The Craft of Code: Practice and Knowledge in the Production of Software

Pierre Depaz

January 2021

1 Introduction

Software development as a practice has been developing for the past sixty years, emerging as a corollary from the field of computer science. In the 21st century, the importance of code in everyday lives has been highlighted from general-audience journalism[1] to government initiatives[2]. However, the distinction between programming and computer science is a blurry one. While a computer science degree does provides the appropriate formation for a programmer, successful programmers do not necessarily need a computer science background in order to be competent at their jobs. Indeed, while programming has historically emerged as an occupation which responded to the *ad hoc* requirements of computing as a developing field[3], developing further into a codified and identified practice[4], programming has nonetheless often been approached in media- and software- studies with a focus on the phenomenon of computation[5], on the literary[6], or on the social[7], amongst others, rather than on the practice of programming as part of a historical tradition.

Starting from this observation, this article proposes to investigate a different tradition than that of the sciences to highlight the specificities of programming as a practice—the tradition of craftsmanship. Indeed, code isn't just code, but rather a myriad of socio-technical *assemblages* composed of programming languages (e.g. Ruby, C, Julia, JavaScript), operating systems (e.g. Linux, BSD, OSX, Windows) and tools (e.g. IDEs, debuggers, compilers and processors). Those assemblages are in turn used within a cultural context made up of stories, sayings and texts, both from academic and folklore origins. This approach relies on a shift from a conceptual perspective of code, to one in which the word *code* implies varieties of activities[8], in which the variety of practices, self-identifications and narratives from programmers themselves is put at the forefront. The main distinction this article starts with is between computer scientist, as the theoretical approach to field of computation and the software developer, practical implementer, immersed in the practice of writing and reading code.

While links between craftsmanship and programming have existed as self-proclaimed ones by programmers themselves, as well as by academics and writers [9, 10], they have not yet been elucidated under specific angles. Indeed, craftsmanship as such is an ever-fleeting phenomenon, a practice rather than a theory, in the vein of Michel De Certeau's *tactics*, bottom-up actions designed and implemented by the users of a situation, product or technology as opposed to *strategies*[11], in which ways of doing are prescribed in a top-down fashion. It is this practical approach that this article chooses, the informal manners and standards of working, in order to provide an additional, cultural studies perspective, to its media and software studies counterparts.

A comparative approach of a broad mode of economic and cultural activity (craftsmanship) with a narrower technical know-how (software development) asks us first to verify to what extent such an approach is even valid. How, then, is the designation by software developers of their own practice as craftsmanship relevant? Where is the comparison a productive one, and where does it show its limits? How can such comparison enrich both our understanding of code as a practice, as well as craftsmanship practices within the highly networked environment that has become the backdrop of the 21^{st} century? Particularly, how does it re-present processes of knowledge acquisition and aesthetic judgment?

The article proceeds in a comparative fashion, mobilizing texts about craftsmanship as well as sources from the field of programming, describing programmers' self-identification with craftsmanship. From those, I analyze how those references by programmers enter into a productive dialogue with our historical and cultural conception of craftsmanship. To do so, I approach these questions through three different, contiguous topics. First, the focus is set on the historical unfoldings of craftsmanship and software development, showing parallels. After inquiring into the modes of organization and the economical development of both craftsmanship and software development, this section focuses on a particular comparison: that of programming with building, and in particular its relationship with architecture. Second, we shift our focus from a broad view to the specific practices of knowledge acquisition and production. There, we highlight the similarities in terms of tacit and personal knowledge, as well as the means of education and information available both to traditional craftspeople and software developers. This, in turn allows us to discuss the differences of learning environments by taking into account the networks of environment that have developed exponentially with the Internet. Finally, the third section turns to aesthetic judgment of the crafted product. Building on discussions of beautiful craft and beautiful code, the focus here is on the standards which allow practicioners to ascribe beauty to a piece of software; particularly, this section discusses the materiality of code not in terms of bits, bytes and languages, but rather as a material that is worked with and worked through.

These incursions in the practices of software development via the lens of craftsmanship as a cultural practice concludes on the productive differences and similarities in terms of knowledge circulation and materialities when programming code; ultimately resulting in new understandings of both craftsmanship and software development as mutually influencing practices.

2 Parallels

This section starts by providing an overview of the various perspectives and realities on craftsmanship, starting from the Late Middle-Ages until the 20^{th} century. This allows us to highlight some initial important features of craftsmanship: social organization, the nature of work and the transmission of knowledge. Jumping to the history of software development, starting in the mid-20^{th} century, it looks at the claims that programmers make about themselves in relation with the term and concept of craftsmanship. By examining formal and informal texts, we focus on the fact that software developers ground their practice in passion, know-how and myths. Finally, we inspect the place of architecture, both in historical craftsmanship and contemporary software development, in order to qualify further the relationship between design and implementation in those two fields.

2.1 Craftsmanship through the ages

Craftsmanship in our contemporary discourse seems most tied to a retrospective approach: it is often qualified as that which was *before* manufacture, and the mechanical automation of production[12]. So while the practice of developing a skill in order to build something with a functional design has been considered at its apex of craftsmanship in Western late Middle-Ages, it should be noted here that non-Western craftsmanship are as equally rich and unique as their Western counterparts, for instance in China[13] and Japan[14]; however, these lie beyond the immediate scope of this article. Following Sennett, we will use his definition of craftsmanship as *hand-held, tool-based, intrinsically-motivated work which produces functional artefacts, and in the process of which is held the possi-*

bility for unique mistakes[9].

Late Middle-Ages craftsmanship stands out as such for a couple of reasons: their socio-economic organization, and their relationship to knowledge. First and foremost, craftspeople were indissociable from the guilds they belonged to[15]. As tightly-knit communities, they exhibited strong cohesion: vertically, between a master and their apprentices, and horizontally, between equal practicioners, enforcing a uniform quality control assurance and price management (Managerial Techniques by Wolek). This cohesion, in turn, has limited the amount of individual fame and glory that craftspeople could accumulate, as compared to fine-artists[16].

A key aspect of the craftsmanship of this time is the relationship that they maintained with explicit, formalized standards. While various crafts did include specific lexical fields to describe the details of their trade[17], usually compiled into glossaries, the standards for quality were less explicit. Cennino Cennini, in his *Libro dell'arte*, one of the first codexes to map out technical know-how necessary to a painter in the early Renaissance, lays out both practical advice on specific painting techniques, but does not explicitly lay out how to make something *good*[18]. Further work, at the eve of the Industrial Revolution, continued on this intent to formalize the practice of craftsmanship[19]. In this sense, quality work is rooted in implicit knowledge: a good craftsman knows a quality work when they see it[9].

Another component of craftsmanship is its alleged incompatibility with manufacture[20, 21]. However, studies have shown that the craftsmanship, rather than standing at the strict opposite of the industrial[22], has been integrated into the process of modern industrialization. The practice of the craftsman, then, integrates into the design and operation of machines and industrial-scale organizations, informing ways of making in our contemporary world[23, 24].

These characteristics of tight and rigid communities, implicit knowledge, and ambiguous separation with design, framing the foundation of a desire for good work are particularly highlighted in the field of the built environment, and later in the development of architecture. Before examining how such a field has a connection to software development, we take a look at programming's emergence as a field of craft.

2.2 Software developers as craftsmen

Computer programming as an activity came to be as an offshoot of computer science, perhaps best illustrated by the collaboration of Charles Babbage and Ada Lovelace on The Analytical Engine, the prototype of the modern computer. With Babbage acting as the overall designer, Lovelace was key in practically implementing some of the mathematical formulas which The Analytical Engine was built to solve. What we see here is a dyad of work, distinguished between design and conception on one side, and implementation and practice on the other side. These two approaches are echoed throughout the early days of programming (1950s-1970s), with programmers becoming distinct from computer scientists by their approach to the problem (they'd rather write code on a terminal than write algorithms on a piece of paper) and by their background (trained as scientists but more comfortable with tinkering)[25]. In particular, the group of computer enthusiasts described as hackers developed organizational features similar to their historical counterparts: work was being done on distinct topics and fields in different geographic locations (Stanford, MIT, Bell Labs)[26], emphasis was put on engagement with tools (steaphenson), inquiring into peers' work[27] and later formalized into bottom-up archives[1]. Additionally, little accountability was required when it came to design explicitness. As examples, both the UNIX operating system and the TCP/IP protocol were originally realized without overarching supervision and without extensive ongoing documentation[28, 26].

As computer science solidified as a distinct field in the 1960s[29], there

[1]The most famous of which is the Jargon File, later to be published as the The New Hacker's Dictionary: `http://www.catb.org/jargon/html/`

was a process of formalizing the hitherto *ad hoc* techniques of programming computers. As a response to the myth of carefully hand-made code[2] and unconstrained approaches to writing code came the structured programming approach, initally proposed by E. W. Djisktra[4]. With the operating system and the personal computer revolutions, access to tools became widespread, and transformed tightly communities into a global network of exchange, first via Usenet, then through the Web. Inquiries into the relationship of craftsmanship with programming started to take place in the mid-1970s from an educational perspective[30], from an organizational perspective[31] and an inter-personal perspective[32], and culminated with the publication of several trade books[33, 34], explicitly connecting the craft of programming with previous craft activities, and emphasizing the need for intrinsic motivation and the aim of a job well-done[35, 36].

Comparisons of software development with craftsmanship are abundant, and relate then mostly to the relationship between unstructured practice and formalized theory; as such, it is used to self-categorize programmers as skilled makers rather than passive thinkers[3].

2.3 The case of architecture

The field of architecture helps us tie these two traditions together a little more explicitly. Architecture as a field and the architect as a role have been solidified during the Renaissance[37], consecrating a separation of abstract design and concrete work, in which the craftsman is relegated to the role of executioner, until the arrival of civil engineering and blueprints overwhelmingly formalized the discipline.

The classical architect, here, serves as the counterpart to the computer

[2]See The Story of Mel, A Real Programmer, a folktale of early programmers: `https://www.cs.utah.edu/~elb/folklore/mel.html`

[3]See code monkey: `http://www.techopedia.com/definition/31469/code-monkey`

scientist, except in an inverse relation: the architect emerged from centuries of hands-on work, while the computer scientist (formerly known as mathematician) was first to a whole field of practicioners as programmers, followed by a need to regulate and structure those practices. Different sequences of events, perhaps, but nonetheless mirroring each other. On one side, construction work without an explicit architect, under the supervision of bishops and clerks, did indeed result in significant results (Notre Dame de Paris, Basilica of Sienna). On the other side, letting go of structured and restricted modes of working characterizing computer programming up to the 1980s resulted in a comparison described in the aptly-named *The Cathedral and the Bazaar*. This essay described the Linux project, the open-source philosophy it propelled into the limelight, and how the quantity of self-motivated workers without rigid working structures (which is not to say without clear designs) can result in better work than if made by a few, select, highly-skilled individuals[26, 38].

What we see, then, is a similar result: individuals can cooperate on a long-term basis out of intrinsic motivation, and without clear, individual ownership of the result; a parallel seen in the similar concepts of *collective craftsmanship* in the Middle-Ages and the *egoless programming* of today[31]. The further sections will investigate how such a phenomena of building complex structures through horizontal networks is possible, from both epistemological and aesthetic perspectives.

3 Knowledge acquisition and production

3.1 Bus factor and implicit knowledge

The problem of knowledge in software development can be examplified by the ”bus factor”[4]. It describes the risk of crucial information being lost due to the disappearance or incapacity of one of the programmers of the

[4] `https://en.wikipedia.org/w/index.php?title=Bus_factor`

project, and aims at the problem of *essential complexity*[5]. Given the inherent complexity of programming as a task, along with the compulsive behaviours sometimes exhibited by programmers as a by-product of intrinsic motivation[39], the gap between design and implementation—the domain of the craftsman—often relies on tacit knowledge[40].

Explicit knowledge, in programming as in most disciplines, is carried through books, academic programs and, more recently, web-based content that is either structured (e.g. MOOCs, Codeacademy, Khan Academy) or unstructured (e.g. blog posts, forums, IRC channels), but both seem to be insufficient to reach an expert level[41]. As demonstrated by a popular comic, the road to good code is unclear, particularly when communicated in such a highly-formal language as diagramming. Given the fact that an individual can become a programmer through non-formal training—as opposed to, say, an engineer or a scientist—, the learning process must include implicit knowledge.

3.2 Apprentices and masters

The acquisition of such implicit knowledge in craftsmanship takes place in two different ways: the apprentice-master relationship, and the act of copying. First comes the apprentice-master relationship, in which a learner starts by imitating the way of working of the master (Sennett), resulting in a *teaching by showing*, where important aspects of the craft are being demonstrated to the apprentice by a more experienced practicioner, rather than formalized and learned *a priori* of the practice. Sometimes, this relationship to a master is implemented explicitly through practices such as pair programming[42] or corporate mentorship programmings (IBM's Master programmer initiative). Other times, it is re-interpretated through fictional accounts designed to impart wisdom on the readers, and taking inspiration from Taoism and Zen[43, 44]. From higher-level programming

[5]See the *No Silver Bullet* essay in *The Mythical Man-Month*, op.cit.

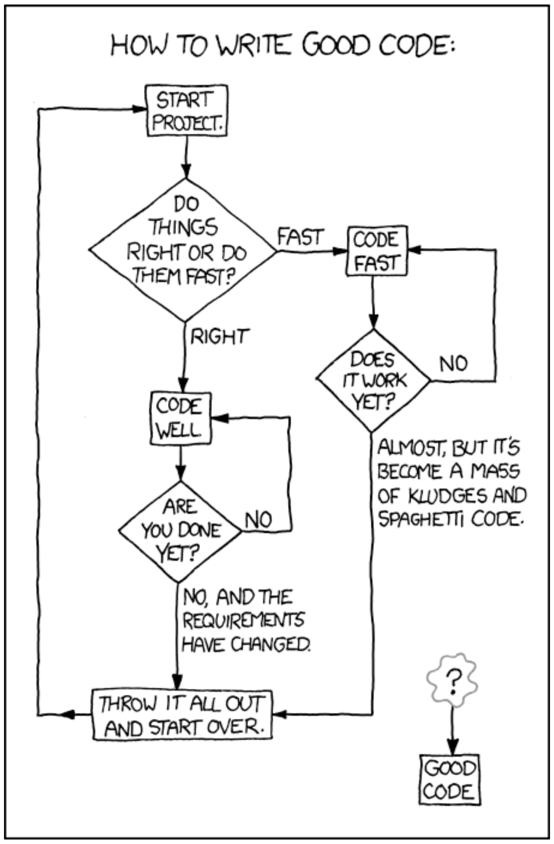
HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

Figure 1: Source: https://xkcd.com/844/

wisdom featuring leading programmers such as Marvin Minsky and Donald Knuth, this sort of informal teaching by showing has been implemented in various languages as a practical learning experience[6]. Without the presence of an actual master, the programming apprentice nonetheless takes the program writer as their master to achieve each of the tasks assigned to them. The experience historically assigned to the master craftsman is delegated into the code itself, containing both the problem, the solution to the problem and hints to solve it, straddling the line between formal exercises and interactive practice.

Code's ability to be copied and executed on various machines provides a counterpoint to the argument of software as craftsmanship in terms of knowledge transmission. Traditionally, since craftsmanship has been understood as that which is done by hand, and since craftsmen were working with unique artefacts (i.e. no artefact can be perfectly copied), copying someone else's realization was physically inconceivable. The realm of software, on the opposite, relies heavily on the technical affordance of code to be duplicated, uploaded, downloaded and executed on multiple platforms through source code files[45]. The first immediate consequence of this is the ability for all to inspect and use source code, both on an institutional level (as guaranteed by projects such as GNU[7]), and on a vernacular level (as enabled by Web 2.0 platforms such as StackOverflow and GitHub). Even though the ability to perfectly copy anyone else's work became widely available to programmers, the difference between amateur and expert programmers lied in the extent to which they indeed blindly copy external code, or write their own, inspired by the external code[8].

Practices from Eastern craftsmanship further qualify these essentially different approaches to copying. *Moxie*, a Chinese term for copying and

[6]See, for instance: `http://rubykoans.com/`

[7]See: `https:gnu.org`

[8]See the discussions on `https://softwareengineering.stackexchange.com/questions/36978`

practice, is a key concept to understand how an apprentice can equal his master through thougtful replication [46], an approach equally present in Japanese crafts histor[14]. Here again, manually copying from established quality work to seize their elusive essence is an essential aspect to craftsmanship.

3.3 The problem with copying

If implicit knowledge can be acquired through a showing and copying of code, software development as a craft presents an additional dimension to this, a sort of *piecemeal knowledge*. Best represented by Stack Overflow, a leading question and answer forum for programmers, on which code snippets are made available as part of the teaching by showing methodology, this piecemeal knowledge can both help programmers in solving issues as well as deter them in solving issues *properly*[47]. Code as such is freely and easily accessible as piecemeals, but often lacks the essential context.

So while programmers are to acquire implicit knowledge through a process of learning by doing (realizing koans, coding small projects, re-using copied code), we now need to assess how much of it happens through observing. Implied in the apprentice-master relationship is that what is observed should be of *good quality*; one learns through ones own mistakes, and through ones presentation with exmaples of good work. Coming back to the relationship between architecture and software development, Christopher Alexander asks, in the preface of Richard P. Gabriel's *Patterns of Software*[48],

> *For a programmer, what is a comparable goal? What is the Chartres of programming? What task is at a high enough level to inspire people writing programs, to reach for the stars?*

If a craftsman learns their trade by comparing their work with work of a higher quality (either their master's, or publicly available works, assembled as a canon[49]), the programmer is faced with a different problem: a lot of examples, but a few good ones. Copyright stands in the way of pedagogical copying. With software becoming protectable under copyright laws in the 1980s[50], great works of programming craft became unacessible to programmers, despite the value they would bring in knowledge acquisition[51]. One of the most famous examples is *Lions' Commentary on UNIX 6th Edition, with Source Code* by John Lions, an annotated edition of the UNIX source code, which was circulated illegaly in classrooms for twenty years before its official publication was authorized by the copyright owners[52].

With implicit knowledge being a key component in both disciplines, its manifestation through the copying of source code in software development is hampered either by decontextualized, uploaded code snippets or by copyrighted protection on works, leading to a lack in an established canon of great works. Nonetheless, the other advice given to beginner programmers—practice[9]—hints at another aspect: direct engagement with code.

4 Material aesthetics

At the heart of knowledge transmission and acquisition stands the *practice*, and inherent in the practice is the *good practice*, the one leading to a beautiful result. This section investigates the aesthetics of code within the broader context of the aesthetics of craftsmanship, highlighting code's specificity as a material.

4.1 The beauty of a thing well-made

A traditional perspective is that of the motor skills, with dexterity, care and experience as essential features of a craftsman's ability to realize some-

[9]`https://quora.com/What-are-some-of-the-best-ways-to-learn-programming`

thing beautiful[53], along with self-assigned standards of quality[54, 9]. These qualitative standards which, when pushed to their extreme, result in a craftsperson's *style*, are to be gained through practice and experience, rather than by explicit measurements[54] [10]. Two things are concerned here: tools and materials[54]. A craftsperson should have a deep, implicit knowledge of both, what they use to manipulate (chisels, hammers, ovens, etc.) as well as what they manipulate (stone, wood, steel, etc).

This relationship to tools and materials is expected to have a relationship to *the hand*, and at first seems to exclude the keyboard-based practice of programming. But even within a world in which automated machines have replaced hand-held tools, Osborne writes:

> *In modern machine production judgement, experience, ingenuity, dexterity, artistry, skill are all concentrated in the programming before actual production starts.[53]*

He opens here up a solution to the paradox of the hand-made and the computer-automated, as programming emerges from the latter as a new skill. This very rise of automation has been criticized for the rise of a Osborne's "soulless society"[53], and has triggered debates about authorship, creativity and humanity at the cross-roads between artificial intelligence and artistic practice[55]. One avenue out of this debate is human-machine cooperation, first envisioned by Licklider and proposed throughout the development of Human-Computer Interaction[56, 57]. If machines, more and more driven by computing systems, have replaced traditional craftsmanship's skills and dexterity, this replacement can nonetheless suggest programming as a distinctly 21st-century craftsmanship, as well as other forms of cratsmanship-based work in an information economy.

[10]See Pye's account of craftsmanship, and his intent to make explicit the question of quality craftsmanship and *"answer factually rather than with a series of emotive noises such as protagonists of craftsmanship have too often made instead of answering it."*

4.2 Code as material

Beautiful code, code well-written, is indeed an integral part of software craftsmanship[58]. More than just function for itself, code among programmers can, and should be held to beauty standards[59]. Such standards are another relationship with traditional craftsmanship—form following function.

A craftsman's material consciousness is recognized by the anthropomorphic qualities ascribed by the craftsman to the material[9]. In the case of code, adjectives such as ”clean”, ”elegant”, ”smelly” occur over and over in online discussions of programmers. Clean code, elegant code, are indicators not just of the awareness of code as a raw material that should be worked on, but also of the necessities for code to exist in a social world. As software craftsmen assemble in loose hierarchies to construct software, the aesthetic standard is *the respect of others*[60].

Another unique feature of software craftsmanship is its blending between tools and material: code, indeed, is both. This is, for instance, represented at its extreme by languages like LISP, in which functions and data are treated in the same way[61]. In that sense, code is a material which can be almost seamlessly converted from information to information-*processing*, and vice-versa. Disregarding for now the very real impact of computing on the environment[62], code as a material is perhaps the only non-finite material that craftspeople can work with—along with words.

Code, then, is not just an overarching, theoretical concept which can only be reckoned with in the abstract, but also the very material foundation from which the reality of software craftsmanship evolves. An analysis of computing phenomena, from software studies to platform studies, should therefore take into account the close relationship to their material that software developers can have. As Fred Brooks put it,

> *The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, cre-*

ating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures.[31]

4.3 The implications of beautiful code

So while there are arguments for developing a more rigorous, engineering conception of software development[25], a crafts ethos based on a materiality of code holds some implications both for programmers and for society at large.

On the one side, since craftsmanship aesthetic standard relies on the process and the immediate usage of the product, little attention might be given to the long-term consequences of such a product. When computing systems start to get entangled with complex domains such as culture[63] or education[64], programmers play a significant role in the development of these systems[65], and their intrinsic motivation to work with code for its own sake without a broader perspective might lead to undesired outcomes—a situation in which the function of the product is no longer beautiful.

On the other side, this engagement with code-as-material opens up possibilities for the acknowledgement of a different moral standard. As Pye puts it,

> *[...] but still the quality of the result is clear evidence of competence and assurance, and it is an ingredient of civilization to be continually faced with that evidence, even if it is taken for granted and unremarked.[54]*

If most commentators on the history of craftsmanship, following Ruskin, lament the disappearance of a better, long-gone way of doing things, before computers came to automate everything, locating software as a contemporary iteration of the age-old ethos of craftsmanship opens-up the possibility for a more conscious, careful and diligent way of building the future.

5 Conclusion

After qualifying a relationship to craftsmanship that has been claimed by software developers since the mid-1980s by a parallel with craftsman building and programming work, we've shown how these practicioners evolve in a somewhat different environment. First, their relation to knowledge and the need to acquire and transfer tacit knowledge has been limited by both the disappearance of strict master-apprentice relationships, and the removal from the public domain of some of software development's greatest achievements. Second, their relationship to code, as both materials and tools, while holding intrinsically the possibility for better, and more moral work, re-states the importance of code as a product which can be studied, learned and perfected.

References

[1] Paul Ford. What Is Code? If You Don't Know, You Need to Read This. *Bloomberg.com*, 2015.

[2] Cameron Wilson. Hour of code—a record year for computer science. *ACM Inroads*, 6(1):22, February 2015.

[3] Wendy Hui Kyong Chun. On Software, or the Persistence of Visual Knowledge. *Grey Room*, 18:26–51, January 2005. Publisher: MIT Press.

[4] Edsger W. Dijkstra. Chapter I: Notes on structured programming. In *Structured programming*, pages 1–82. Academic Press Ltd., 1972.

[5] David M. Berry. *The Philosophy of Software: Code and Mediation in the Digital Age*. Palgrave-Macmillan, 2011.

[6] N. Katherine Hayles. *My Mother Was a Computer: Digital Subjects and Literary Texts*. University of Chicago Press, March 2010. Google-Books-ID: IwaRyOZfBzgC.

[7] Adrian Mackenzie. *Cutting Code: Software and Sociality*. Peter Lang, 2006. Google-Books-ID: 083BUgMnLKQC.

[8] Brian Hayes. Cultures of Code, February 2017.

[9] Richard Sennett. *The Craftsman*, volume 1. Yale University Press, New Haven, CT, 2009.

[10] Vikram Chandra. *Geek Sublime: The Beauty of Code, the Code of Beauty*. Graywolf Press, September 2014.

[11] Michel de Certeau, Luce Giard, and Pierre Mayol. *L'invention du quotidien*. Gallimard, 1990. Google-Books-ID: GAwEAQAAIAAJ.

[12] Daniel V. Thompson. The Study of Medieval Craftsmanship. *Bulletin of the Fogg Art Museum*, 3:3–8, 1934. Publisher: [Harvard University Art Museums, Harvard Art Museums, The President and Fellows of Harvard College].

[13] Ganlin Zhang, Zhou Cheng, and Qingli Wang. Jingdezhen's Ceramic Civilization: the Past and Today. pages 9–14. Atlantis Press, September 2015. ISSN: 2352-5398.

[14] Brenda G. Jordan and Victoria Louise Weston. *Copying the Master and Stealing His Secrets: Talent and Training in Japanese Painting*. University of Hawaii Press, January 2003. Google-Books-ID: TMCH-pmDXUeIC.

[15] Antony Black. *Guilds and Civil Society in European Political Thought from the Twelfth Century to the Present*. Methuen, 1984. Google-Books-ID: oQMOAAAAQAAJ.

[16] Daniel Varney Thompson. *The Materials and Techniques of Medieval Painting*. Courier Corporation, January 1956. Google-Books-ID: I1DFuGQeG10C.

[17] Associate Conservator Department of Decorative Arts and Sculpture Conservation Jane Bassett, Jane L. Bassett, Peggy Fogelman, David A. Scott, Getty Conservation Institute (Los Angeles Calif.), and Ronald C. Schmidtling. *The Craftsman Revealed: Adriaen de Vries*. Getty Publications, 2008. Google-Books-ID: E8oxCwAAQBAJ.

[18] Cennino Cennini. *The Craftsman's Handbook*. Courier Corporation, April 2012. Google-Books-ID: 4Z2jAQAAQBAJ.

[19] John R. Pannabecker. Diderot, the Mechanical Arts, and the Encyclopdie: In Search of the Heritage of Technology Education. *Journal of Technology Education*, 6:45–57, 1994.

[20] John Ruskin. *The seven lamps of architecture. With illustrations drawn by the author*. London Waverley Book Co, 1920.

[21] George Sturt. *The Wheelwright's Shop*. Cambridge University Press, Cambridge ; New York, revised ed. edition edition, January 1963.

[22] Matthew L. Jones. *Reckoning with Matter: Calculating Machines, Innovation, and Thinking about Thinking from Pascal to Babbage*. University of Chicago Press, Chicago ; London, 1st edition edition, November 2016.

[23] Robert B. Gordon. Who Turned the Mechanical Ideal into Mechanical Reality? *Technology and Culture*, 29(4):744–778, 1988. Publisher: [The Johns Hopkins University Press, Society for the History of Technology].

[24] David McGee. From Craftsmanship to Draftsmanship: Naval Architecture and the Three Traditions of Early Modern Design. *Technology and Culture*, 40(2):209–236, 1999. Publisher: [The Johns Hopkins University Press, Society for the History of Technology].

[25] Nathan L. Ensmenger. *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*. The MIT Press, Cambridge, Mass., August 2012.

[26] Eric S. Raymond. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. "O'Reilly Media, Inc.", 2001. Google-Books-ID: W2t2d2KP6HsC.

[27] Steven Levy. *Hackers: Heroes of the Computer Revolution - 25th Anniversary Edition*. "O'Reilly Media, Inc.", May 2010. Google-Books-ID: mShXzzKtpmEC.

[28] Peter Seibel. *Coders at Work: Reflections on the Craft of Programming*. Apress, September 2009.

[29] Matti Tedre. The development of computer science: a sociocultural perspective. In *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, Baltic Sea '06, pages 21–24, New York, NY, USA, February 2006. Association for Computing Machinery.

[30] Edsger W. Dijkstra. "Craftsman or Scientist?". In Edsger W. Dijkstra, editor, *Selected Writings on Computing: A personal Perspective*, Texts and Monographs in Computer Science, pages 104–109. Springer, New York, NY, 1982.

[31] Frederick Phillips Brooks and Frederick P. Brooks Jr. *The Mythical Man-month: Essays on Software Engineering*. Addison-Wesley Publishing Company, 1975. Google-Books-ID: gWgPAQAAMAAJ.

[32] Gerald M. Weinberg. *The Psychology of Computer Programming*. Dorset House Pub., 1998. Google-Books-ID: j_MJAQAAMAAJ.

[33] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, August 2008. Google-Books-ID: _i6bDeoCQzsC.

[34] Mike Hendrickson and Pete McBreen. *Software Craftsmanship: The New Imperative*. Addison-Wesley Professional, 2002. Google-Books-ID: C9vvHV1lIawC.

[35] Dave Hoover and Adewale Oshineye. *Apprenticeship Patterns: Guidance for the Aspiring Software Craftsman*. "O'Reilly Media, Inc.", October 2009. Google-Books-ID: I3xFAoZT_5AC.

[36] Pete Goodliffe. *Code Craft: The Practice of Writing Excellent Code*. No Starch Press, 2007. Google-Books-ID: i4zCzpkrt4sC.

[37] N. Pevsner. The Term 'Architect' in the Middle Ages. *Speculum*, 17(4):549–562, 1942. Publisher: [Medieval Academy of America, Cambridge University Press, University of Chicago Press].

[38] Erik Henningsen and Håkon Larsen. The Joys of Wiki Work: Craftsmanship, Flow and Self-externalization in a Digital Environment. In Ragnar Audunson, Herbjorn Andresen, and Cicilie Fagerlid, editors, *Libraries, Archives and Museums as Democratic Spaces in a Digital Age*, pages 345–362. De Gruyter Saur, September 2020.

[39] Joseph Weizenbaum. *Computer Power and Human Reason: From Judgment to Calculation*. W H Freeman & Co, San Francisco, 1st edition edition, March 1976.

[40] Harry Collins. *Tacit and Explicit Knowledge*. University of Chicago Press, June 2010. Google-Books-ID: ONzRaIXOtEMC.

[41] Simon P. Davies. Models and theories of programming strategy. *International Journal of Man-Machine Studies*, 39(2):237–267, August 1993.

[42] Laurie Williams and Robert R. Kessler. *Pair Programming Illuminated*. Addison-Wesley Professional, 2003. Google-Books-ID: LRQhdIrKNE8C.

[43] Geoffrey James. *The Tao of Programming*. InfoBooks, 1987. Google-Books-ID: idkNAAAACAAJ.

[44] Eric S. Raymond and Guy L. Steele. *The New Hacker's Dictionary*. MIT Press, 1996. Google-Books-ID: g80P_4v4QbIC.

[45] Lev. Manovich. *The language of new media*. MIT Press, Cambridge, MA, 2001.

[46] Eva Kit Wah Man. Influence of Global Aesthetics on Chinese Aesthetics: The Adaptation of Moxie and the Case of Dafen Cun. In Eva Kit Wah Man, editor, *Issues of Contemporary Art and Aesthetics in Chinese Context*, Chinese Contemporary Art Series, pages 95–103. Springer, Berlin, Heidelberg, 2015.

[47] C. Treude and M. P. Robillard. Understanding Stack Overflow Code Fragments. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 509–513, September 2017.

[48] Richard P. Gabriel. *Patterns of Software: Tales from the Software Community*. Oxford University Press, 1998. Google-Books-ID: uwFLPwAACAAJ.

[49] Paul Taylor. Patterns as Software Design Canon. *ACIS 2001 Proceedings*, January 2001.

[50] Ralph Oman. Computer Software as Copyrightable Subject Matter: Oracle V. Google, Legislative Intent, and the Scope of Rights in Digi-

tal Works. *Harvard Journal of Law and Technology*, 31(Special Issue Spring 2018):639–652, 2018.

[51] Richard P. Gabriel and Ron Goldman. Mob Software: The Erotic Life of Code, 2001.

[52] John Lions. *Lions' Commentary on UNIX 6th Edition with Source Code*. Peer-to-Peer Communications, 1996. Google-Books-ID: OIZ3QgAACAAJ.

[53] Harold Osborne. The Aesthetic Concept of Craftsmanship. *British Journal of Aesthetics*, 17(2):138, 1977. Publisher: Oxford University Press.

[54] David Pye. *The Nature and Art of Workmanship*. Herbert Press, illustrated edition edition, July 2008.

[55] Marian Mazzone and Ahmed Elgammal. Art, Creativity, and the Potential of Artificial Intelligence. *Arts*, 8(1):26, March 2019. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[56] J. C. R. Licklider. Man-Computer Symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1(1):4–11, March 1960. Conference Name: IRE Transactions on Human Factors in Electronics.

[57] Jonathan Grudin. From Tool to Partner: The Evolution of Human-Computer Interaction. *Synthesis Lectures on Human-Centered Informatics*, 10(1):i–183, December 2016. Publisher: Morgan & Claypool Publishers.

[58] Andy Oram and Greg Wilson, editors. *Beautiful Code: Leading Programmers Explain How They Think*. O'Reilly Media, Beijing ; Sebastapol, Calif, 1st edition edition, July 2007.

[59] Erik Pineiro. *The aesthetics of code : on excellence in instrumental action*. PhD thesis, KTH, Superseded Departments, Industrial Eco-

nomics and Management., 2003. Publisher: Industriell ekonomi och organisation.

[60] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs - 2nd Edition*. Justin Kelly, 1979. Google-Books-ID: MXZQAwAAQBAJ.

[61] John McCarthy, Michael I. Levin, Paul W. Abrahams, Massachusetts Institute of Technology Computation Center, and Daniel J. Edwards. *LISP 1.5 Programmer's Manual*. MIT Press, 1965. Google-Books-ID: 68j6lEJjMQwC.

[62] Patrick Kurp. Green computing. *Commun. ACM*, 51(10):11–13, October 2008.

[63] Nick Seaver. Captivating algorithms: Recommender systems as traps. *Journal of Material Culture*, 24(4):421–436, December 2019. Publisher: SAGE Publications Ltd.

[64] Carlo Perrotta. Programming the platform university: Learning analytics and predictive infrastructures in higher education. *Research in Education*, page 0034523720965623, October 2020. Publisher: SAGE Publications Ltd STM.

[65] Pierre Lévy. *De la programmation considérée comme un des beaux-arts*. Textes à l'appui. Anthropologie des sciences et des techniques. Éd. la Découverte, Paris, 1992.