Turing's Influence on Programming

—

Book extract from *"The Dawn of Software Engineering: from Turing to Dijkstra"*

Edgar G. Daylight*

Eindhoven University of Technology, The Netherlands
egdaylight@dijkstrascry.com

Abstract

Turing's involvement with computer building was popularized in the 1970s and later. Most notable are the books by Brian Randell (1973), Andrew Hodges (1983), and Martin Davis (2000). A central question is whether John von Neumann was influenced by Turing's 1936 paper when he helped build the EDVAC machine, even though he never cited Turing's work. This question remains unsettled up till this day. As remarked by Charles Petzold, one standard history barely mentions Turing, while the other, written by a logician, makes Turing a key player.

Contrast these observations then with the fact that Turing's 1936 paper was cited and heavily discussed in 1959 among computer programmers. In 1966, the first Turing award was given to a programmer, not a computer builder, as were several subsequent Turing awards. An historical investigation of Turing's influence on computing, presented here, shows that Turing's 1936 notion of universality became increasingly relevant among programmers during the 1950s. The central thesis of this paper states that **Turing's influence was felt more in programming after his death than in computer building during the 1940s.**

1 Introduction

Many people today are led to believe that Turing is the father of the computer, the father of our digital society, as also the following praise for Martin Davis's bestseller *The Universal Computer: The Road from Leibniz to Turing*[1] suggests:

> At last, a book about the origin of the computer that goes to the heart of the story: the human struggle for logic and truth. Erudite, gripping, and humane, Martin Davis shows the extraordinary individuals *through whom the groundwork of the computer came into being*, and the culmination in Alan Turing, *whose universal machine now dominates the world economy*. [14, first page, my italics] [16, back cover, my italics]

*This paper is an extract from the book *The Dawn of Software Engineering: from Turing to Dijkstra* [18].

[1] See [15]. The paperback version of the book is entitled *Engines of Logic: Mathematicians and the origins of the Computer* [14]. The second edition of the book [16] was published in 2012. There is essentially no difference between the first and the second edition except for some technical editing.

These words are from no one less than Andrew Hodges, the biographer of Turing.

The central thesis in Davis's book is that logic and especially Turing's work have played a central role in the advent of the first universal computers[2]. He tries to defend this claim by writing mostly about logicians and mathematicians — including Frege, Cantor, Hilbert, Gödel, Turing, and Von Neumann — and intentionally leaving the work of the many engineers out of the picture[3].

Davis does not elaborate on the early work of the many engineers and numerical analysts like Konrad Zuse and Howard Aiken. As a result, his claim about Turing's priority with regard to the first universal computers is unwarranted. Both Zuse and Aiken had already built "universal" computers by 1941 and 1944, respectively. They did not use the adjective "universal" to describe their machines because they did *not* depend on developments in logic, nor on Turing's 1936 notion of universal machine in particular, to further their early research[4].

To put it gently, Davis overstates Turing's role in the history of the computer[5].

Turing's involvement with computer building was popularized in the 1970s and later. Most notable are the accounts by Brian Randell [43], Andrew Hodges [27], and, as mentioned previously, Martin Davis [13][14][15]. A central question is whether John von Neumann was influenced by Turing's 1936 paper when he helped build the `EDVAC` machine, even though he never cited Turing's work. This question remains unsettled up till this day. As remarked by Charles Petzold [39, p.164], one standard history [8] barely mentions Turing, while the other, written by a logician [15], makes Turing a key player[6].

The influence of Turing's 1936 paper on computer programming, on the other hand, has barely been documented. This is a rather peculiar observation when one notes that the first Turing Award was given in 1966 to a *computer programmer* (Alan J. Perlis), as were several subsequent Turing Awards. In 1966, the potentially significant connection between Turing and Von Neumann concerning the `EDVAC` was not public knowledge: the popular accounts of Randell (1973), Hodges (1983), Davis (1988, 2000), and others had yet to be published!

Therefore, instead of only documenting Turing's alleged role in the advent of the first universal computers, it is no less fundamental to examine why Turing's 1936 paper was, and is, of great importance for the field of computer programming. Doing so leads to the conclusion that (i) many first-generation programmers did not read Turing's 1936 paper, let alone understand it, and (ii) those computer practitioners who *did* become acquainted with Turing's 1936 work during the 1950s–1960s received it in at least one of the following three ways:

1. The (universal) Turing machine was viewed as a model for a digital (all-purpose) stored-program computer. Furthermore, some researchers tried to actually build Turing machines during the 1950s, i.e. *after* the first all-purpose computers were already available.

[2]Davis claims that Turing played a central role in the advent of the "first universal computers" and, in particular, in influencing Von Neumann's work with the `EDVAC` [14, p.186–187][16, p.166].

[3]As Davis states himself in [14, p.xii][16, p.xiv]. Some engineers *are* treated toward the end of the book.

[4]See Chapters 5 and 6 in Mark Priestley's book *A Science of Operations* [41]. Zuse's 1941 machine was called the Z3 and Aiken's 1944 machine was called the Mark I. See page 46 in Zuse's biography for his *later* reception of the propositional calculus [52].

In the first edition of his book, Davis does not mention Zuse who started building computing machines even before Turing began writing his 1936 paper [43, p.155]. A similar concern has also been raised by Blanke in his book review of Davis's bestseller [4]. In the second edition of his book, Davis has only added two stand-alone sentences about Zuse. See [16, p.158].

According to Zuse's biography [52, p.34,53,135], he was initially not aware of Babbage nor of Turing. Other sources that back up this claim are presented in the sequel.

[5]Extensive support for this conclusion can be found in Priestley [42, p.89][41, p.125].

[6]See also Priestley [41, p.136–137, 139] for scrutiny of Davis's account of Turing's role in influencing Von Neumann's work with the `EDVAC`.

2. Turing's universal machine helped lay the foundations of automatic programming, i.e. the activity of seeking automatic techniques and programming styles to overcome the tediousness of instructing computers.

3. The unsolvability of the Halting Problem helped some researchers lay bare certain limitations in automatic programming. The 1972 Turing award winner, Edsger W. Dijkstra, was definitely one of the first to do so.

The first and second item are discussed in the sequel. For the third item, I refer to Chapter 2 in my book *The Dawn of Software Engineering: from Turing to Dijkstra* [18]. All three items, together, lead to my central thesis: **Turing's influence was felt more in programming after his death than in computer building during the 1940s.**

The first item alludes to the 1950s when switching theorists, hardware engineers, and some mathematical logicians tried to close the gap between Turing's theoretical 1936 paper and already existing stored-program computing machines. Notable actors are the switching theorist Edward F. Moore and the logician Hao Wang. The second item will mainly be illustrated by discussing the work of two strong and early proponents of machine-independent programming languages: Saul Gorn and John W. Carr, III. These men grasped Turing's 1936 notion of universal machine in the context of programming languages and also explicitly referred to the work of the logicians Gödel and Kleene. Concerning the third item, Dijkstra was one of the first programmers to apply the unsolvability of the Halting Problem in the context of programming. He used it as preparatory work for his now-famous 1968 letter 'Go To Statement Considered Harmful' [19]. Likewise, he applied it in his 1978 correspondence with the American Department of Defence (DoD) on a technical programming example in order to convey the need to separate language design from language implementation [20]. Dijkstra's work, in short, shows that undecidability has practical implications in software engineering today.

2 Contextualizing Turing's 1936 Paper

Grasping Turing's now-famous 1936 paper 'On Computable Numbers, with an Application to the Entscheidungsproblem' [47] requires appreciation for the work of mathematical logicians, including Hilbert, Gödel, Church, and Kleene. Unsurprisingly, then, Turing's paper is rather difficult to understand for the modern-day computer professional (cf. [39]). A similar remark held for first-generation programmers in the 1950s, as the following words from 1953 illustrate:

> Türing Machine: In 1936 Dr. Turing wrote a paper on the design and the limitations of computing machines. For this reason they are sometimes known by his name. The umlaut is an unearned and undesirable addition, due, presumably, to an impression that anything so incomprehensible must be Teutonic. [1]

According to Dijkstra, Turing's seminal 1936 paper had, at least until 1950, not attracted much attention in the mathematical world at large [21]. Moreover, Hoare tried to comprehend Turing's paper during the early 1950s but with difficulty. To understand Turing's work better, he later read Davis's 1958 book *Computability and Unsolvability* [12], but also without full comprehension [18, Chapter 4]. Likewise, Naur tells me in an interview [17] that he became acquainted with Turing's work early on, but only started studying it in detail during the early 1990s (cf. [34, 35]). Toward the end of his professorship, Naur asked several computer professionals in conferences whether they had ever heard of Turing's 1936 paper. The vast majority of answers were negative.

Many other computer practitioners of the 1950s were either not aware of Turing's 1936 paper, or did not clearly see the connection between it and computing. The 1973 Turing Award recipient Charles W. Bachman, for instance, "did not really know who Alan Turing was" prior to 1973 [25, p.100]. And, the leading lady in compiler building, Grace Hopper, said the following in 1978:

> I think I can remember sometime along in the middle of 1952 that I finally made the alarming statement that I could make a computer do anything which I could completely define. I'm still of course involved in proving that because I'm not sure if anybody believes me yet. [29, p.9]

The last sentence shows that Hopper and, hence, also many of her colleagues, were not well acquainted with Turing's theory of computation. For, the crux of Turing's work is that there *are* well-defined problems that cannot be computed (i.e. are algorithmically unsolvable).

The previous paragraphs are not meant to belittle researchers in computing; they serve to illustrate that the implications of Turing's work only surfaced gradually, if at all, in certain quarters of computing.

Turing's machines and Turing machines

Turing's 1936 paper was rather peculiar, compared with the work of Church and Gödel, in that he introduced an "automatic machine" as a model for a human computing a real number[7]. Each of Turing's machines, in essence, computes a real number r, just as a very disciplined and patient human can compute r on paper with a pencil and eraser [47, p.249–251]. Consider, for instance, the real number 1/3, which is equal to $0.01010101\ldots$ in binary notation. The dots in the sequence signify the fact that the digits 0 and 1 alternate forever. Turing explained how to construct an automatic machine that computes the sequence $0.01010101\ldots$. Likewise, to compute 1/4, which in binary notation is equal to 0.01, Turing's corresponding automatic machine prints the digits 0 and 1, and then forever prints the digit 0 in accordance with the sequence $0.0100000\ldots$. In short, Turing was only interested in machines that print digits forever and *not* in machines that print a finite number of digits [39, p.76].

Important in Turing's paper was his construction of the universal machine, a construction which relied on Gödel numbering but which I shall not delve into here. In the words of Hodges:

> Turing had the vital perception that operations and numerical data could be codified alike [... I]n a remarkable application of that perception, Turing showed that 'it is possible to invent a single machine which can be used to compute any computable sequence'. This invention was his *universal machine*. [28, p.4, Hodges's italics]

By 1946 and perhaps a bit earlier, but definitely not during the 1930s, Turing was well aware of the fact that his 1936 universal machine could, essentially, serve as a mathematical model of an all-purpose stored program computer [28, p.4,6][42, p.76]. Turing and his close associates may well have been the only people to have seen this connection during the 1940s [42, p.79, 84]. It was by presenting his 1950 paper 'Computing machinery and Intelligence' [49], in which he devoted a section to 'The Universality of Digital Computers', that Turing was able to change the common perception among some of his contemporaries[8].

[7] Besides Turing, also Post conducted research along these lines [42, p.36]. And, it was Church who, in 1937, had reformulated Turing's machines as modelling arbitrary machines instead of human calculators [28, p.8–9].

[8] Priestley mentions Wilkes, Prinz, and Oettinger as people who, during the early 1950s, had clearly grasped the practical implications of Turing's theoretical notion of universal machine [42, p.86–87]. Other examples are presented later.

The work of Church, Turing, and Post of the 1930s led logicians like Post [40], Kleene[9], and Davis [12] to reformulate Turing's automatic machines during the 1940s–1950s. The recast machines compute integer functions[10] instead of real numbers, similar to the way real computing machines work. In this modified setting, a machine is provided with a finite number of digits as input, representing an integer. The machine then either computes forever and, hence, does not halt. Or, the machine only prints a finite number of digits and halts. In Petzold's words:

> In the Kleene and Davis formulation, machines that don't halt are considered *bad* machines. Determining whether a Turing Machine will properly complete its calculation and halt was termed — by Davis — as the *halting problem*. [39, p.234, Petzold's italics]

In the previous quote and in the rest of this paper, the words "Turing machine" describe the reformulated machines, *not* Turing's original automatic machines.

At this point it is interesting to delve a bit deeper into Martin Davis's own recollections. In a 2008 interview, he explained that after having given a course on mathematical logic in 1950–51, which included the work of Turing, he became one of the world's first programmers by programming the `ORDVAC` [3, p.565–566]. Doing so allowed him to connect Turing's theoretical work with physical computing machines:

> I began to see that Turing machines provided an abstract mathematical model of real-world computers. (It wasn't until many years later that I came to realize that Alan Turing himself had made that connection long before I did.) [7, p.60]

Davis's realization of the *connection* between Turing's work and real-world computing machines cannot be emphasized enough. It led him to write his 1958 book, *Computability & Unsolvability* (cf. Calude [7, p.60]). As Davis stated himself, the book was technically not novel, but placing Turing machines to the fore was [12, p.vii–viii]. In fact, one of the reviewers of his book derided the connection he was proposing with actual computing machines[11]. It was Davis's book which initiated the study of computability in the curriculum of computer science majors (cf. Petzold [39, p.328]).

In his book, Davis also formulated and proved a theorem stating the unsolvability of the Halting Problem[12]. In plain English and under some widely accepted assumptions, the unsolvability of the Halting Problem amounts today to stating that:

> [I]t is impossible to devise a uniform procedure, or computer program, which can look at any computer program and decide whether or not that program will ever terminate. [32, p.153]

This theoretical result has had practical implications in the design and implementation of programming languages, some of which are described in this paper.

The theoretical work in mathematical logic was paralleled by the work of many engineers who built physical programmable computing machines. Between 1938 and 1941, Konrad Zuse built his Z3 machine in Germany [44, 52]. During and especially after World War II, engineers in the USA and England built several machines, which were primarily used by applied

[9] Kleene's monumental *Introduction to Metamathematics* [30] was published in 1952. It covers several topics of mathematical logic, including the concepts of "recursively defined functions" and "Turing machines". In Chapter XIII, Kleene showed that both concepts are equivalent, thereby reformulating what he, Church, Turing, and others had accomplished during the 1930s.

[10] Functions whose domains and ranges are restricted to the integers.

[11] See Calude [7, p.66]. Martin Davis thinks the anonymous reviewer was the logician J. Barkley Rosser as he conveyed to me in a private discussion in Ghent, Belgium on 8 November 2011.

[12] Cf. Davis [12, p.70]. The essential ideas underlying the proof were not novel, due to the prior work of Church and Turing [10, 11, 47].

mathematicians to solve numerical problems. Among those involved were Howard Aiken, Presper Eckert, Herman Goldstine, John Mauchly, John von Neumann, Alan Turing, and Maurice Wilkes [13, 14].

The advent of the physical programmable computing machine had a great impact on the field of numerical analysis. Unlike the logicians and the electrical engineers, the numerical analysts, by their very profession, took programming seriously [31, p.3]. Several of them gradually became more involved in seeking automatic techniques and programming styles to overcome the tediousness in programming their machines. The corresponding activity, first called *automatic coding* and later *automatic programming*, would eventually include the design of high-level programming languages and their implementation by means of compilers and runtime systems.

3 Different Receptions of Turing's 1936 Paper

During the 1950s, an increasing number of computer practitioners began to view the Turing machine as a model for a digital stored-program computer, including Van der Poel [50], Burks [6], and Moore [33]. Concerning the latter, Edward F. Moore was a switching theorist who in 1952 wrote a paper entitled 'A Simplified Universal Turing Machine' [33]. In his paper, Moore started off by connecting Turing's 1936 work with his own field of expertise and then noted that Turing's machines were initially used by Turing to model humans, and that in later years they had become models for actual computers [33, p.50]. In his words:

> In fact, several present-day digital computers do actually use magnetic or perforated paper tapes as auxiliary memories, instead of merely as input-output media. Hence, a Turing machine *could also be* considered a mathematical model [...] of a digital computer. [33, p.50, my italics]

Moore also wrote that the universal Turing machine "can, loosely speaking, be interpreted as a completely general-purpose digital computer" [33, p.51].

Besides switching theorists and hardware engineers, also mathematical logicians tried to close the gap between Turing's 1936 theory and practical computing machinery. In 1954, Hao Wang presented 'A Variant to Turing's Theory of Computing Machines', which was published three years later [51]. To simplify Turing's theory and to make it more accessible to the machine designer, he defined a machine which is equivalent in power to a Turing machine but which cannot erase data from its tape. In his words: "erasing is dispensable, one symbol for marking is sufficient, and one kind of transfer is enough" [51, p.63]. Lecturing to computer designers, Wang talked about physically realizing his machines and, likewise, about the "physical realization of a universal Turing machine" [51, p.87–88]. Being concerned about *both* logicians and engineers, he made three analogies:

- "Just as logicians speak of theorems and metatheorems, there are programs and metaprograms.
- Just as logicians distinguish between using and mentioning a word, automatic coding must observe the distinction between using an instruction and talking about it.
- Just as logicians contrast primitive propositions with derived rules of inference, there is the distinction between basic commands and subroutines." [51, p.88]

Wang concluded by expressing his hope that logic and computing would bond deeper than had been the case up till 1954.

Also in post-war western Germany, researchers such as Hasenjaeger tried to build actual Turing machines, even though their own Konrad Zuse had already succeeded in building several

computing machines independently of both Turing and Von Neumann[13]. According to Hasenjaeger's recollections, he and his colleagues, in an attempt to materialize a "Turing tape", were greatly aided by Wang's variant machine in which erasing is dispensable[14]. For, the practical implication of Wang's machines is that writing on the tape may be realized by punching holes in the tape. Although this particular technique of the West-Germans never worked quite well, it did pave the way for their register[15] version of the Turing machine [26, p.183].

Besides viewing the Turing machine as a model for a digital stored-program computer, there were also researchers who focused on Turing's key notion of *universality* in the context of programming. Two such researchers were Saul Gorn and John W. Carr, III. As we shall see, they advocated a universal machine-independent language in computing. By doing so, they contributed greatly to introducing the language metaphor and, hence, the word "language" — as in "programming language" — into computing during the 1950s.

To set the stage, it was spring 1954 and the venue of interest was a conference in Washington D.C. called *Automatic Programming for Digital Computers* [38]. Instead of having a programmer tediously write down machine code, the conference attendees wanted to be able to provide the programmer with a more mathematical notation in which he could express himself more easily. The research challenge was to design a computer program that could automatically translate the mathematical expressions of the programmer into the instructions of the machine. Various automatic-translation programs were presented and discussed at the conference.

Most presentations at the 1954 conference covered mathematical notations and automatic translation programs that only worked for a specific kind of machine. Two exceptions, however, were the presentations of Gorn [23] and Brown & Carr [5]. These three researchers discussed translation techniques that were applicable for any type of machine. To obtain such a general technique, they realized that the mathematical notation, intended for the programmer, had to be independent of any computing machine. Furthermore, Gorn, Brown, and Carr sought a *universal* machine-independent *language*, i.e. a language that was close to the universal language of mathematics and, hence, applicable to a large class of mathematical problems.

Brown and Carr distinguished in their paper between the "outside human language" (i.e. the "language of mathematics and formal logic") and the "less understandable interior instruction languages" [5, p.84]. They wanted to bridge the "gap" between such languages and, hence, contribute to the emerging discipline called automatic programming.

While many people at the aforementioned 1954 conference hardly used the word "language" in their lectures — Hopper, for instance, did not use it a single time — Gorn and Carr did so extensively. Furthermore, their inspiration to do so came clearly from mathematical logic. Carr, for instance, referred to metamathematics in his paper:

> Most machine users know intuitively "how to program", now must come the stage where this intuition is formalized and transferred into the heart of the machine itself. What are the steps by which a code is developed?
>
> Such investigations would appear to lead into the regions of *metamathematics*, where the problems deal with the generation of systems rather than the systems themselves. [5, p.89, my italics]

Likewise, Gorn referred to metamathematics and Gödel in particular [23, p.75].

[13] See Hasenjaeger [26, p.182], Zuse [52, p.34,53,135], Blanke [4], Rojas [44], and Randell [43, Chapter IV].

[14] Hasenjaeger was also aided by the work of Moore and by Shannon's seminal paper 'A universal Turing machine with two internal states' [45].

[15] In this regard, see also the register machines proposed by Shepherdson and Sturgis in 1963 [46]. Their paper can be viewed as furthering Wang's research agenda, namely closing the gap between the theoretical and practical aspects of computation.

Gorn and Carr's inspiration from mathematical logic becomes even more apparent when studying their 1957 work. At Purdue University, Carr gave a lecture in which he advocated an "outside-in view" (i.e. a problem-oriented view) toward computing, and where he viewed computers as "symbol manipulators". The machine user "must become much more problem oriented and much less equipment oriented" [9, p.21]. In this regard, Carr mentioned the work of Turing, Post, and Markov, and the "Universal Turing Machine" in particular. Gorn, in turn, did not only refer to Turing's 1936 paper, but also explicitly referred to Kleene's Metamathematics [30] and the equivalence between general recursive functions and Turing's machines. The central notion of interest was Turing's universal machine, as Gorn's words illustrate:

> On the one hand the sequence of imperative and interrogative sentences which constitutes a code of instructions causing a general purpose machine to produce a desired output may be looked upon as the recursive [i.e. computable] definition of that output[.]
>
> [O]n the other hand such a sequence of instructions may be looked upon as the set of specifications of a *special purpose machine* designed specifically to give the desired output, and which the *general purpose machine* copies and imitates. From this second point of view the general purpose machine is the equivalent of the "Universal Turing Machine" which could produce anything producible by any special machine by its ability to accept and react to the description of such a machine as input data. [24, p.255, my italics]

Gorn furthermore introduced the contradistinction between *syntax* and *semantics* in the emerging field of what many would later call computer science [24, p.260–261]. He promoted the language metaphor by making an analogy between verbs and nouns, on the one hand, and order types and variables on the other hand. That is, he projected linguistics (verbs and nouns) onto the practice of coding a machine (order types and variables) [24, p.259]. As a result, the phrase "programming language" eventually became standard jargon in computing. The metaphor of language has proved to be extremely successful; it is, after all still with us today[16].

Gorn and Carr were, of course, not the only researchers in automatic programming who had grasped Turing's 1936 notion of universality. In the spring of 1959, a working conference was held on automatic programming in Brighton, England, where Andrew Booth in his opening address credited the late Dr. A. M. Turing[17] as he who "first enunciated the fundamental theorem upon which all automatic programming is based" [22, p.x]. Continuing, Booth said:

> In its original form the theorem was so buried in a mass of mathematical logic that most readers would find it impossible to see the wood for the trees. Simply enunciated, however, it states that *any computing machine which has the minimum proper number of instructions can simulate any other computing machine, however large the instruction repertoire of the latter.* All forms of automatic programming are merely embodiments of this rather simple theorem and, although from time to time we may be in some doubt as to how FORTRAN, for example, differs from MATHMATIC or the Ferranti AUTOCODE from FLOW-MATIC, it will perhaps make things rather easier to bear in mind that they are simple consequences of Turing's theorem. [22, p.1, my italics]

Given that Turing's work was recognized to be of fundamental importance, Booth continued his speech by openly wondering why Turing's 1936 work had received so little recognition until

[16]The first occurrence of the words "programming language" in a technical paper seems to be in 'Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristic' [36], by Newell, Shaw, and Simon in 1957. An earlier occurrence can be found in a 1956 newsletter [2]. Source: Nofre et al. [37] which covers the metaphor "programming language" in greater detail.

[17]All papers were collected in a book, edited by Richard Goodman [22]. That book is dedicated to Alan Mathison Turing and Appendix One contains Turing's 1936 paper [47] and his follow-up correction [48].

recently; that is, only after the advent of the computer did Turing's work "assume importance". Booth's answer, in short, was that the first computing machines were used almost exclusively by their constructors and, hence, by people who were intimately aware of their internal construction. It took some years before the machines were used for scientific applications, devised by people who were and wanted to remain ignorant of the machine itself and, hence, had to rely on automatic programming techniques. At that same conference, Stanley Gill, too, elaborated on Turing's notion of universal machine and discussed a "hierarchy of programming languages" [22, p.186]. All of this happened before the 1960s.

4 Closing Remarks

To examine logic's role in the history of computing, it is fruitful to investigate where and how papers from logic were cited and why those papers were discussed at conferences. I have tried to do just that with regard to Turing's 1936 paper and primarily for the field of automatic programming. Similar investigations have yet to be conducted for artificial intelligence and complexity theory. My thesis is that Turing's influence was felt more in programming and after his death than in computer building during the 1940s. From this historical perspective, it is no surprise that the first Turing award went to Alan J. Perlis for his contributions in the area of advanced programming techniques and compiler construction (i.e. automatic programming).

Moreover, claiming that Turing played a crucial role in the advent of the first universal computers requires elaboration on the earlier work of Zuse, Aiken, and others. Research conducted by professional historians shows, however, that both Zuse and Aiken had already built "universal" computers by 1941 and 1944, respectively. They did not use the adjective "universal" to describe their machines because they did not depend on Turing's 1936 notion of universal machine to further their research.

Therefore, instead of plunging into the history of the computing machine, I have pondered Turing's influence on programming. Doing so has brought much of Turing's true legacy to the fore. Turing's 1936 notion of universal machine was recast by logicians like Post, Church, Kleene, and Davis. Eventually and gradually, during the 1950s, recast notions helped some leading switching theorists, hardware engineers, and researchers in automatic programming to see the bigger picture of what they were accomplishing. Later, the undecidability results of Church and Turing, in the form of Davis's Halting Problem or in a form equivalent to it, influenced some experts in the emerging field of high-level programming. This last observation lies outside the scope of this paper and is discussed at length in my book *The Dawn of Software Engineering: from Turing to Dijkstra* [18].

References

[1] *Faster Than Thought: A Symposium on Digital Computing Machines*, 1953.

[2] Formation of USE — a Cooperative Organization of 1103A Users, February 1956. pages 264-265.

[3] Interview with Martin Davis. *Notices of the AMS*, 55(5):560–571, May 2008.

[4] B.E. Blanke. The universal computer: The Road from Leibniz to Turing. *Notices of the AMS*, 48(5):498–501, May 2001.

[5] J. Brown and J.W. Carr, III. Automatic programming and its development on the MIDAC. In *Symposium on Automatic Programming for Digital Computers*, pages 84–97, Washington D.C., May 1954. Office of Naval Research, Department of the Navy.

[6] A.W. Burks and I.M. Copi. The logical design of an idealized general-purpose computer. *J. Franklin Inst.*, 261:299–314, 421–436, 1956.

[7] C.S. Calude, editor. *People and Ideas in Theoretical Computer Science*. Springer, 1999.

[8] M. Campbell-Kelly and W. Aspray. *Computer: A History of the Information Machine*. Basic Books, 1996.

[9] J.W. Carr, III. Lecture given at the Purdue university, Computer Symposium, November 15, 1957. *Computers and Automation*, 7:21–22, 25–26, 1958.

[10] A. Church. A Note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1, 1936.

[11] A. Church. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58(2):345–363, April 1936.

[12] M. Davis. *Computability and Unsolvability*. McGraw-Hill, 1958.

[13] M. Davis. Mathematical logic and the origin of modern computers. In R. Herken, editor, *The Universal Turing Machine - A Half-Century Survey*. Oxford University Press, 1988. Originally in: Studies in the History of Mathematics. Mathematical Association of America, 1987, pages 137-165.

[14] M. Davis. *Engines of Logic: Mathematicians and the origins of the Computer*. New York NY: W.W. Norton & Company, 1st edition, 2000.

[15] M. Davis. *The Universal Computer: The Road from Leibniz to Turing*. Norton, 1st edition, 2000.

[16] M. Davis. *The Universal Computer: The Road from Leibniz to Turing*. CRC Press, 2nd edition, 2012.

[17] E.G. Daylight. *Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains*. Lonely Scholar, October 2011. www.lonelyscholar.com.

[18] E.G. Daylight. *The Dawn of Software Engineering: from Turing to Dijkstra*. Lonely Scholar, 2012. www.lonelyscholar.com, ISBN 9789491386022.

[19] E.W. Dijkstra. Go to statement considered harmful. *Letters to the Editor, Communications of the ACM*, 11:147–148, 1968.

[20] E.W. Dijkstra. EWD 658: On language constraints enforceable by translators, an open letter to Lt. Col. William A. Whitaker. Technical report, March 1978.

[21] E.W. Dijkstra. EWD 682: The nature of computer science (first draft). Technical report, written between 1976 and 1979.

[22] R. Goodman, editor. *Annual Review in Automatic Programming I: Papers read at the Working Conference on Automatic Programming of Digital Computers held at Brighton, 1–3 April 1959*. Pergamon Press, 1960.

[23] S. Gorn. Planning universal semi-automatic coding. In *Symposium on Automatic Programming for Digital Computers*, pages 74–83, Washington D.C., May 1954. Office of Naval Research, Department of the Navy.

[24] S. Gorn. Standardized programming methods and universal coding. *Journal of the ACM*, July 1957. Received in December 1956.

[25] T. Haigh. An interview with Charles W. Bachman OH XXX. Conducted on 25-26 September, 2004, Tucson, Arizona. Interview conducted for the Special Interest Group on the Management of Data (SIGMOD) of the Association for Computing Machinery (ACM). Transcript and original tapes donated to the Charles Babbage Institute, Center for the History of Information Processing, University of Minnesota, Minneapolis, Copyright 2004.

[26] G. Hasenjaeger. On the early history of register machines. In *Computation Theory and Logic'87*, pages 181–188, 1987.

[27] A. Hodges. *ALAN TURING: The Enigma*. Burnett Books, 1983.

[28] A. Hodges. Alan Turing: the logical and physical basis of computing. *British Computer Society*, 2007. http://www.bcs.org/ewics.

[29] G.M. Hopper. Keynote address of the opening session and the corresponding transcript of question

and answer session. In R.L. Wexelblat, editor, *History of Programming Languages*, pages 7–22. New York: Academic Press, 1981.

[30] S.C. Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.

[31] M.S. Mahoney. The roots of software engineering — an expanded version of a lecture presented at CWI on February 1990. In *CWI Quarterly*, 1990.

[32] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc, 1967.

[33] E.F. Moore. A simplified universal Turing machine. *Proc. ACM*, September 1952.

[34] P. Naur. Understanding Turing's universal machine—personal style in program description. *The Computer Journal*, 36(4):351–372, 1993.

[35] P. Naur. *Knowing and the Mystique of Logic and Rules*. Kluwer Academic Publishers, 1995. ISBN 0-7923-3680-1.

[36] A. Newell, J.C. Shaw, and H.A. Simon. Empirical explorations of the logic theory machine: A case study in heuristic. In *Proc. Western Joint Computer Conf*, pages 218–230, 1957.

[37] D. Nofre, M. Priestley, and G. Alberts. When technology becomes language: New perspectives on the emergence of programming languages, 1950-1960. *Presented at SHOT 2011 and in preparation for publication*, 2011.

[38] Office of Naval Research, Department of the Navy. *Symposium on Automatic Programming for Digital Computers*, Washington D.C., May 1954.

[39] C. Petzold. *The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine*. Wiley Publishing, Inc., 2008.

[40] E. Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 12:1–11, 1947.

[41] M. Priestley. *A Science of Operations: Machines, Logic and the Invention of Programming*. Springer, 2011.

[42] P.M. Priestley. *Logic and the Development of Programming Languages, 1930-1975*. University College London, May 2008. PhD thesis.

[43] B. Randell, editor. *The Origins of Digital Computers: Selected Papers*. Springer-Verlag, 1973.

[44] R. Rojas. How to make Zuse's Z3 a universal computer. *IEEE Annals of the History of Computing*, 20(3):51–54, 1998.

[45] C.E. Shannon. A universal Turing machine with two internal states. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 157–166. Princeton University Press, 1956.

[46] J.C. Shepherdson and H.E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.

[47] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 2nd series*, 42:230–265, 1936.

[48] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. A correction. *Proceedings of the London Mathematical Society, 2nd series*, 43, 1937.

[49] A.M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[50] W.L. van der Poel. A simple electronic digital computer. *Appl. sci. Res.*, 2:367–399, 1952.

[51] H. Wang. A variant of Turing's theory of computing machines. *Journal of the ACM*, 4(1):63–92, January 1957. Presented in 1954.

[52] K. Zuse. *The Computer — My Life*. Springer-Verlag, 1993.