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

Pierre Depaz

January 2021

1 Introduction

Code has often been examined through computation[1], humanities [2], or social sciences[3], rather than as a result of the practice of programming. This article proposes to investigate the specificities of writing code within the tradition of craftsmanship. Indeed, code is a myriad of socio-technical *assemblages* composed of programming languages, operating systems and tools. These in turn exist within a cultural context made up of stories and texts, both from academic and folk origins. This article relies on a shift from an abstract conception of code, to one in which a variety of actors[4], self-identifications and narratives from programmers themselves is put at the forefront. The main distinction we start with is between the theoretical computer scientist, and the practical software developer, focused on writing and reading code.

Links between craftsmanship and programming have existed as self-proclaimed ones by programmers themselves, as well as by academics [5], but have not yet been elucidated under specific angles. Michel De Certeau describes the concept of *tactics*, bottom-up actions designed and implemented by the users of a situation, product or technology as opposed to *strategies*[6], in which ways of doing are prescribed in a top-down fashion.

It is two of these tactics that we investigate here—knowledge and beauty in programming.

To do so, we start from a parallel with medieval architecture, then move on to the place of implicit knowledge in process of programmers learning their craft, and conclude on the reality of code as a material. These incursions in programming as craftsmanship conclude on the possibility of a specific craft of the twenty-first century.

2 Parallels

2.1 Craftsmanship through the ages

Craftsmanship is often seen as that which was *before* manufacture, and as *tool-based, intrinsically-motivated work which produces functional artefacts*[5]. It stood out in the late Middle-Ages through their socio-economic organization, and their relationship to knowledge. Organized in guilds, they exhibited strong cohesion: between a master and their apprentices, and between equal practicioners[7, 8].

While crafts included glossaries to describe the details of their trade[9], the standards for quality were less explicit. Cennino Cennini, in his *Libro dell'arte*, one of the first codexes to map out craft knowledge in the early Renaissance, lays out practical advice on specific painting techniques, but does not explicitly mention how to make something *good*[10] and further attempts, at the eve of the Industrial Revolution, continued this formalization[11], resulting into being integrated with modern industrial processes[12, 13].

2.2 Programmers as craftsmen

Early computer enthusiasts described as hackers developed organizational features similar to their historical counterparts: spread in different geographic locations (Stanford, MIT, Bell Labs)[14], emphasis was put on engagement with tools[15], inquiring into peers' work[16] and later formalized into bottom-up archives[1]. Little accountability was required when it came to design explicitness: both the UNIX operating system and the TCP/IP protocol were originally realized without overarching supervision or extensive documentation[17, 14].

As computer science solidified as a distinct field in the 1960s, there 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 free-form programming came the structured programming approach[18]. With the operating system and the personal computer revolutions, access to tools became widespread, and transformed tightl communities into a global network of exchange. Inquiries into the relationship of craftsmanship with programming started to take place in the mid-1970s from educational[19], organizational[20] and inter-personal perspectives[21], and culminated with the publication of several trade books[22, 23], 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[24, 25], rather than passive execution[3].

2.3 The case of architecture

A counterpart to the computer scientist is the classical architect, but in an inverse relation: the architect emerged from centuries of hands-on work[26], while the computer scientist was first to a field of practicioners, followed by a need to regulate and structure those practices. Different sequences of events, perhaps, yet mirroring each other. On one side, con-

[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/`

[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`

struction work without an explicit architect, under the supervision of bishops and clerks, did indeed lead to significant results (e.g. 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 was described in *The Cathedral and the Bazaar*. Discussing the Linux project and its open-source philosophy, it showed how the quantity of self-motivated workers without rigid structures can result in better work than if made by a few, highly-skilled individuals[14].

In both cases, cooperation on a long-term basis out of intrinsic motivation, and without clear, individual ownership of the result, is possible; a parallel echoed in the similar concepts of *collective craftsmanship* in the Middle-Ages and the *egoless programming* of today[20].

3 Acquiring knowledge

3.1 Bus factor and tacit knowledge

The problem of knowledge in programming 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 project. Given the *essential complexity*[5] of programming, along with the compulsive behaviours sometimes exhibited by intrinsically motivated programmers[27], the gap between design and implementation—the domain of the craftsman—often relies on tacit knowledge[28].

Explicit knowledge, in programming as in most disciplines, is carried through books, academic programs and, more recently, web-based content, but both seem insufficient to reach an expert level[29]. Without tacit knowledge, the road to good code is quite unclear.

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

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

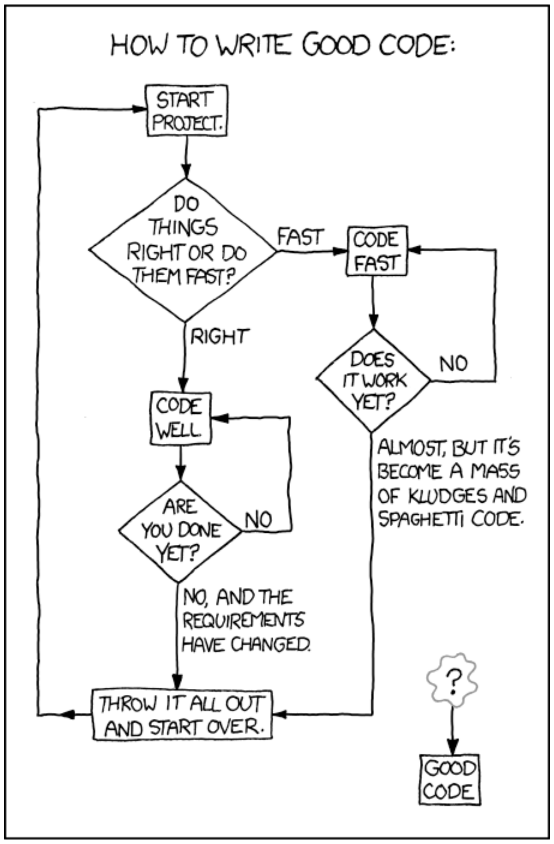
HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

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

3.2 Apprentices and masters

The acquisition of such implicit knowledge in craftsmanship takes place through the apprentice-master relationship, and the act of copying. In the former, a learner starts by imitating the way of working of the master, where important aspects of the craft are being demonstrated to the apprentice by a more experienced practicioner, rather than formalized and learned independently. Sometimes, this relationship to a master is explicit via practices such as pair programming[30] or corporate mentorship (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[31]. 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 takes the program writer as their master to achieve each of the assigned tasks. The role historically given to the master craftsman is delegated into the code itself, containing both the problem and the solution, straddling the line between formal exercises and interactive practice.

Code's ability to be copied complicates the argument of software as craftsmanship in terms of knowledge transmission. 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 on the technical affordance of code to be seamlessly duplicated[32]. This lets anyone inspect and use code, but even though the ability to perfectly copy anyone else's work became widely available to programmers, the difference between amateur and expert programmers lie in the extent to which they indeed blindly copy, or draw inspiration to write their own[33][7].

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

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

Eastern craftsmanship further qualify these different approaches to copying. *Moxie*, a Chinese term for copying and practice, is a key concept in how an apprentice can equal their master through thougtful replication [34, 35]—manually copying from quality work to seize their elusive essence is an essential aspect to craftsmanship.

3.3 The problem with copying

So while programmers are to acquire implicit knowledge through a process of learning by doing, we 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 seeing examples 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*[36],

> *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, the programmer faces a problem: a lot of examples, but a few good ones[37]: copyright stands in the way of pedagogical copying. With software becoming protectable under copyright laws in the 1980s[38], great works of programming craft became unacessible to programmers, despite their knowledge-value[39]. One of the most famous examples is *Lions' Commentary on UNIX 6th Edition, with Source Code*, which was circulated illegaly in classrooms for twenty years before its official publication was authorized by the copyright owners[40].

4 Material aesthetics

Complementing this process of knowledge acquisition is the *practice* of programming, and inherent in the practice is the *good practice*, the one leading to a beautiful result.

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 something beautiful[41], along with self-assigned standards of quality[42, 5]. These standards result in a craftsperson's *style*, and are gained through practice and experience, rather than by explicit measurements[42] [8]. A craftsperson should have a deep, implicit knowledge of tools and materials, 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.[41]*

He opens 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 "soulless society"[41], and has triggered debates about authorship, creativity and humanity at the cross-roads between artificial intelligence and artistic

[8]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."*

practice[43]. One avenue out of this debate is human-machine cooperation, first envisioned by Licklider[44, 45], which suggests programming as a distinctly 21st-century craftsmanship, as well as other forms of crafts-based work in our information economy.

4.2 Code as material

Beautiful code is an integral part of software craftsmanship[46]. More than just function for itself, code among programmers can, and should be held to beauty standards[47], rooted in traditional craftsmanship—form following function.

A craftsman's material consciousness is recognized by the anthropomorphic qualities ascribed by them to the material[5]. For code, adjectives such as ”clean”, ”elegant”, ”smelly” occur over and over in online discussions. They are indicators not just of the awareness of code as a raw material that should be worked with, but also of the necessities for code to exist in a social world. As programmers assemble in loose hierarchies to construct software, an aesthetic standard is *the respect of others*[48].

Another 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[49]. 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[50], code is perhaps the only non-finite material that craftspeople can work with—along with words.

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, creating 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.[20]*

4.3 The implications of beautiful code

So while there are arguments for having a more rigorous, engineering conception of software development[51], a crafts ethos based on a materiality of code has 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[52] or education[53], programmers play a significant role in the development of these systems[54], 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.[42]*

If most commentators on the history of craftsmanship, following Ruskin, lament the disappearance of a better, long-gone way of doing things, 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.

References

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

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

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

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

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

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

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

[8] Francis Wolek. The managerial principles behind guild craftsmanship. *Journal of Management History (Archive)*, 5:401–413, November 1999.

[9] 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.

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

[11] 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.

[12] 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].

[13] 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].

[14] 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.

[15] Neal Stephenson. *In the Beginning...was the Command Line*. William Morrow Paperbacks, New York, first paperback edition edition, November 1999.

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

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

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

[19] 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.

[20] 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.

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

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

[23] Mike Hendrickson and Pete McBreen. *Software Craftsmanship: The New Imperative*. Addison-Wesley Professional, 2002. Google-Books-ID: C9vvHV1IIawC.

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

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

[26] 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].

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

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

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

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

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

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

[33] 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.

[34] 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.

[35] 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.

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

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

[38] Ralph Oman. Computer Software as Copyrightable Subject Matter: Oracle V. Google, Legislative Intent, and the Scope of Rights in Digital Works. *Harvard Journal of Law and Technology*, 31(Special Issue Spring 2018):639–652, 2018.

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

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

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

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] Erik Pineiro. *The aesthetics of code : on excellence in instrumental action*. PhD thesis, KTH, Superseded Departments, Industrial Economics and Management., 2003. Publisher: Industriell ekonomi och organisation.

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

[49] 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.

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

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

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

[53] 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.

[54] 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.