Summary:

The article "Computer Science != Software Engineering" by the software consultant Chuck Connell depicts the argument how software engineering and computer science disciplines are vastly different from each other. The author discusses each discipline to a certain degree and then contrasts them to illustrate their differences. Connell draws the line between software engineering and computer science where the human comes into play. It is emphasized that everything that requires human component is an element of software engineering which was and still is attempted to be analyzed rigorously. Furthermore, Connell claims that every concept in computer science has rigorous analysis and is well-defined. However, he also argues that "we should not try to make software engineering into an extension of mathematically-based computer science. According to him, doing so would distract us from important advancements.

Supporting Points:

Firstly, computer science concepts are all well-defined and clearly explained. Each of them is built on earlier stuff. That's why computer science is cumulative. Its results can be revisited because it is simply a branch of science. Yet, questions that computer scientists are interested in diverges from practical software engineering challenges.
Software engineering has always been slippery and usually has imperfect results. We have had a good number of different development techniques such as structured programming, extreme programming, OO programming, open source etc. and these appear to be best-practices. However, this only tells us that there is a good chance that they could work in the new software development project with a new team. It does not go further than that. This is mainly due to involvement of human component. Software engineering results are heavily influenced by people, whereas in computer science although the results can be used by people, they are not directly affected by people. The formal results in computer science do not depend on human behavior which is unpredictable.

Points Against:

1) The thesis is self-fulfilling. If some area of software engineering is solved rigorously, you can just redefine software engineering not to include that problem

This point does not address the whole scope of software engineering, however it shows that if you define your problem narrow enough, then you can derive rigorous results.

2) Statistical results in software engineering already disprove the thesis.

This is also a point brought up by challengers to this article. Chuck argues, however, that those statistics are based on estimation and approximation because they involve human component. Furthermore, statistics only look at past data to extrapolate for the future.

3) Formal software engineering processes, such as cleanroom engineering, are gradually finding rigorous, provable methods for software development. They are raising the bright line to subsume previously squishy software engineering topics.

This approach tends to be very subjective. They fail to analyze the engineering process broadly. Instead, they narrow it down to make them mathematically provable, leaving the core still very ambiguous.

Reader's Opinion:

As a student who is taking both Theory of Computing and Software Engineering in the same semester, I should admit that this article seemed very interesting to me. In the theory class the main focus lies on proving mathematics of computer science, ranging from computability to complexity whereas in software engineering class we learn about different methods in conducting software creation process all of which are still used in different contexts. Therefore, I believe that there is no single right answer/proof in software engineering. It's a rapidly changing field. Computer Science, however, is difficult to change. Whenever a new result is obtained, it is perceived as a breakthrough.

Although I agree with the most points that the author makes, I am confused by the intention of this article. It seems to me that by comparing software engineering with computer science it is almost as if we are comparing apples and oranges. It is true that there is some relationship between computer science and software engineering, but there is a reason why one of them is called science and the other engineering. Those two disciplines, because of the way they are conducted, differ vastly from each other and are not meant to share similarities. Science has always been rigorous, yet engineering always focused on making practical things that are of some use for humanity's progress.

My personal experience and observations so far showed me that the two disciplines feed each other. No single discipline is above the other and both support each other interchangeably. It is difficult to imagine one field without the other. Nevertheless, one can confidently say that without computer science, or in other words, without founding the mathematics, the tenets of software engineering would not be built.