software engineers are more like an architect and programmer are the implementer. We didn’t discuss much about IT; we just said they are more like maintenance of computer system and software. Computer Science is more like a theory and research area where you work on making better product and technique and one needs to have programming knowledge in this area too. Almost all the software field needs programming knowledge except IT, so we said those programs do not deserve to be even close to the software.

So, we said that if one wants to work in the software field, not project management then one needs to have a knowledge of programming. One needs to know the inner working of software and should be able to design software in a team. If one knows how programming language works, then they can design software in more ease even if they don’t program by themselves.

1. In the middle of the first column on page 22 of the Parnas article, the point is made that computer scientists and software engineers need to have a different perspective on the body of knowledge associated with computing. What are your thoughts on this issue?

As we discussed this we mostly agreed on the fact that the differences that were pointed out in the article between scientists and engineers were mostly true. However, we talked a lot about how we don’t see those differences as prevalent in the computer science and software engineering degrees. After it was said that we don’t really see those differences as prevalent, we looked up the courses necessary for computer science students and software engineering students. We noticed that the biggest difference between the two was discrete math and algorithms & complexity. These classes being required by computer science majors backs up the premise that computer science students need to be more focused on research. However, we thought that it was false that it was only important for computer science students to stay up-to-date on the most recent technology. It is important for anyone in software related careers to stay up-to-date on new technology.

1. Please read the first full paragraph on the second column of page 22 of the Parnas article. A point is made there about "responsibility for correctness." What does the author mean by this? Do you agree with his point?

We decided that we think the author is trying to state that since science students are doing innovative research, they can publish their results and receive feedback from other scientists and professionals. However, when you are a software engineer and you are developing a product for a company, your main source of feedback is from other software engineers that you are working with and the consumers. In this case, you are more responsible for making sure that your work is done correctly because ultimately, you will be held responsible for errors. We decided that we agree with this premise to an extent. However, the point was made in our discussion that most software engineers work on teams where they get to review each other’s work, so often times, you still have professionals to help make sure your work is done correctly.

1. Please read the section titled "Differences in topic coverage" of the Parnas article. Which classes and/or topics in our programs are currently taught in a "computer science way" and which are taught in a "software engineering way?" Which could or should be taught in a way different than they currently are?

Our discussion again revolved around the courses that computer science majors are required to take that software engineering students are not required to take. Most of these classes are math related and are used to develop better algorithms. These classes are definitely more science related. However, a member of our group made the point that the courses that deal with writing code in c++ could be more engineering minded in that it would be helpful for engineering students to see more of the transition from writing code to finalizing a product. The course that we all agreed was the most engineering minded is software design and development. This course is fully centered on developing a mobile application from start to finish.

**Debate**

1. Should BYU-Idaho focus its program on computer science or software engineering?

Pros: BYU-Idaho could focus its program more on one major to raise its credibility. This would help students trying to get jobs in the chosen field after they graduate. It would also lend itself to a more focused faculty insuring better prepared graduates.

Cons: BYU-Idaho accepts many students and focusing on one major over the other would reduce the number of applicants they could accept. It would be unfair to students in the other field. Overall, we feel it is better to have both major’s because BYU-I works better as a well-rounded school then a focused one.

1. Should the BYU-Idaho software engineering degree be certified by ABET?

Pros: This would be an excellent addition to the program. It would give graduates more credibility. Overall as a team we felt the benefits outweigh the negatives on this one.

Cons: It could become more costly for the school and prove difficult to keep up, while also providing higher workloads for students.

1. Does software engineering deserve to be a separate discipline than computer science?

Pros: This would more accurately represent the industry differences in the fields. Currently at BYU-I Software Engineering appears to be in less advanced version of Computer Science when in reality they cover different aspects. We think it would be better to separate them so that Software Engineering isn’t just an easier version of Computer Science.

Cons: This would increase the class roster of BYU-I, increases necessary faculty and complicate the system.

**Works Cited**

“Software Development vs. Software Engineering.” *Software Development Versus Software Engineering*, Software Engineering Insider, www.softwareengineerinsider.com/articles/software-development-software-engineering.html.

Curran, Bill. “What Is Software Engineering?” *Ubiquity*, 1 Oct. 2001.