Software Engineer Biography

CSU33012-202122 (SOFTWARE ENGINEERING)

Essentially, if you have a technical argument or question, you have to sway two or three other people who are very savvy. They know what is going on, and you can't put anything over them.

Ken Thompson said this in an interview (Unix and Beyond: An Interview with Ken Thompson) when asked to give insight into the coherent vision of his projects. It is a quote that resonates with me as someone who makes a great appeal to problem solving and abstract reasoning. Many aspects everywhere in Thompson's career there is a strong foundation of theoretical computer science and mathematics research, which I decided to write about.

As a preface, I think it needs to be said explicitly the kind of person Ken Thompson is. He loved games as much as he loved computer logic and mathematics in grade school. In his early 20s he had developed a game for a cutting-edge operating system about landing on to-scale planets in the solar system with your savvy in-game spaceship. Furthermore, he even ported the same game to an older machine's architecture just to go on playing it. These are the kind of things you need to remember about Thompson because it is already evident how much he cares about the finer aspects of software. Bear in mind the portability of his videogame and the user engagement inherent to games. Such concepts of personal computing were novel in his time.

Thompson was one of the researchers working at Bell Labs in the 1960s to abandon the Multics project to develop a new kind of operating system. Multics envisioned Unix, although Thompson and his colleagues were wholly responsible for why it fostered the new age for communal computing. Unix is the first portable operating system, created in 1969. It was written entirely in C which meant it could essentially operate anywhere, on anything. Accounts say that it was tooled together initially with Thompson's personal projects such as his beforementioned game Space Travel on an obsolete machine. That was because there was no organizational backing for Unix, or any prospect of getting off the ground with some revolutionary software.

Ken was responsible for many components in Unix, such as the hierarchical file system's conception and the B programming language (a precursor to C). But in order to get the attention of his managers in Bell Labs there needed to be some interfacing with the user. One of the first key elements of the operating system was the text editor and what Ken made was a fine example of one in retrospect. What inspired editors like emacs today, ed for Unix was a reimplementation of qed, developed at his alma mater University of California Berkeley. But there was a twist – ed was cobbled together with Ken's own reimplementation of qed and contained new features in the realm of software. For searching text, he had implemented regular expressions in one of the first applications of such a concept in history. It is known to be a factor in the popularity of regular expressions among programmers today, as well as the place they have in universal standards like POSIX.

Although even ed was part of that standard – rather, this is because Unix was at the centre of computing ever since Ken was able to successfully move the operating system to more and more new machine architectures. Creating accessible programs and engaging fellow software developers was the kind of feedback loop to create as Dennis Ritchie called it "a system with which a fellowship could form". It makes a lot of sense why in that time Ken wanted Unix to be by software developers for software developers – it was the beginning of making computers personal.

The sheer power of Thompson’s codebase created a new niche for small but efficient computer systems that had one option to do exactly what they needed to. It is a philosophy that carries on in Ken Thompson’s career that things are made simple. I think the widespread exposure of his ideas spawned many great communities in the field of computer science ever since. It became a beautiful and large feedback loop in his work on the Go programming language and its acceptance as a competitive programming language with the likes of Java and C++.

But for now, I would like to move on to another of Thompson’s endeavours. He clearly saw the exercise in problem solving and user interfacing in computer chess. Besides just creating a chess-playing program he worked with new colleagues on something far more challenging. The world champion chess computer “Belle” demonstrated Thompson’s use of brute-force. It was also one of his core philosophies to resort to brute-force when all else fails. What came of it was the complete enumeration of chess endings with the perfect mesh of Joseph Condon’s hardware and Ken Thompson’s “endgame tablebases” software.

It was something else to engage in an activity like computer chess. It was an example of how inspiration for clever engineering theories can come from anywhere. Or if anyone were to benefit from creating games and computer players, I think it would be the pure application of ideas. Ken Thompson had no small part in setting the standard for what would be sophisticated deep learning projects today is what I think. All it took was even a naïve, brute-forced approach to creating an abstraction of a chess player. That he can lean into complex theories of computers making “decisions” is a testament to his problem-solving skills and an impressive feat without a doubt.

In 1983 upon Thompson and Ritchie receiving the Turing Award for their work on Unix, Thompson gave an acceptance speech “Reflection on Trusting Trust”. He poses that it is more important to trust the software developer than the software. “To what extent should one trust a statement that a program is free of Trojan horses?” This philosophizing in the area of software engineering is my favourite since it directly deals with security. The speech discusses self-replicating programs and the learned behaviour of compilers – how the C compiler can learn new tricks.

Thompson is trying to communicate something kind of significant in his career. Pattern matching is a branch of theoretical computer science that he explores from his humble beginnings and continues to elaborate on here. He talks about how the C compiler in its state could match and replace the login command in Unix, and essentially give him unrestricted access to any user’s data as a result. Of course, “any casual perusal of the C compiler would raise suspicions” and rightly so as it would be an elaborate but concealed flaw in Unix. Thompson asks what can be done of this. The point of Thompson’s talk was grim but important. It was about how trustworthy using programs or program-handling programs that are not written entirely by the user. “A well-installed microcode bug would be almost impossible to detect.” Many movements came out of these reasonable questions of software efficacy, such as the open sourcing of code and scrutiny of bugs.

Having moved away from his more playful antics, I think Ken Thompson is serious about his passion for good software and it shows. He took care in his acceptance speech to make a poetic tale about his experience writing in the C language and the dangers of compiler design and software engineering. I think a lot about the layers of abstraction in computers that I am not fully in control of or have no stake in whatsoever. It is a great perspective to have as it means I can ask the same questions Ken did. If there were a simpler point to all of Ken Thompson’s speech it would be that writing programs and developing software was an involved process. It requires a lot of your attention as the developer and maybe everyone ought to be the same with how prominent software has become in our lives.

The 90s had Thompson further contributing to universal standards in computers. He went to work on the UTF-8 encoding scheme for characters. It is another way besides Unix he has shaped the internet and in particular the World Wide Web. It was initially implemented on the Plan 9 operating system at Bell Labs. Plan 9 was a project Thompson had heavy involvement in, as it heavily utilized the principles of Unix. The idea was to expand on the idea for major system facilities in a new setting where personal computing is rapidly changing. Unsurprisingly these “principles of Unix” so ingrained into this ever-increasing family surrounding it incorporated many of Thompson’s own ideas. “When in doubt, use brute-force.” It stands that Thompson’s most memorable wise remarks cemented themselves as commandments in software engineering culture. Another research operating system called Inferno would see Ken Thompson and the usual suspects work with the concept of the virtual machine.

Lately, Thompson joined to co-author the Go programming language at Google. With the intention of making a language where he would have to be talked into every feature (as if only one feature should do a certain task within reason), Go would compete with high level programming languages. There would be “no extraneous garbage put into the language for any reason”. I was introduced to the likes of Java or C++ in my journey into computer science and couldn’t agree more with the statement that popular languages have too many bells and whistles for the mere purpose of “having as few implementation dependencies as possible”. It is something I believed long before learning about Go as well and admired as non-existent ideal.

It goes to show with people like Ken Thompson there is plenty to be done with software yet. Where I am now starting out in computer science and software engineering, I hope to approach pair programming the way Thompson did. The pair that was Thompson and Ritchie were often commended for their extensive periods of examining and writing programs together for just about every project they could over the years. The writing code itself seemingly had endless possibilities to Thompson or else there would not be so many great things under his name. When asked if he still having fun in the field before the turn of the 21st century in the beforementioned interview (Unix and Beyond: Interview with Ken Thompson), he had only one thing to say.

Yes, there are still a lot of fun programs to write.

Bibliography

- 1999 Interview – https://cse.unl.edu/~witty/class/csce351/howto/ken_thompson.pdf
  Interview with Ken Thompson
  Insight and quotes
- IEEE Computer Society – https://www.computer.org/profiles/kenneth-thompson
  Biography summarizing achievements
- About Page – http://cs.bell-labs.co/who/ken/
  Supposedly published by Ken Thompson
  Fascinating links, resume etc
- Unix – http://www.catb.org/esr/writings/taoup/html/ch01s06.html
  Inspiring list of Unix philosophies and background
  Insight and quotes
- Acceptance Speech – https://dl.acm.org/doi/10.1145/358198.358210
  Title – Reflections on Trusting Trust
  Published 1st August 1984