II Programming

> It is sometimes difficult to say things that are quite simple.
> —Hutchins (1995, 356)

If part I led, I hope, to interesting insights, it was nonetheless mundane-biased. Although I kept on insisting on the ordinary aspect of ground-truthing—criticizing previous papers, selecting data, defining targets, and so on—I remained very vague about less common practices that those who are not computer scientists generally expect to see in computer science laboratories. For example, where is the mathematics? If the Group managed to define relationships between input-data and output-targets, it certainly formulated them with the help of mathematical knowledge and inscriptions. And where are the cryptic lines of computer code? If the Group managed to first design a web application and later test its computational model on the evaluation set, it must have successfully written machine-readable lists of instructions. If I really want to propose a partial yet realistic constitution of algorithms, do I not need to account for these a priori exotic activities as well? The practices leading to the definition of mathematical models of computation will be the topic of part III. For now, I need to consider *computer programming*, this crucial activity that never stops being part of computer scientists' daily work.

Let us warm up with some basic assertions. Is it not a platitude to say that computer programming is a central activity? Every digital device that takes part in our courses of action required indeed the expert hands of "programmers" or "developers" who translated desires, plans, and intuitions into machine-readable lists of instructions. Banks, scientific laboratories,

high-tech companies, museums, spare part manufacturers, novelists, ethnographers: all indirectly rely on people capable of interacting with computers to assemble files whose content can be executed by processors at electronic speed. If by a mysterious black-magic blow all programmers who make computers compute in desired ways were removed from the collective world, the remaining people would very soon end up yapping around powerless relics like, as Malraux says, *crowds of monkeys in Angkor temples*. The current importance of fast and reliable automated processing for most sectors of activity positions computer programming as an obligatory passage point that cannot be underestimated.

Yet if the courses of action of computer programming are terribly important—without them, there would be no digital tools—their study does not always appear relevant. Most of the individuals of the collective world rightly have other things to do than spending time studying what animates the digital devices with which they interact. Moreover, those who study these individuals—for example, sociologists and social scientists—can also take programming practices for granted as political, social, or economic processes often appear *after* innumerable programming ventures have been successfully conducted. For many interesting activities and research topics, then, it makes perfectly sense *not* to look at how computer programs are empirically assembled.

In other situations, though, the activity of computer programming is more difficult to ignore. Computer scientists and engineers cannot, for example, take this activity for granted as it would imply ignoring an important and often problematic aspect of their work.[1] Unfortunately, as we shall see later, the methods they use to better understand their own practices tend to privilege the evaluation of the results of computer programming tasks rather than the practices involved in the production of these results. Programmers' insights resulting from the analysis of programming tasks thus remain distant from the actions of programming, for which they often remain unaccountable.

But programming practices are also difficult to ignore for *cognitive scientists* who work in artificial intelligence departments: as human cognition is—according to many of them—a matter of computing, understanding how computers become able to compute via the design of programs seems indeed to be a fruitful topic. But just like computer scientists and engineers, cognitive scientists have difficulties with properly accessing and inquiring

into computer programming courses of action. For entangled reasons which I will cover in the following chapter, when cognitivists inquire into what makes programs exist, they cannot go beyond the form "program" that precisely needs to be accounted for. In a surprisingly vicious circle that has to do with the so-called computational metaphor of the mind, cognitivists end up proposing numerous (mental) programs to explain the development of (computer) programs.

Programming practices therefore appear quite tricky: terribly important but at the same time very difficult to effectively study. What makes these courses of action so elusive? Is it even possible to account for them? And if it is, what are their associative properties? And what do these properties suggest? The goal of this part II is to tackle some of these questions. The journey will be long, never straightforward, and sometimes, not developed enough. But let the reader forgive me: as you will hopefully realize, a full historical and sociological understanding of computer programming is a life project of its own. So many things have been said without much being shown! The reasons for dizziness are legitimate, the chances of success infinitesimal; yet, if we really care about these entities we tend to call algorithms, an exploratory attempt to better understand the practices required to make them effectively participate in our courses of action might not be, I hope, completely senseless.

Part II is organized as follows. In chapter 3, I start by retracing how the activity of programming was progressively made invisible before proposing conceptual means to help restore its practicality. I first focus on an important document written by John von Neumann in 1945 that presented computers as input-output devices capable of operating without the help of humans. This initial setting aside of programming practices from electronic computing systems further seemed to depict them as self-sufficient "electronic brains." In the second section of the chapter I present academic attempts to make sense of the incapacity of "electronic brains" to operate meaningfully. As we shall see, for intricate reasons related to the computational metaphor of the mind, I assume that researchers conducting these studies did not manage to properly approach computer programming practices, thus further contributing to their invisibilization. In the last section of the chapter where I progressively try to detach myself from almost everything that has been said about the practice of computer programming, I draw on contemporary work in the philosophy of perception to propose

a definition of cognition as *enacted*. This enactive conception of cognition will further help us fully consider *actions* instead of *minds*. In chapter 4, I build on this unconventional conception of cognition as well as several other concepts taken from *Science and Technology Studies* to closely analyze a programming episode collected within the Lab. The study of these empirical materials makes me tentatively partition programming episodes into three intimately related sets of practices: *scientific* with the alignment of inscriptions, *technical* with the work-arounds of impasses, and *affective* with the shaping of scenarios. The need for constant shifting among these three modes of practices might be a reason why computer programming is a difficult yet fascinating experience. The last section of chapter 4 will be a brief summary.

The Constitution of Algorithms

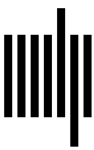
Ground-Truthing, Programming, Formulating

By: Florian Jaton

Citation:

The Constitution of Algorithms: Ground-Truthing, Programming, Formulating
By: Florian Jaton
DOI: 10.7551/mitpress/12517.001.0001
ISBN (electronic): 9780262363235
Publisher: The MIT Press
Published: 2021

The open access edition of this book was made possible by generous funding and support from Arcadia – a charitable fund of Lisbet Rausing and Peter Baldwin



The MIT Press

The open access edition of this book was made possible by generous funding from Arcadia—a charitable fund of Lisbet Rausing and Peter Baldwin.

This book was set in Stone Serif and Stone Sans by Westchester Publishing Services.

Library of Congress Cataloging-in-Publication Data

Names: Jaton, Florian, author. | Bowker, Geoffrey C., writer of foreword.
Title: The constitution of algorithms : ground-truthing, programming, formulating / Florian Jaton ; foreword by Geoffrey C. Bowker.
Description: Cambridge, Massachusetts : The MIT Press, [2020] | Series: Inside technology | Includes bibliographical references and index.
Identifiers: LCCN 2020028166 | ISBN 9780262542142 (paperback)
Subjects: LCSH: Algorithms--Case studies. | Computer programming--Case studies. | Algorithms--Social aspects. | Mathematics--Philosophy.
Classification: LCC QA9.58 .J38 2020 | DDC 518/.1--dc23
LC record available at https://lccn.loc.gov/2020028166