Review of:
“A View of 20th and 21st Century Software Engineering”

Elmar Keij
University of Utrecht, The Netherlands
ekeij@cs.uu.nl

Abstract
This review paper is written in partial fulfillment of the Software Engineering Course at the University of Utrecht. It consists of a small summary of a paper written by Barry Boehm, “A View of 20th and 21st Century Software Engineering”. The paper by Boehm gives a view of the history and the future of Software Engineering. It discusses every decade of the last century from the 1950’s up until the 2000’s and his predictions for the 2010’s, 2020’s and beyond. He identifies the trends in those decades and there relation with succeeding developments in software engineering. After the summary I will discuss my opinion and thoughts about the paper.

1. Summary

The field of software engineering is vast and we can distinguish many different types of software engineering. Besides this diversity the basic elements for this field of engineering have changed and probably will continue to change significantly over time, as apposed to other fields of engineering. To analyze the past (the 1950’s to the present) and predict the future of software engineering Boehm tries to apply the Hegelian hypothesis.

The thesis of the 1950’s was “Engineer software like you engineer hardware”, this resulted in a sequential waterfall-type model. And the software engineers were really either hardware engineers or mathematicians.

In the 1960’s there emerged an antithesis, software crafting. Because people realized that software was easy to modify more and more organizations adopted the code and fix model instead of the Critical Design Reviews that is used in hardware engineering. Also with the development of higher order programming languages it became easier for non engineers to enter the field of software engineering. These “new” people were more comfortable with the code and fix model. However this approach resulted in unreliable spaghetti code. At the end of the 60’s people realized that better organized software engineering methods had to be designed.

In the 1970’s the Structured Programming movement began. This movement can be divided into two branches the formal movement(antithesis) and the waterfall processes(synthesis). The first focused on the correctness of a program whilst the second focused on structured programming. Sadly people misinterpreted Royce’s Waterfall Model and thought it was a purely sequential model. At the end of the 70’s problems were popping up with both movements. The formal was to difficult for the majority of the programmers and also wasn’t scalable enough. And the waterfall processes resulted in buggy code, because the managers pushed their teams to coding instead of documenting. By the end of the 70’s quantitative methods were devised for measuring software.

The synthesis of the 1980’s was all about improving the productivity and scalability of software engineering. In 1984 the software capability maturity model was developed and software tools were improving/maturing significantly in the 80’s. Also improvement of software processes increased productivity by reducing rework and work avoidance. A lot of next generation languages were developed to facilitate better software reuse. These, mostly object oriented languages, worked great and turned out to be the biggest improvement in terms of productivity.

The 1990’s introduced an antithesis, concurrent vs. sequential processes. Organizations realized that the sequential waterfall model was insufficient for developing some software. Especially for software where the requirements were not known up front. Therefore people turned away from the waterfall model and began to develop other models like the risk-driven spiral model which supported concurrent engineering. Another great example of concurrent engineering was introduced in
the 90’s, the open source community. Also the importance of the usability of software became very important.

In the 2000’s there was partial synthesis and antithesis, namely value based software engineering and agile methods. In the 00’s people became frustrated with the big and heavy plans, specifications and documentation and several agile methods were introduced. The most well known is eXtreme Programming (XP). Agile methods reduces the bureaucracy which was introduced over the years. However big projects were unfeasible with just agile methods. The value based approach puts this into perspective and determine which part of a project can be done with agile methods and which parts should be done with plan driven methods. Also model driven development became very popular.

For the future, Boehm predicts that in the coming decade (the 2010’s) globalization, systems of systems and reuse of COTS are going to be the primary points of development. Beyond that Boehm predicts smarter and more (artificial) intelligent and adaptive systems.

2. Position within the Software Engineering Field

The paper “A View of 20th and 21st Century Software Engineering” sketches a high level view of the history of software engineering. Therefore it is difficult to assign the paper to a certain “part” of software engineering. However I like to position it toward the software process “part”, because the paper primarily talks about the different software processes which emerged or evolved over time. The goal of the paper is to analyze the history of software engineering and point out the strong and weak points of a certain decade. As the author points out this information is valuable to remember, for that “Those who cannot remember the past are condemned to repeat it”1 and “If you haven’t been made aware of successful histories, you’re often condemned not to repeat their successes”2.

3. Question for this paper

Consider the pluses and minuses (listed in Section 4.1) and choose one that you think has had an influence on (some) programming languages and describe this influence. I think there is not just one point that has had an influence on some programming languages. In my opinion all of following points have had their influence on function programming languages.

+ Think outside the box. At the time that functional paradigm was thought up it was definitely thinking outside the box. The fact that it is an whole other paradigm is proof enough, I think. The same can be said about the object oriented paradigm, however this paradigm has much more similarities with the procedural paradigm and the step from procedural to OO looks a bit more obvious. But the functional paradigm on the other hand has some fundamental different ideas at it’s core.

+ Eliminate errors early. This goal also has had a clear influence on functional programming languages like Haskell. Just take a look at the static type checking mechanism of Haskell. This is entirely geared toward catching errors at compile even before a program can be run. Of course this goal is not exclusive for Haskell or functional programming, but I think that Haskell at the moment has the strongest type checking mechanism.

- Avoid cowboy programming. Also the influence of this is notable in functional languages it is very much related with the previous point. It is simply harder to do cowboy programming in a purely function language like Haskell, because it just won’t compile. Of course the functional paradigm does not exclude all cowboy programming but I do think that it makes it harder to do.

4. Opinion

I can be pretty short about my opinion about the paper, I liked it. It was well structured and therefore not hard to read. I have never read a paper about the entire history of software engineering, let alone one that analysis the strong and weak points and I think the author did a pretty good job at identifying those points. However my primary problem with the paper is that it is to short. The subject of this paper is so big that I don’t think that it can be handled in fourteen pages. I’m also not convinced that all strong/weak points were found and that more can be identified. I would have also liked to see more explanations and reasoning about the different decades. Things that I missed in the paper were primarily better explanations or explanations at all of abbreviations and definitions, like the Hegelian View which is pretty crucial for this paper. To conclude with some positive comment, I really liked the conclusion. In the paper it is not made very concrete what the success and failure factors of a certain decade are and paragraph 4.1 gives a nice concise enumeration of these positive and negative points.

\[^{1}\text{George Santayana}\]
\[^{2}\text{Barry Boehm}\]