CSU33012 – Software Engineering

Biography of a Software Engineer – John McCarthy

Over the course of my studies in Computer Science, I have become acquainted with many impressive pioneers who have propelled the field to new heights in both the realms of the practical and the theoretical. One figure who made critical contributions in both domains, and for whom I have a particular personal reverence, is John McCarthy. When contemplating a degree path, I was torn between choosing philosophy or computer science, and while having chosen the latter I still retain a deep interest in the former. My favourite area of inquiry is the philosophy of mind and consciousness, and this area has a fascinating theoretical overlap with the realm of Artificial Intelligence (AI). For this reason, I was compelled to discover more about the man who is credited with launching the field of AI. While McCarthy himself may not be foremostly considered a software engineer, there is no doubt that much of the development landscape today is steeped in the fruits of his research. His practical innovations can still be observed in the modern day, and are present in many of my course modules this semester. Among these innovations are the concept of *time-sharing*, which is considered to be a precipitant to the modern explosion in the popularity of cloud computing, and the development of the LISP programming language, which is the precursor language to Haskell. I will discuss these practical innovations, alongside the importance of AI to software engineering in general, in this biography.

John McCarthy was born on September 4, 1927 in Boston, Massachusetts to an Irish longshoreman father and Lithuanian activist mother[1]. His prodigy became apparent in childhood, as he taught himself college level mathematics. McCarthy graduated high school early at sixteen years old, and found himself attending graduate-level mathematics courses in his freshman year at Caltech[2] where he pursued a bachelor's degree in mathematics. Shortly after graduating in 1948, he attended a talk given by polymath John Von Neumann at the *Hixon Symposium on Cerebral Mechanisms in Behavior*, an event intending to consolidate research done regarding human cognition across the fields of biology, psychology, and mathematics. A loose concept resembling AI had first entered academic thought in 1936 when the Church-Turing Thesis proved that any process of formal reasoning could be replicated by a Turing Machine. This led researchers to ponder whether human cognition could too be reduced to formal reasoning and therefore modelled and replicated artificially. Von Neumann's paper touched on this by focusing on systems whose self-reproductive behaviours were defined by logical rules. McCarthy was deeply inspired by the discussions at the event, and was influenced to devote his career to research that would have a massive impact on computer science as a discipline, and software engineering by proxy.

McCarthy sent his independent research to Von Neumann, who encouraged him to take up a research position at Princeton. There, he met contemporaries whose help he would enlist in the project that would become AI. Among the most notable were the father of information theory, Claude Shannon, and Marvin Minsky, who would lead advances in AI in parallel to McCarthy in the following decades. Shannon himself had recently published a

paper on what was essentially a rudimentary chess-playing AI, and eagerly agreed to set up a journal alongside McCarthy to catalogue papers written in the area of intelligent machines, called '*Automata Studies*' [2]. Unsatisfied with the first round of submissions the pair received which were unrelated to AI, McCarthy officially coined the term '*Artificial Intelligence*' to nail down the concept he intended to investigate. The 1956 *Dartmouth Conference on Artificial Intelligence* was approved shortly thereafter, and is generally considered to be the advent of the field, unifying mathematicians and computer scientists alike in research towards a common goal of creating software systems that "enable computers to display behaviour that can (broadly) be characterized as intelligent" [3]. The field received huge interest from the press, as systems quickly emerged that could beat human checkers and chess players. The US Department of Defence followed suit, having seen how integral Turing's work had been to the Allies' efforts in World War II twenty years earlier, and provided funding for research at several US universities. The theoretical side of the field took off, prompting the establishment of accomplished AI research labs at MIT, University of Toronto, University of Texas at Austin, and Imperial College London in the years following.

Despite lofty ambitions at the time of 'solving' the brain by the dawn of the 21st century, there is no doubt that the field of Artificial Intelligence has been successful in spurring a massive upheaval in all aspects of life, for the user and software engineer alike. In short, when placed in a certain context, a modern AI system can read its environment and *learn* what to think, rather than being *told* what to think in advance. This makes software solutions highly adaptable, and behave in useful ways that may not be obvious to a programmer ahead of time. This in itself provides a boost in productivity for software engineers - rather than having to code this logic themselves using a conventional algorithmic approach, it allows the system to devise its own behavioural rules in real-time. Many subdomains of AI have seen practical applications introduced in the modern world, with software analytics company Teradata reporting that AI solutions are already in production in some capacity in ~80% of businesses [9]. For example, language comprehension technologies can be deployed for translation, auto-correct, auto-completion, speech recognition services, and can perform sentiment analysis to analyse customer satisfaction or provide e-commerce sites with virtual shopping assistants. Computer vision techniques are included in the design of autonomous vehicles, facial recognition software, quality control checks in factories, security systems, and medical diagnosis systems. While software engineers develop AI systems as components of their end products, the tools they use to produce these products are themselves enhanced by AI. For example, *Trello* is a popular productivity management tool often used to aid in AGILE development. It features personalised automation tips and improve the software engineering workflow based on repeated actions performed by the team. The impact AI has had on software engineering is only set to become more pronounced in the coming years as deep learning and neural network technologies improve.

Concurrent to his establishment of the MIT AI lab in 1958, McCarthy developed a programming language family called *LISP,* to complement his studies into AI. The language was considered the de facto language for AI programs at the time, and is still in use today. Its influence can be seen in the countless successor languages it inspired, including *JavaScript*, *Perl*, *Lua*, *R*, *Python*, *Swift*, and *Haskell*. LISP was based on an alternative model of

computation to the Turing Machine called the *Lambda Calculus*, devised by logician Alonzo Church in 1936. In this model, computation is captured by applying a function to data, rather than a being written as a series of sequential instructions [4], as was the case with so-called *imperative* languages of its time such as *ALGOL* and *FORTRAN*. This approach to programming is known as *functional programming*. In a more general sense, McCarthy also established the link between the Lambda Calculus and logic programming, where an algorithm is defined not by directly specifying the exact steps to be performed to reach a solution, but rather indirectly, by detailing the facts and scope of the problem domain and leaving the compiler to determine a precise solution. Functional and logical programming languages, while not inherently domain specific, lend themselves easily to the construction of AI programs, since together they can dynamically update the source code of program and can learn independently to produce solutions without being imbued with the precise algorithmic steps to solve a problem in advance. Software engineers in the present day embrace functional languages as an alternative to imperative languages in cases where they covet purer, safer code, and a programming paradigm that is closer to the abstract mathematical and logical roots of computation. Haskell alone has seen use in crucial industry projects at companies such as *Facebook*, *Google*, *Intel*, *NVIDIA*, and *Microsoft* [10].

McCarthy's innovations are not solely confined to the domain of the theoretical; he also developed the first working prototype of a *time-sharing* operating system in the 1960s, whose introduction into the mainstream greatly improved access to computing technology at a time where such resources were incredibly expensive. McCarthy himself defines a time-sharing operating system as *"operating system that permits each user of a computer to behave as though he were in sole control of a computer, not necessarily identical with the machine on which the operating system is running"*[5]. During this period, organisations performed computing tasks using powerful mainframe computers whose operating systems were designed to work with lengthy batch-processing jobs. Such systems were not suited to the context of academia, in which many students and academics sought concurrent access to the computer to perform numerous smaller tasks, and would find themselves waiting in a queue perhaps for days while previously scheduled jobs ran to completion. [6] Such systems also saw a waste of resources in downtime as new programs were loaded, and in order to maximise efficiency of their costly investments, academics sought to reduce these "dead periods". The concept of time sharing was introduced, which divides up the processing time of a single central computer into time slots which are distributed among a number of concurrent programs initiated by users at separate terminals. Such systems saw success at universities such as MIT, Dartmouth, UC Berkeley, and Stanford, and were commercially adopted into companies such as IBM, CompuServe and HP. It also became a feature of the Unix operating system during the 1970s. Despite mainframe systems being phased out during the 1980s in the advent of personal computing, the concept of resource-sharing of system among multiple users is still practiced commercially today. This is seen in the recent explosion of the cloud computing sector. Cloud computing service providers provide an on-demand model for client businesses with regards computing resources, typically servers. This model is highly attractive for the client, since they need not invest the capital into a server farm, whose return on investment is harmed as resources are underutilised in response to varying demand

from their user base; the same way "dead periods" led to downtime for mainframes. Instead, businesses can use as much computing power as is needed, in the same way water can be drawn from a tap in quantities as needed. The market for cloud computing is expected to expand to $330 billion in 2020, and presently roughly a third of companies' IT budgets are spent on cloud services [7]. Of course, with this level of investment comes a high demand for software engineers to devise and maintain strategies to support cloud services. This speaks to the legacy McCarthy's introduction of timesharing has had on the world of software engineering into the present day.

John McCarthy was a leader in innovation in the field of computer science, whose impacts on the process of building software are so ubiquitous it is difficult to pinpoint them all. Having started his academic career in the 1940's, he has contributed crucial developments to the field in every decade he was working, that has shaped the way software engineers work today. He has provided us not only with tools we use to write software, but also the technical means to make the software engineering process feasible on a large scale. Far from having a diminishing impact up to sixty years later, his innovations have only re-emerged in new and equally powerful forms, as can be seen in the explosion of popularity in cloud computing, the adoption of functional programming languages like Haskell, and the popularity of Artificial Intelligence. On the ever-reincarnating fruits of McCarthy's work, his Stanford colleague Lester Earnest said, *"The Internet would not have happened nearly as soon as it did except for the fact that John initiated the development of time-sharing systems. We keep inventing new names for time-sharing. It came to be called servers.… Now we call it cloud computing. That is still just time-sharing. John started it."*

References:

1. Williams, S. (2002). *Arguing A.I.: The Battle for Twenty-First Century Science*. Ch. 2. Retrieved from https://archive.org/details/arguingai00samw/.
2. Hayes, P. J., & Morgenstern L (2007). AI Magazine Volume 28 Number 4 (2007): *On John McCarthy's 80th Birthday, in Honor of His Contributions*. P.93-102. Retrieved from https://aaai.org/ojs/index.php/aimagazine/article/view/2063/2057.
3. Thomason, R., *Stanford Encyclopaedia of Philosophy.* Retrieved from *https*://plato.stanford.edu/entries/logic-ai/.
4. Hemmendinger, D., *LISP (Encyclopaedia Britannica)*. Retrieved from https://www.britannica.com/technology/LISP-computer-language.
5. McCarthy, J., *Reminiscences on the History of Time Sharing*. Retrieved from https://web.archive.org/web/20071020032705/http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html
6. Arms, W., *Early Timesharing*. Retrieved from http://www.cs.cornell.edu/wya/academiccomputing/text/earlytimesharing.html
7. Hosting Tribunal blog post. Retrieved from https://hostingtribunal.com/blog/cloud-computing-statistics/#gref

8. Woo, E., *John McCarthy dies at 84; the father of artificial intelligence*. Retrieved from https://www.latimes.com/local/obituaries/la-me-john-mccarthy-20111027-story.html
9. Teradata, *State of Artificial Intelligence for* Enterprises. Retrieved from http://assets.teradata.com/resourceCenter/downloads/ExecutiveBriefs/EB9867_State_of_Artificial_Intelligence_for_the_Enterprises.pdf?fbclid=IwAR15ojzkHHRqGGOEAHRYX0JfG8Qx0Yi5mFin67HTfOahFrn3O0-e-In2CpM
10. Haskell Wiki entry on 'Haskell in Industry'. Retrieved from https://wiki.haskell.org/Haskell_in_industry