Summary of Barry Boehm’s “A View of 20th and 21st Century Software Engineering”

Boehm’s article, “A View of 20th and 21st Century Software Engineering”, is an in depth look at the world of Software Engineering from the eyes of experience. The author doesn’t clutter his article with praises or mockeries; although, he offers his opinions with evidence and support. This articles traces the history of Software Engineering from it’s initial birth from theory into practice, and follow it until present day. Boehm also let’s us look into his crystal ball, as he tries to predict the future of Software Engineering based on it’s own history. In short, the article is a compacted summary of major obstacles, triumphs, and defeats that Software Engineering has endured, and how Software Engineers can enhance the field by examining and learning from these.

Boehm begins his article by explaining the emergence of Software Engineering in the 1950’s. He describes how, in the beginning, Software Engineering was treated the same as all other sorts of engineering. Basically, the process would involve a team of Mathematicians working with a team of Hardware Engineers. The plan of development that would most naturally result from a team of such characters would be a model very similar to today’s waterfall model. This would be due to the Hardware Engineer’s background and instinctive design techniques.

Boehm continues his dive into the history of Software Engineering when he outlines the details of the field’s progress during the 1960’s. He explains how designing of software diverged from the usual concept of sequential models to a more layered style of editing and implementation. The reasons being for this were, in short, that software engineers were beginning to realize that “software was much easier to modify that hardware” was, and that software, also unlike hardware, did not suffer the consequences of ware and tear. This style of software development became known as the “code-and-fix” method. The unfortunate result of this design strategy was unmaintainable code, or spaghetti code, unorganized utilization of programs, and an uprise in “cowboy programmers”. At the end of this decade, “it was clear that better organized methods and more disciplined practices were needed to scale up to the increasingly large project and products that were being commissioned”.

Next, the author examines the 1970’s. He explains how certain pitfalls from concepts such as goto-statements led to the Structured Programming movement. The success of this movement led to many other structure approaches in software engineering. The waterfall model was re-implemented with a more structured look. Towards the end of the 1970’s, however, it was evident that the waterfall approach was inadequate due to such drawbacks as being “heavily document-intensive, slow-paced,
and expensive to use”. Because of these drawbacks “many organizations were finding that their software costs exceeding their hardware costs”.

In the 1980’s, according to Boehm, was all about productivity and scalability. By now it was obvious that engineering software was, indeed, a different animal than that of conventional engineering; thus, different methods and practices would have to be explored to tame the beast. To help combat issues with productivity and scalability, the US Department of Defense, DoD, introduced the Software Capability Maturity Model, SW-CMM. Once refined, the SW-CMM “provided a highly effective framework for both capability assessment and improvement”. The success of this model soon spread to the rest of the world, where in organizations began to apply this model to their own practice. Another introduction to the software world was the Unified Modeling Language, UML, and “silver bullets”, which were basically obtuse solutions to former problems with programming. These “silver bullets” included such advances as expert systems, higher level languages, object oriented design, powerful workstations, and visual programming.

During the 1990’s, the introduction of the World Wide Web, and the internet, strengthened the use of object oriented design, and further developed the UML. Also, the development of open source software matured and grew, thanks to such establishments as GNU, and the Free Software Foundation. Another huge standpoint for the 1990’s was the need for user friendly software. This made human-to-computer interaction a very marketable trait for software to have.

Now, in the decade of the 2000’s, we are recognizing alternative trends in software engineering, such as rapid application development. “This rapid pace of change has cause increasing frustration with heavyweight plans, specifications, and other documentation imposed by contractual inertia and maturity model compliance criteria”. Basically, that is to say, that rapid application design is not very friendly when it comes to documenting specifications, etcetera. That it is easier to just dive right into the coding process and keep implementing features until the software works, get paid, and move on. Another reoccurring dilemma has to do with quality of software and human-to-computer interaction. I think that Boehm says it best when he states “a recurring user-organization desire is to have technology that adapts to people rather than vice-versa”.

Boehm continues to explain his predictions for future trends in software engineering. A few to note in particular are his theories on globalization and systems of systems. In a way, these two concepts go hand-in-hand. Think that, a bunch of independent systems can be unified by another system, and those systems can be unified by another system, which will result in a hierarchy of systems. Thus, to unite a myriad of separate systems, it is important to have a global system.

The history of our programs in this course can be summarized using Boehms ideas expressed in his article. In our experience, we set up a simple waterfall model to follow, and this model set milestones that were to be strictly adhered to, this is very similar to the SW-CMM model, which also tailored to a waterfall sort of regiment. Of course, our methods are also unique, as I assume most are, since every project has unique requirements. All in all, Boehm does an excellent job of summarizing the history of Software Engineering.