

## Viewpoint

# Reading CS Classics

*Revisiting required reading.*

**W**E OFTEN FOCUS SO much of our attention on our particular research areas that we do not fully utilize the potential coming from the core theoretical computer science. We lack the fundamental theoretical knowledge of the field. Moreover, the computer science classics are unknown to many computer scientists. Knowledge of the theories of computer science helps in understanding the limitations of the field. This directly influences your ongoing research by providing you with new perspectives and insights. In addition, the stories of the pioneers of the field inspire young professionals, provide a common history to unite the community, and facilitate the recognition of computer science as an independent science and profession.

With these ideas in mind, I organized CS classics meetings in my computer engineering department during the last summer term. Our group selected a subset of classics to initialize the project. The selected classics and their respective ordering reflected our personal interests; in the end, they become part of a coherent whole.

It can be a good practice for CS professionals to compile their own list of classics that highlights some key scientific concepts of the field. Such an attempt improves the understanding of the field and serves as a valuable source of reference, as this Viewpoint attests. Our group discussed these CS classics:

- ▶ “The Emperor’s Old Clothes,” C.A.R. Hoare
- ▶ “An Axiomatic Basis for Computer Programming,” C.A.R. Hoare

### Knowledge of the theories of computer science helps in understanding the limitations of the field.

- ▶ “Gödel’s Undecidability Theorem,” S.F. Andrilli
- ▶ “Computing Machinery and Intelligence,” A.M. Turing
- ▶ “Reflections on Trusting Trust,” K. Thompson
- ▶ “The Humble Programmer,” E.W. Dijkstra
- ▶ “An Interview with Edsger W. Dijkstra,” P. Frana
- ▶ “Computer Programming as an Art,” D. Knuth
- ▶ “The ‘Art’ of Being Donald Knuth,” E. Feigenbaum
- ▶ “Donald Knuth: A Life’s Work Interrupted,” E. Feigenbaum

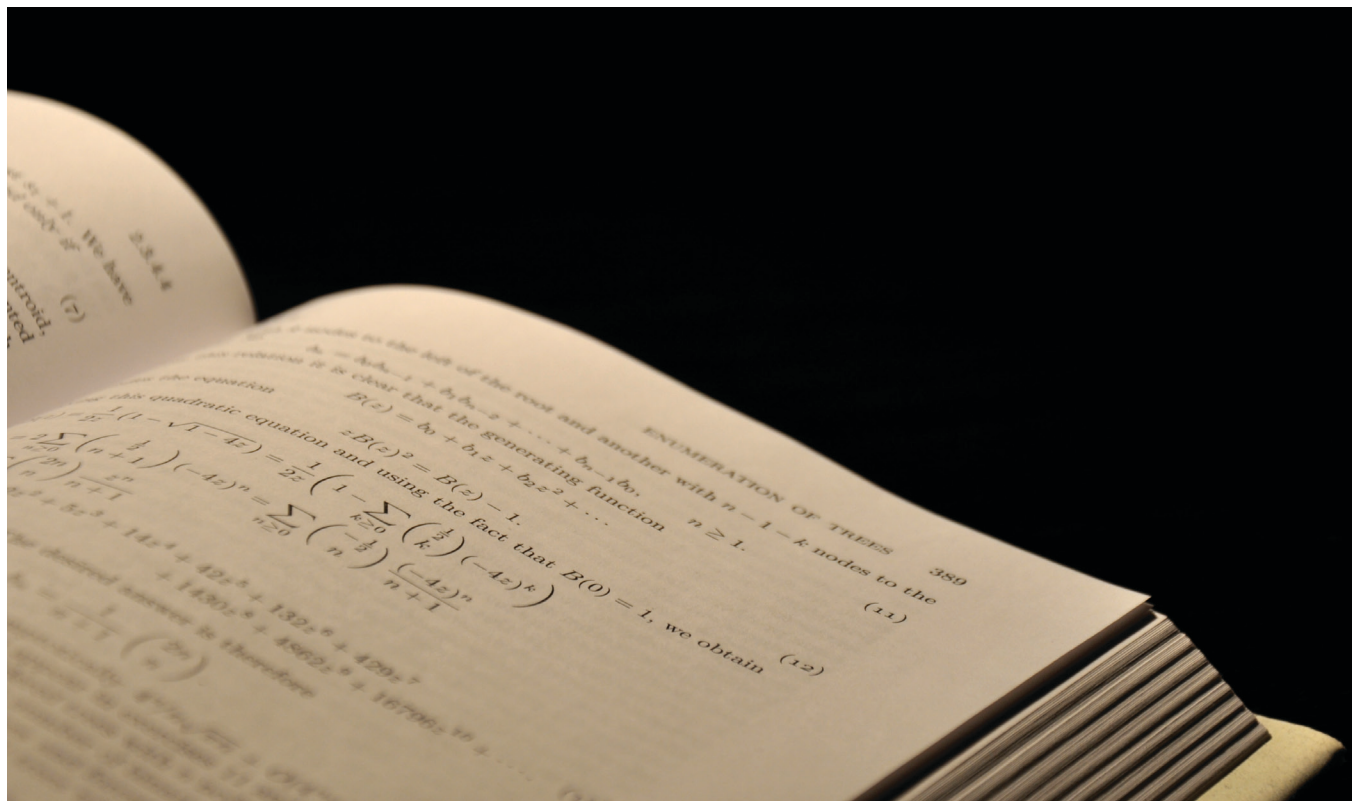
We found these intellectual gatherings quite useful and subsequently decided to make the CS classics group reading a regular activity of our academic environment. Here, I give an overview of the classics we discussed and encourage further reading.

#### Classics Overview

In reading Hoare,<sup>6,7</sup> you learn about the computing industry of the 1960s

and 1970s in Britain. The programming languages community of those years was also well described in the reading. Hoare wrote a more efficient sort algorithm than the one invented by D.L. Shell.<sup>9</sup> When he had the opportunity to hear about the recursive procedures in an ALGOL 60 course, Hoare realized this mechanism is the right way of expressing his new sort algorithm, which is the original QuickSort. The moral of this example is that one should communicate with people to seek better solutions to the problems at hand and extend the existing solutions. His remark on simplification is of high importance as well. A simple, reliable core is critical for a programming language, an operating system, and even for any software product. With this realization, Hoare provides a foundation for the formal proofs of programs by an algebraic assertions-based approach, which is named as “An Axiomatic Basis for Computer Programming.”<sup>7</sup>

Gödel’s undecidability theorem<sup>1</sup> states that any mathematical system containing all the theorems of arithmetic is an incomplete system. This opens the way for Turing to introduce the famous halting problem: There is no general algorithm that can always correctly predict whether a randomly selected computer program will run or not.<sup>11</sup> Before knowing about Gödel and his undecidability theorem, Turing stands out as the most prominent figure in computer science. After you hear about Gödel’s work, you realize Turing is standing on the shoulders of giants. The proof of the undecidability theorem has important implications



for computer science by introducing the Gödel numbering scheme, which introduces unique numbering to each symbol, formula, or proof in the system. This system is the basis of the computer numbering systems that provide unique representation to every programming construct: due to this property, code can be treated as data.

The idea of this unique numbering system can be better explained by the challenge of writing a source program that, when compiled and executed, will produce as output an exact copy of its source. It is a Turing machine SELF that is printing itself. The SELF machine is constructed such that it contains two concatenated machines and one of them is the Gödel number equivalent of the other. Such a self-reproducing program is introduced by Ken Thompson in “Reflections on Trusting Trust”<sup>10</sup> as the most primitive version of today’s trojans. When you see the scientific layers on top of each other like the one presented, you begin to appreciate the real beauty of science and the scientific developments.

By reading Dijkstra independent from his contemporary Hoare you have information about the computing environment of that era. To make a correct assessment of that time

period and the products that were launched, independent but consistent views are required. In this sense, Dijkstra and Hoare’s identical views on ALGOL 60 help us appreciate this programming language. Additionally, the realization of the recursion mechanism by both is spectacular—a good example of the axiom “great minds think alike.”

Dijkstra’s dialogue with his professor cannot be overlooked. Most significant is his realization of the high intellectual challenge of programming and the professor’s encouragement that made him one of the greatest minds of computer programming.<sup>5</sup>

One lesson comes from the huge abstraction capability/potential in-

**Reading CS classics widens your perspective by introducing stable, timeless ideas.**

herent in computer science. Abstraction is extending the viewpoint in a way that the specificities of the problem can be reflected in a better way rather than being vague. The tools we work with can then have vital importance in abstracting. Dijkstra’s comment on computing tools is remarkable in this sense:<sup>2</sup> he states that computing tools have direct influence on the thinking habits of their users. If you constrain yourself with one specific tool, your thinking becomes constrained in the boundaries of this tool. You continue to stay at the same level of thinking as the creator of this tool in accordance with the famous quote by Einstein: “The significant problems we face cannot be solved at the same level of thinking we were at when we created them.”

Donald Knuth is extraordinary with his perspective on computer programming.<sup>3,4</sup> His definition of programming identifies the right balance between conceptual clarity and implementation efficiency. He says: “Programming is the art of telling another human being what one wants the computer to do.”

In understanding the importance of this definition, one should realize it is beyond the traditional definition of the task of telling a computer what to


do. Knuth's viewpoint is more encompassing and is helpful in understanding the diversity and convergence of programming languages. Moreover, it points out an important trade-off between conceptual clarity and implementation efficiency. When the task is to define a job to a computer, low-level instructions are better in terms of execution efficiency. However, people have difficulty in understanding such written code. When you try to describe a task to a human being, you can skip some steps because humans are good at filling in the blanks; machines have difficulty doing this. The best is to compromise: to discuss the task at a high level but in a manner that can be converted into a machine-processable format as indicated by Knuth's prodigious statement.

Knuth's opinions about tools are similarly noteworthy.<sup>8</sup> Like Dijkstra, he thinks the tools we utilize have direct influences on what we accomplish. He puts emphasis on the artistic aspect of programming. According to him, the beauty and aesthetics of tools improves the enjoyment of users and enhances their thinking habits. Com-

binning the assessments of Dijkstra and Knuth, what we (plan to) do is not independent of how we (plan to) do it. The process is a good indicator of the resultant product most of the time.

### Conclusion

Reading CS classics widens your perspective by introducing stable, timeless ideas. You escape the popular themes of your times and evaluate the field from a more literal position. You learn about the qualities that make a person a great scientist. You realize those people are delighted to think over problems. By learning the history of computers and studying the lives and works of eminent computer scientists we all recognize the true merit of being part of such a respectful profession and privileged community.

I hope this Viewpoint raises readers' interest in CS classics, causes CS professionals to revise their reading lists to include these books and articles, and inspires them to further extend their classics library. Time spent on the classics is not wasted but is an investment in your career as a researcher as well as an educator. 

### References

1. Andriilli, S.F. Gödel's Undecidability Theorem. *Applications of Discrete Mathematics*. J.G. Michaels and K.H. Rosen, Eds. McGraw-Hill, 1991.
2. Dijkstra, E.W. The humble programmer. *Commun. ACM* 51, 10 (Oct. 1972), 859–866; DOI: 10.1145/355604.361591.
3. Feigenbaum, E. Donald Knuth: A life's work interrupted. Shustek, L., Ed. *Commun. ACM* 51, 8 (Aug. 2008), 31–35; DOI: 10.1145/1378704.1378715.
4. Feigenbaum, E. The 'art' of being Donald Knuth. Shustek, L., Ed. *Commun. ACM* 51, 7 (July 2008), 35–39; DOI: 10.1145/1364782.1364794.
5. Frana, P. An interview with Edsger W. Dijkstra. T.J. Misa, Ed. *Commun. ACM* 53, 8 (Aug. 2010), 41–47; DOI: 10.1145/1787234.1787249.
6. Hoare, C.A.R. The emperor's old clothes. *Commun. ACM* 24, 2 (Feb. 1981), 75–83; DOI: 10.1145/358549.358561.
7. Hoare, C.A.R. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580; DOI: 10.1145/363235.363259.
8. Knuth, D.E. Computer programming as an art. *Commun. ACM* 17, 12 (Dec. 1974), 667–673; DOI: 10.1145/361604.361612.
9. Shell, D.L. A high-speed sorting procedure. *Commun. ACM* 2, 7 (July 1959), 30–32; DOI: 10.1145/368370.368387.
10. Thompson, K. Reflections on trusting trust. *Commun. ACM* 27, 8 (Aug. 1984), 761–763; DOI: 10.1145/358198.358210.
11. Turing, A.M. I—Computing machinery and intelligence. *Mind* LIX, 236 (1950), 433–460; <http://mind.oxfordjournals.org/content/LIX/236/433.full.pdf+html>

**Selma Tekir** ([selmatekir@iyte.edu.tr](mailto:selmatekir@iyte.edu.tr)) is a postdoctoral instructor in the Department of Computer Engineering at Izmir Institute of Technology in Turkey.

I would like to thank Burcu Külahçoğlu, Murat Özkan, and Serap Şahin for the joyful CS classics meeting we had.

Copyright held by author.

# SIMONS FOUNDATION

## Graduate Fellowships in TCS

Up to 10 Fellowships will be awarded to applicants with a track record of outstanding results in theoretical computer science.

Applicants must be Ph.D. students at a U.S. institution of higher education.

There is a limit of one application per university; please coordinate with the Department Chair

**Application Deadline: May 1, 2012**

### Simons Foundation Program for Mathematics & the Physical Sciences

seeks to extend the frontiers of basic research. The Program's primary focus is on the theoretical sciences radiating from Mathematics: in particular, the fields of Mathematics, Theoretical Computer Science and Theoretical Physics.

For more information on our grants programs visit  
**[simonsfoundation.org](http://simonsfoundation.org)**