Armour, P.G., 2007. The Conservation of Uncertainty. Comms. ACM [online], 50 (9), 25-28

Critical Review by Christopher Jones

Introduction

Software measurement involves gathering quantitative data about certain products and analysing it to influence actions and plans, with technology being used for a wide variety of purposes. However, there are very few methods available to measure software effectively.

On September 2007, ACM Senior Member, Phillip Glen Armour created the article called "The Conservation of Uncertainty" that discusses the measurement of software in specific ways such as counting source lines of code (SLOC) and counting the requirements of projects. He also describes the importance and differences it has compared to the past as programmers have, "…less to learn than we would do writing the same application in a lower-level language, so we can create the functioning system much quicker".

According to Dictionary.com, the adjective, uncertain is defined as, "…not able to be relied on; not known or definite". Armour is calling the measurement of software to be an uncertain process and is trying to find alternative ways of measuring systems instead of the usual SLOC method. I am analysing the style, structure, content of the article and examining certain points that he could improve upon.

Style

The style of the article is very academic, showing precise, lengthy information that most readers who are familiar with the computing industry will quickly recognise. It's styled as if you are having a direct conversation with the reader about his opinion on the matter, grabbing the attention of a small number of readers that are in that area of expertise. However, for most readers, it is not styled to make them fully understand the points Armour is trying to get across.

During the first half of the article, it has interesting points. On the other hand, it gets harder for viewers to continue reading through to the end. The reason why this is true is because the topic isn't engaging enough to read 5-pages of it. Also, the author makes no attempt to explain why you need to know the complexity of a system. Planning an article is a very important stage and I feel that Armour should've spent more time organising his ideas for this article.

Lastly, writing a good conclusion on an article to evaluate the points that have gone before and concluding it all is very important. Armour says that, "The conservation of uncertainty means that if this will be true of any metric we can devise to size a system." The conclusion to this 5-page article could've been improved by reminding the audience why it was written in the first place.

Structure

With the article having field-specific information about software measurement, the structure of the article that Armour wrote is unorganised throughout the article as if it's a stream of consciousness. The first half of the article is average at best, explaining what he thinks about SLOC as well as showing the differences between measuring code now and measuring it 20 years ago.

The closer you get to the end of the piece, the more convoluted it gets and has no organisation at all. The structure of the article goes as follows; showing the readers Armour's opinion on the process of measuring software and the problems with counting SLOC. It then discusses the difference between measuring code now compared to measuring it 20 years from now. After that it begins with describing aspects of sizing systems, rounding it all up by showing why they called the article "Conservation of Uncertainty". In my opinion, I feel that there could've been a better way to structure the article to make it more user-friendly.

Content

Armour begins the article by discussing two known methods of measuring software, counting SLOC and the software requirements that were given during the design stages. With regards to counting SLOC, he quoted an email saying that "I can't believe that, in this day and age, you suggest counting lines of code". Armour doesn't create a balanced argument as the article is implying that there are no good reasons to count SLOC to measure the software quality which is obviously false.

Using SLOC is somewhat controversial; however, the work effort does have a high correlation with SLOC, meaning it can be useful in estimating the time it takes to develop a program. However, SLOC is very poor at measuring the functionality of software because a more skilled programmer would be able to develop a similar program with less code. The International Journal of Emerging Technologies [2] state in their article 'Analysis of

Source Lines of Code (SLOC) Metric' that, "Line of Code serves as an intuitive metric for measuring the size of software because it can be seen and the effect of it can be visualized." This shows that despite Armour's criticism, SLOC has its advantages.

The best way to prove that measuring software by counting SLOC isn't the most effective method is to compare it with an author writing a book. The quality of the book and the length of the book have no correlation, this is because quality has a higher priority than quantity. The amount of work completed by the author and the length of the book usually has a direct correlation as it takes longer to write the book.

He then reviewed the method of counting requirements to measure the quality of software, claiming that he "took part in a study to see if alternative metrics to the perennially unpopular SLOC could be used". Armour concluded that they could get an actual count of the requirements given for their project and believes that it "held out great hope for an alternative to SLOC". I believe that it is successful during the early stages of development because the developers are able to query the requirements for their current project and get an actual count of the requirements. The need for requirements gathering is confirmed by Linda Westfall in 'Software Requirements Engineering: What, Why, Who, When, and How' [3] when she states, "If software requirements are not right, companies will not end up with the software they need." This confirms the need to gather requirements in the early stages of developing the software's programming.

Armour then talked about an exercise he participated in where they, "…came across two requirements quite close to each other. Both requirements were worded almost identically". He continued by discussing that when creators are looking at the potential for knowledge content in their units, you should "access the size/complexity of a system; we see that different units can have different knowledge densities and different ranges of knowledge content". I agree with his statement because the size of the system usually has no connection with the complexity of the system, as really good programmers can complete the same task using fewer lines of code.

Summary

This was a very detailed article, giving interesting and articulate ideas. There was a definite need for a short conclusion to summarise the article and never clearly explained to the readers why measuring software is important. The article did get convoluted during the final half of the article, describing the ways of sizing a system which could've been improved. Overall, it was a very interesting article, but isn't a significant contribution to the field of software engineering and would be very hard to read, regardless if you were in that field of expertise or not.

Bibliography

[1] Armour, P.G., 2007. The Conservation of Uncertainty. Comms. ACM [online], 50 (9), 25-28
Source: http://www.ijetae.com/files/Volume2Issue5/IJETAE_0512_25.pdf

[2] International Journal of Emerging Technology and Advanced Engineering, 2012. Analysis of Source Lines of Code (SLOC) Metric. [online], 150
Source: http://www.ijetae.com/files/Volume2Issue5/IJETAE_0512_25.pdf

[3] Linda Westfall - 'Software Requirements Engineering: What, Why, Who, When, and How', 2006 [online].
Source:https://cs.anu.edu.au/courses/comp3530/readings/The_Why_What_Who_When_and_How_Of_Software_Requirements.pdf

Word Count with Reference: 1271
Word Count without Reference: 1216