Preface

The first spark that eventually led to this book was a conversation between two expert programmers that I witnessed at the NDC Oslo conference in 2014. They were advocates of two different programming languages, Erlang and Haskell, talking about how a programming language can help you write good software. The two experts, both knowledgeable and open-minded speakers at the conference, were simply not able to have a reasonable conversation. The Erlang advocate explained how robust error recovery in Erlang makes software reliable, whereas the Haskell advocate was explaining how type systems can be used to eliminate programming errors. One could not understand why bother with types when you will have failures in your software anyway, whereas the other could not understand how restarting a process could make software correct. The two programmers were talking about a similar kind of programming problem, but each of them was looking at the problem from a different perspective and hence saw very different issues.

Despite the contemporary inspiration, this book is about the history of programming. Once you start looking for the kind of disagreements that I witnessed in Oslo, you find them in many places throughout the history of programming, ranging from the 1968 conference that popularised the term 'software engineering' to the debate in the 1980s about whether formal verification of software is logically possible and the rise of Agile development methods in the first decade of the twenty-first century. Fortunately, the interactions between different ways of thinking about programming do not lead just to disagreements. Once you start looking, you also find that many great advances in programming happened when different ways of thinking contributed to a single concept, starting with the very idea of a programming language.

What is surprising about those interactions is that the different ways of thinking about programming that have been shaping the discipline since the 1950s remain surprisingly stable over its 70-year history. In the 1950s, the different ways of thinking, which I refer to as *cultures of programming*, originated from the individual disciplines that contributed to programming including logic, mathematics, electrical engineering, but also business, psychology and military research. In the early days, this diversity may have been a sign of an immature field, but 70 years later, the existence of the same cultures of programming is instead a sign that programming is, and will remain, an inherently pluralistic discipline.

This book retells the history of programming through the perspective of interactions between five cultures of programming: the hacker; the engineering; the mathematical;

Downloaded from https://www.cambridge.org/core. IP address: 216.73.216.41, on 23 Dec 2025 at 07:40:49, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781009492379.001

the managerial; the humanistic. While the five cultures are often linked to specific communities, the ways of thinking they represent also reappear independently in different contexts. As such, my use of the term culture is figurative rather than literal. I use it to highlight that different cultures of programming have different beliefs, assumptions and practices. Many of the cultures I identify in this book are, in some way, a part of computing folklore. This book exposes their more basic nature. To do so, I follow a number of historical strands from the 1950s to the present day, looking at how the different cultures of programming clash over the nature of programming, but also how they contribute to programming concepts ranging from programming languages and software engineering to structured programming, types, objects and tests.

Thinking about the different cultures of programming sheds a new light on the technical history of programming, but it also provides a new way of thinking about contemporary programming issues. The answer to what is a good program and how to create one differs dramatically when we see programs as mathematical entities, engineered socio-technical systems or media for assisting human thought. Similarly, the answer to how to best teach programming differs when you see programming as applied mathematics, a new form of literacy or a craft that can only be mastered through practice. To borrow a programming term, cultures of programming are a useful abstraction.

This book also points to broader socio-technical issues that are often discussed in the context of the history and philosophy of computing. The history of software engineering cannot be told without mentioning the struggle for managerial control and attempts to develop a more masculine notion of a computer professional. Similarly, the history of interactive programming cannot be told without a reference to the 1960s counterculture and the political ideology of free software. Although I focus on how different cultures of programming shape the way we program, the individual cultures can often be associated with particular social and political views. This book makes it possible to draw connections between the technical and the social history. In doing so, it is more technical than most history books and more historical than most computer science books.

My hopes for the readers of this book are twofold. On the one hand, I hope to provide computer scientists and programming practitioners with a useful perspective through which to look at their field, a much-needed context for understanding ‘how we got here’, but also to provide material that can inspire new ways of thinking. On the other hand, I hope to make the rich technical history of programming legible to those who are not programmers themselves, so that it can serve as a source of interesting episodes for further historical and philosophical reflection. Serving this dual purpose requires some careful balancing. In particular, I will sometimes use modern terminology to describe things from the past if it helps to make the text understandable. One specific challenge in writing this book has been the capitalisation of early programming language names where I generally follow the capitalisation used in recent historical reflections written by their creators.

To emphasise the importance of interactions between the five cultures of programming, each chapter starts with a dialogue between a teacher and students that represent

Downloaded from https://www.cambridge.org/core. IP address: 216.73.216.41, on 23 Dec 2025 at 07:40:49, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781009492379.001

positions of the individual cultures. Although I sometimes adapt quotes from actual historical exchanges, the dialogues are a work of fiction.

In reality, individuals are much less explicit about their beliefs and rarely align with a single culture of programming completely. As we will see, the most interesting historical actors often bridge and contribute to multiple cultures over time. The purpose of the dialogues is to illustrate the clashes between the cultures, the ways in which they structure their arguments and how they can contribute to a single programming concept despite having different basic ideas about the nature of programming.

The students in the dialogues are named after ancient Greek philosophers. This distances the dialogues from actual historical actors and debates, but it provides a mnemonic for remembering which character represents which culture. I am well aware of the many inaccuracies and limitations of the name choices, but I hope you can forgive me those and enjoy the dialogues. The five students we will encounter in the openings of each chapter are:

Pythagoras, representing the *mathematical culture*, is named so for his many mathematical discoveries, but also the mystic view that all things were made of numbers. According to Aristotle, he believed that the principles of mathematics were the principles of all things.

Diogenes, speaking for the *hacker culture*, is named so as a critic of a corrupt society who believed that virtue is better revealed in action than in theory. He is also known for his unconventional lifestyle that included sleeping in a ceramic jar.

Xenophon, a philosopher and a military leader, represents the *managerial culture*. He is named so as the leader who managed a military force of 10,000 Greek mercenaries and for his later writings on its organisation.

Archimedes, speaking for the *engineering culture*, is named so as one of the first thinkers applying mathematics to physical phenomena and for his many practical inventions such as a screw pump and a compound pulley.

Socrates represents the *humanistic culture* for his pursuit of truth and knowledge, for his critical Socratic method of inquiry and his influence on the later humanist movement. He famously believed that the unexamined life is not worth living.

Downloaded from https://www.cambridge.org/core. IP address: 216.73.216.41, on 23 Dec 2025 at 07:40:49, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781009492379.001