A Game of Persistence, Self-doubt, and Curiosity: Surveying Code Literacy in Digital Humanities

Elli Bleeker[1], Marijn Koolen[1], Kaspar Beelen[3], Liliana Melgar[4], Joris van Zundert[1], and Sally Chambers[2]

[1]Huygens Institute for Dutch History and Culture
[2]Ghent University
[3]Alan Turing Institute
[4]Utrecht University

The interpretation of "code" and its role in the digital humanities has been a topic of debate ever since the first experiments in "humanities computing" in the late 1950s. Even some years ago, the question whether humanities students need to know how to code was sincerely provocative. At present digital humanists seem to agree that knowing how to code is relevant, if not essential for digital humanities research. However, there is a lack of agreement on what the community means by "code literacy", and as a result the efforts to improve code literacy among students and researchers are dispersed. This paper presents the, to our knowledge, largest ever survey on code literacy and related questions in the field. We expound the results and analysis of the first two overarching research questions of the four that the survey aimed to tackle: 1) What are the definitions and interpretations of code literacy across humanities disciplines?; 2) How important is code literacy as part of digital humanities scholarship? We explain what according to the survey answers are the most important elements or dimensions of a definition of "code literacy". Among these are the different levels of competence and the varying importance of code literacy across disciplines. The paper concludes by discussing the implications of these findings, opening up questions for the debate on academic curricula regarding digital humanities.

Keywords: code literacy; pedagogy; survey; computational thinking; coding skills; humanities computing/programming; concept analysis

1 Introduction

"Should humanists learn to code?" Less than a decade ago this question would have ignited quite a controversy in the field of digital humanities (DH). Today, the consensus

is that a certain level of code literacy is preferred.[1] Instead of arguing whether code literacy deserves to be part of DH's skill set, the debate has moved on to discussing what it means, exactly, to be code literate. How we define code literacy appears a Rorschach test: our definition depends on "our values, our experiences, our skills, our visions of the future as it pertains to technology, to computers, to communications, to information" (Beverly Hunter, cited in Dobberstein 1993, 430).

The somewhat confused state of definitions and discourse is not much helped by a proliferation of adjacent, overlapping, and vaguely defined literacies such as "computer literacy" (Dobberstein 1993, Tafazoli et al. 2017), "digital literacy" (Spante et al., 2018), "media literacy" (Potter, 2010), and so on. After mapping the use of the term "computer literacy" between 1965 and 1985, Kenneth King concluded half-jokingly, that this type of literacy is "widely regarded as an impossible term to define; however, whatever it is, it is important for every student to have it" (King, 1985, 20). Even those opposing the promotion of computer literacy, such as Douglas Noble – "What has apparently convinced an entire population that something as vague and worthless as computer literacy is essential to their lives?" – acknowledge that the community has "unusual difficulty arriving at a suitable definition."(Noble, 1984, 602,607) It does not help that a definition is inevitably influenced by the technological developments of the time.

Over the years, various researchers engaged with the topic of code literacy, presenting valuable insights. Their observations remained, however, limited to their own experiences with teaching digital humanities courses or with working in a digital humanities context (e.g., Van Zundert, Antonijević, and Andrews 2020). During a recent round table at the DH Benelux 2019 conference in which we explored the perception of code literacy in the field, the participants expressed smart opinions without providing much empirical evidence (Melgar et al., 2019). The animated discussion was largely informed by anecdotal evidence, repeating phrases such as "in my experience, ...", "in my department, ..."). To establish an informed and evidence-based definition of code literacy in the humanities we decided to ask the community for their opinion by means of a survey.

In this paper, we report on the results of the questionnaire answers focusing, first, on the present-day definitions and interpretations of code literacy across humanities disciplines and, secondly, on the importance of code literacy as a part of DH scholarship. The survey used a questionnaire that was widely distributed and received a large number of responses from a demographically diverse audience. As a result, the results allow us to analyse how factors like background, education, and experience shape opinions about code literacy. Instead of providing a clear-cut definition of "code literacy" – which would suggest a unity of perspective that in reality does not exist – this paper highlights a number of aspects that are considered important across the DH community. Accordingly, the contribution of this paper lies in a community-inferred vocabulary and a practical framework for the discussion, evaluation, teaching, and promotion of code literacy in digital humanities.

[1] Callaway et al. 2020 offer an overview of the topics discussed within the DH community over the past ten years; specifically paragraphs 3, 6, and 13 on the topic of "code". A Google Scholar query for "code", "coding", and "literacy" shows that the discussion is ongoing and anything but limited to the humanities domain.

2 Background

The interpretation of "code" and its role in the digital humanities has been a topic of discussion ever since the first experiments in "humanities computing" in the late 1950s. Few studies have, however, reviewed the history of this notion in a systematic way. This is surprising given that each generation of scholars was required to learn a different set of computer skills, so that perceptions of code (literacy) changed over the years.[2] Since the discussion on code literacy (in the humanities) is distributed over a wide range of sources, reconstructing its history proved to be a difficult task. Academic publications contain various traces and clues. More recent debates tend to take place online, on platforms like Twitter, or blog posts and DH forums.[3] In order to contextualise the findings of our survey, this section synthesises the historical discourse on code literacy on the basis of four shared themes:

1. The perceived role and potentially divisional nature of programming;
2. The importance of context for (teaching) code literacy.
3. The influence of code literacy on humanist thinking;
4. A distinction between multiple levels of literacy;

The literature used in this section is selected following a combination of purposive (or judgement) sampling and snowballing (or citation tracking) (Wildemuth, 2009). Starting from several seminal works on the topic of DH and code literacy, such as Gold (2012a) and Vanhoutte, Nyhan, and Terras (2013), we pursued relevant citations in their bibliographies. And finally, we carried out several search queries via Google Scholar and the JSTOR archive with various combinations of the keywords "code", "coding", "computer", "programming", "digital humanities", "humanities", "electronic", "literacy", and "literate". Although we will not claim that our literature review is exhaustive, we trust that it is representative. A final note on terminology: historical sources often refer to "computer literacy" rather than "code literacy'. For consistency we will use the term "code literacy" throughout.

2.1 In or Out: the Programming Divide

A major theme in the discourse on code literacy is the question of programming. The Oxford English Dictionary (OED) defines "literacy" – rather narrowly – as "the ability to read and write". If applied to code literacy, then, being code literate can be taken to mean "the ability to read and write code". Incidentally, "code" is already an ambiguous term in the humanities, because it can refer to text encoding with XML

[2] For example, being code literate had a different meaning in the 1960s, when few scholars owned a (in the parlance of the time) microcomputer, compared to the early 1980s, when the development of personal computers with a graphical user interface (GUI) took flight, and in the 1990s during the rise of the world wide web. Interestingly, educators have recently reported that students in STEM disciplines are no longer familiar with the directory structure, saving thousands of files on their desktop instead. This development could be explained by the omnipresence of the search function, which no longer requires users to know where their files are located (Chin, 2021). Although we will touch upon it briefly, the educational aspects of code literacy are not within the scope of the present article.

[3] As Gold notes, "some conversations, especially those on Twitter – a platform used extensively by digital humanists – are hopelessly dispersed and sometimes even impossible to reconstitute only a few months after they have taken place" (Gold, 2012b).

or HTML as well as using programming languages to write computer programs.[4] The question whether programming skills are a prerequisite for digital humanists has divided the field from the 50s until today (see Dobberstein 1993; van Zundert and Andrews 2017, ii86; Ide 1987).[5]

Especially when postulated as essential, proficiency in programming languages risks becoming a harmful criterion for exclusion. In the words of Douglas Rushkoff, "program or be programmed" Rushkoff. Earhart et al., for instance, notes an "increasing tension" between DH practitioners and DH theorists. As she sees it, both sides look down on the other: code literate scholars spurn those who are not (Ramsay, 2012), while simultaneously "builders" are not considered real digital humanists because "they haven't theorized their work within the context of humanities and technology" (Earhart et al., 2016). Annette Vee, too, points to the intricate tensions that hide behind the deceptively obvious compound formed by code and literacy. Coming from the sociological oriented New Literacy approach, she compares the historical development of textual literacy to the ideological push for "mass programming movements" (Vee, 2017, 152) and thus explains how a particular set of skills or capabilities can become a much coveted "literacy". Depending on one's perspective such a literacy may present itself as an individually empowering advantage or rather as a power game that excludes specific groups of people.

In 2015, O'Sullivan, Jakacki, and Galvin conducted a survey to find out whether programming is indeed perceived as a barrier to join the DH community. The trigger for the survey, according to O'Sullivan et al., was the essay "Who's in and Who's Out?" (Ramsay, 2013b), in which Ramsay famously declared that digital humanists should learn to code if they were to be part of the field. Although this essay caused quite a backlash and Ramsay later nuanced his statement, the sentiment appears to be persistent. The survey, which was completed by 96 respondents in the field, set out to study the "relevant attitudes in relation to [software] development within Digital Humanities projects" (O'Sullivan et al., 2015, 142). Notably—and against the authors' expectations—it was found that young scholars adhere less importance to programming than older generations: 40% of the respondents in the 25-35 age group considered programming as essential to DH work, in contrast to 60% in the category of 50 years or older. Senior scholars usually claimed to do the programming themselves, whereas younger scholars either collaborated with skilled developers or delegated coding to colleagues. Younger respondents also considered themselves less "technically proficient" (O'Sullivan et al., 2015, 144).

More recently, Callaway, Turner, Stone, and Halstrom (2020) examined "the gate keeping around coding and technical skills more broadly" by topic-modelling a corpus of 334 (English) definitions of DH found in readers and companions on the topic. In a way, their approach can be considered an indirect survey of the field. The authors found "a range of different views towards coding" (Callaway et al., 2020, §13). Few contributions took up a "hard stance" (e.g., humanists *should* learn how to code); most discussed whether programming skills formed a threshold for participation in DH and

[4] In a survey among 96 digital humanists, O'Sullivan, Jakacki, and Galvin found considerable disagreement on the definition of programming: "many respondents mention text encoding, particularly XML and HTML, as opposed to more sophisticated dynamic programming languages" (O'Sullivan et al., 2015, 145).

[5] It is worth pointing out that the term code literacy (or any of its related terms) rarely if ever appears in the computer science literature from this period. The explanation for this could be quite simply that "computer scientists have no more reason to refer to each other as 'computer literate' than say, physicists have in referring to their peers as 'science literate' " (Dobberstein, 1993, 430).

if so, how to overcome related hurdles. Interestingly, the authors noted that—being early-career academics themselves—interpreting the topics proved more challenging (and rewarding) than understanding the technical aspects (of topic modelling). This led them to conclude that "less emphasis should be placed on digital competencies and more emphasis on the step of interpretation that comes after the use of the digital tool" (Callaway et al., 2020, 28).

2.2 The Computer in Context

Indeed, most curricula concentrating on humanities and computing have taken code literacy to be more a broad than a narrow concept which usually comprises more than just programming skills. Early on, Zemsky contended that "we [historians] must ultimately invent a methodology—including computer programs—of our own, a methodology designed to cope with the peculiar kinds of evidence with which we deal" (Zemsky, 1969, 39). During a three-day workshop *Teaching Computers and the Humanities Course* (see also Hockey 1986 and Tannenbaum 1987), all participants—educators in humanities computing—agreed that when teaching computer skills "the example data must be from humanities disciplines" (Hockey, 1986, 228). And, Ide adds, they generally agreed that the computational methodologies need to be contextualised by traditional methodologies from the relevant humanities disciplines (Ide, 1987, 212).

In "Technology and the Historian" Crymble discusses the changing role of technology in the history classroom. Taking a longitudinal perspective—broadly ranging from 1980s to the 2010s—he describes the different ways digital technologies have been embedded in the historical curriculum: the initial emphasis on coding and counting (i.e. statistical analysis) in 1980s, was replaced with a focus on using digital, web-based technologies for public history, making historical narratives and sources accessible to a wider audience. Together with the rise of digital humanities in the 2000s, historians increasingly perceived digital methods as "tools". Historians were less concerned with teaching students how to code than to apply digital tools to historical data. The interest for digital tools was often applied and instrumental, ignoring the mathematics that underpinned many of methods and technologies used in digital history. Moreover, Crymble (2021) argues that "There was no overarching plan for a digital transformation of the historical classroom. Instead, it was a reactive intellectual space led by a few passionate individuals and largely ignored by the rest of the profession. These passionate few were not pushing a coherent agenda."

Reflecting on fifteen years of teaching computing to humanities students, Oakman found that mismatching "technique and subject" can result in outright "boring" courses for humanities students (Oakman, 1987, 232). To prevent this, he argues, humanists need to learn coding in combination with subjects of their interest. Furthermore, Oakman states that being code literate should include a knowledge of "the positive and negative effects of the Computer Revolution on the modern world (computers and society issues: privacy and government datafiles, automation and unemployment, robotics, technophobia, etc.)" (Oakman, 1987, 229). This definition foreshadows the current increase in interest for the sociological, political, and cultural effects of code on society, such as the emergence of the field of critical code studies (Marino, 2006), or the most recent volume of *Debates in the Digital Humanities* (Gold and Klein, 2019).

More recent sources also emphasise the importance of situating code literacy in its historical and social context (Vee, 2013, 43), and "to provide practical technical skills within a humanistic framework" (Clement, 2012, 24). Andrew Piper, for instance, in a recent thread on Twitter, states that to be successful as a DH scholar one needs to be able

to design and employ computational methods, but one also needs to be able to frame these in a theoretical, disciplinary context.[6] From a methodological perspective, and also relating computational work to quantified approaches, Piotrowski and Fafinski (2020) draw similar conclusions. And Ramsay, too, finds that programming offers a "methodology by which the artefacts of the human record are understood, critiqued, problematized and placed into dialogue with one another" (Ramsay, 2012, §6).

2.3 New Ways of Thinking

Acquiring practical computer skills is often associated with learning new or different ways of thinking. This pertains to both reasoning–more formal, more explicit–and to attitudes towards the role of code in DH research. In 1972, Oakman claimed that "learning to program computers [is] excellent training in rigourous thinking and logical reasoning" (quoted in Oakman 1987, 227). And, when evaluating the graduate course *Computing for Humanities* taught at the University of Aberdeen in 1987, Holland and Burgess concluded that the official course objectives were "relatively modest" – learning the terminology, key concepts, components, and customs of computing, including some relevant packages and "very simple programming" – while unofficially, they hoped to bring about a change in the mind set of their students:

> we wanted to encourage a confident, independent, exploratory attitude amongst students which we hoped would eventually give them the confidence and assurance to work out for themselves how to use new machines, new software and how to apply them to new humanities problems. (Holland and Burgess, 1992, 268-269)

Similarly, Ramsay found that learning basic programming skills alters one's way of thinking: "what is gained when humanities students learn to think in the context of sophisticated computational tools is not only computational thinking, but also 'humanistic thinking'", because they learn to engage differently with traditional objects of humanistic study (Ramsay, 2012). Nick Montfort, too, sees programming "as a way of inquiring about and constructively thinking about important issues" (Montfort, 2015, 98) and argues that learning computational methods can augment, diversify, and improve humanistic methods. As iterated above, the sociological study of code literacy and programming present similar findings (Vee 2017, 105; Burgess and Hamming 2011).

2.4 Levels of Literacy

Over the years, several studies have proposed a taxonomy of code literacy in an attempt to analyse the concept in a systematic manner. The general impetus for creating a taxonomy of code literacy is the acknowledgement that a single formal definition of code literacy is too restrictive since individuals may have "differing requirements" (Webster and Webster, 1985, 2). In his analysis of the code literacy in the 1980s, Dobberstein discusses different classifications, each identifying a number of levels, domains, or components of expertise (Blau 1985, Webster and Webster 1985, Konar, Kraut, and Wong 1986 among others). Notably, all taxonomies discussed by Dobberstein are hierarchical and define code literacy along an ascending scale of expertise. A disadvantage of such a scale is that it places "all users on the same continuum of expertise" (Dobberstein, 1993, 431). Consequently, the taxonomy still

[6] See the discussion on Twitter:
`https://twitter.com/_akpiper/status/1430265428711034881`, August 24, 2021.

presents a rather narrow concept of code literacy. In line with the studies discussed in 2.2, Dobberstein argues for a "context-sensitive" approach to code literacy, which accords with domain-specific requirements and skills.

2.5 Summarising the Status Quo on Code Literacy

The discourse on code literacy in the humanities can be characterised by several recurring themes. This is not to say that scholars agree on the themes: overall, we noted a general disagreement—and even confusion—in the literature about the (methodological) role and status of code and code literacy in the humanities. Another topic of discussion is the origins, motivations, and ramifications of the use of "code literacy" as a concept in the humanities. Several contributions mentioned the positive effect of (acquiring) computational skills on humanist research methods: scholars learn to be more formal and more explicit in their argumentation, and to have a confident, exploratory attitude toward computational methods. At the same time, scholars have emphasised the value of a humanist perspective on the sociological effects of code.

There is no lack of opinions from (digital) humanist scholars and educators, but there is little actual research from humanists into the functioning of code and its (methodological) effects in the humanities domain or in our society as a whole. Without a vocabulary or a clearly defined framework, the debate risks getting stuck in confusion and misunderstanding, making it hard to find consensus on the topic. In the past, a number of studies have proposed a taxonomy of code literacy that distinguishes various levels, components, types of knowledge and skills. However, the abstract, one-size-fits-all taxonomies have proven problematic, as they fail to capture the disciplinary diversity in the humanities. A context-sensitive definition of code literacy, one which is firmly embedded into a humanities discipline, is suggested as more appropriate. The contextual aspect of code literacy is emphasised by several scholars: in this definition, being code literate would include understanding how computational technologies can be applied to humanist subjects and methodologies. Opinions differ on whether "applying" means using of existing tools, developing new software, or conceptualising of computational methods.

Indeed, a large part of the discussion tends to focus on whether being code literate also implies being able to read and write code. Having (a lack of) programming skills is often felt to create a gap between DH scholars and practitioners. Some DH scholars consider programming an essential component of DH research, while others prefer theorising over practice. Both sides tend to perceive the other as exclusionary. In fact, a simple dichotomy between "programming skills" and "thinking skills" does not seem tenable in the case of code literacy in the digital humanities. As Burgess and Hamming (2011) and Vee (2017, 105) argue, the distinction between programming as mere material labour and academic scrutiny as pure intellectual endeavour is a gross underestimation of the particular type of interpretation and understanding of information that code literacy affords, and, which cannot be executed without that particular literacy.

3 Methods

The studies discussed in the previous section reveal the complexity of the topic of code literacy: there is no lack of valuable definitions and approaches, but all of these are based on theoretical considerations only or on limited empirical observation.

Furthermore, what counted as code literacy in the 1970s is often outdated in the 2020s due to technological advances. We therefore decided to question individuals currently working in DH by means of a survey in order to create a more stable dataset on the topic of code literacy in digital humanities.

As an instrument for collecting this data we decided upon an online questionnaire, to be distributed widely to anyone working in DH, including those working in the fields of GLAM (galleries, libraries, archives and museums) and LIS (library and information science), regardless of job type or background. Designing the questionnaire took seven months, from March until October 2020, and about 38 hours of discussion spread over 27 Skype calls. We were fortunate to have a diverse team that includes members with a background in information science and the social sciences, and took care to consult several experts in survey design.[7]

3.1 Survey design

By mapping the different forms of code literacy across DH disciplines, identifying problem areas and learning about existing approaches, we aimed to understand better the challenges involved with furthering code literacy in DH. We designed the survey around four research questions:

1. What are the definitions and interpretations of code literacy across humanities disciplines?
2. How important is code literacy as part of digital humanities scholarship?
3. How can we effectively approach the teaching and training of code literacy?
4. How can scholars (be supported to) incorporate code literacy in their research practice and methods?

After providing personal information (background, age group, career stage, etc.), participants were asked four sets of questions, each set designed to address one of the research questions. We followed a mixed-method approach, using both qualitative (open-ended) and quantitative (multiple-choice) questions, which is appropriate for assessing complex topics (Timans et al., 2019). For example, participants had to indicate whether they were satisfied with their own level of code literacy on a Likert scale from 1 (dissatisfied) to 5 (highly satisfied).[8] The respondents who indicated a number lower than 3 were asked to elaborate upon the reason why, in an open text field. All respondents were then asked whether they—using a yes/no question—would like to expand their level of code literacy. If they answered "no", then they were asked in a multiple choice question to give at least one reason why (options ranging from "I don't know where to start" to "I have no time".) This flexibility allows us to gain a more complete understanding of the respondent's situation.

As survey designers, we are all working in the field of DH and each of us has a different experience with learning to become code literate. Being vigilant of our own biases playing a role in the design of the survey, we opted for the "post-positivist" approach as the best way to design the survey (Ryan, 2006). A post-positivist approach

[7] The authors would like to express their gratitude to their peers, with a special mention of Merisa Martinez, PhD candidate at the School of Library and Information Science in Borås, Sweden, for her insights and advice into effective survey design and analysis.

[8] All Likert scale questions in the survey also included the option "Don't know".

recognises that the biases of the parties involved will have some effect on the targeted audience, on the survey's structure, and on the phrasing of the questions. For example, respondents may not share our perceived value of code literacy in research practices, so we should be vigilant about this and phrase questions as neutrally as possible. We also tried to be transparent about our intentions when we distributed the survey.

A post-positivist method requires us to regularly reflect on what exactly we wanted to know and how we could best articulate a question. For example, we needed to provide a broad working definition of code literacy which a large number of respondents would recognise as useful. At the same time, we realised that not all respondents would agree with how we defined it. We concluded that any definition we would provide was open for debate. Therefore, we asked respondents to provide their own definitions as well. They were subsequently asked to indicate whether they would be using our definition or their own when answering the remaining survey questions. We realised that this setup could complicate the analysis of the survey results, as for each question we would need to keep in mind that a respondent may have a diverging interpretation of code literacy. However, it also enabled us to gain insight into the various understandings of coding and code literacy in the DH community. We could explore the relationships between a respondent's definition and their answers to other survey questions, and thus step outside the framework of our own experiences.

As said, the intended audience of the survey was the wider DH community, including but not limited to scholars, teachers, students, librarians, and developers. We distributed the questionnaire via contacts at international research institutes and universities, various international (digital) Humanities email lists—e.g., the Humanist Discussion Group, the mailing list of the Text Encoding Initiative (TEI), DM-L (Digital Medievalist)—several DH slack channels (e.g. DHtech), and social media (Twitter). We identified the intention and objectives of the survey in an accompanying email as well as on the survey's home page.

With 399 responses the questionnaire has reached a large audience, but we are mindful of a self-selection effect among the respondents. That is, a digital humanist might be more inclined to answer a survey on code literacy if they consider themselves (somewhat) code literate. Another (potential) pitfall is that a qualitative survey is by nature a self-report measure. It aims to capture how a respondent feels about, and relates to, the object of study, but it is entirely depended on what the respondent chooses to share. Self-report measures have therefore been said to be subjective and "less robust" (O'Brien and McCay-Peet, 2017, 27). We prove the validity of our findings by triangulating the responses with, first, the findings from related studies (methodological triangulation) in section 2 and secondly, by distinguishing types of respondents based on background, career stage, etc.

3.2 Testing the Survey

We first collaboratively drafted the questionnaire using Google Docs. An extended period followed in which we commented and made suggestions on how the questions were articulated, on the order in which they appeared, and whether they sufficiently addressed the underlying research questions. We then structured the questionnaire using the open source software LimeSurvey[9] and tested it ourselves, discussing over the course of several meetings what did, and what did not, work. Once we agreed

[9] See `https://www.limesurvey.org/`. The software was running on a secure server from the Humanities Cluster of the Royal Netherlands Academy for the Arts and Sciences.

on the instrument as a team, we piloted the questionnaire by asking eight peers from a range of backgrounds (PhD students and postdocs in the Humanities, Cultural Heritage experts, and an Information Science professional) to test it and provide feedback. Specifically, we asked the testers to check the different "paths" through the questionnaire (depending on their answers to a certain question, a respondent would get a different follow-up question), to see if the questions were clearly phrased, and to inform us about the average time it took them to complete the questionnaire. Based on their feedback we adapted the instrument into its final form and distributed it over the aforementioned channels.

3.3 Preparing Response Data for Analysis

In the first round of analysis (on which we report in this article) we focus on the responses to questions related to RQ1 and RQ2 – the definition and relevance of code literacy – and the personal information provided by the respondents regarding their background, career level, etc. We analysed the results following an inductive method, as we had no hypothesis as point of departure, but a set of research questions.[10] The qualitative (open-ended, free text) questions were coded using open coding (Corbin and Strauss, 1990) that was triangulated by several members of our team in order to strengthen the validity of our coding. To give but one example: research question 7 (RQ7) asked all participants to define code literacy in a free text box. Three members of our team each coded the responses to this research question and subsequently categorised their coding. Examples of such code categories are "Conceptual aspects of code", "Aspects of literacy", or "practice". We then merged our codes into one Axial list, and recorded the responses to RQ7 using this axial coding. Next, the other members of our team used the Axial list to code the same data. In between, we held regular meetings to discuss the codes and consolidate our views on tricky categories. We further analysed the data using Jupyter notebooks. The results of this analysis are discussed in section 4.

3.4 Privacy

The landing page of the survey informed the respondents of the privacy aspects regarding the handling of their responses, which also included the names, affiliations and email addresses of the survey creators. We also informed respondents that their responses will not be shared in raw form beyond the creators of the survey. The questionnaire was anonymous, meaning that we do not store any identifying information about the responses, such as IP address. The only exception to this was the option for respondents to leave their email address to be contacted for a follow-up interview. The questionnaire included a number of questions on personal information, such as job type, academic background, gender, career stage, and country. We follow the data management guidelines of the KNAW[11] and the European Union[12] and store the data for a period of five years, to allow our analysis to be reproducible. The survey has been reviewed according to the GDPR and the Utrecht University guidelines for research

[10] Azungah 2018. See also https://conjointly.com/kb/deduction-and-induction/.

[11] https://www.knaw.nl/en/news/publications/big-data-in-wetenschappelijk-onderzoek-met-gegevens-over-personen/@@download/pdf_file/20180515-advies-Big-Data.pdf

[12] https://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/ethics/h2020_hi_ethics-data-protection_en.pdf.

involving human subjects.[13]

4 Results

In this section we discuss the responses related to the two research questions addressed in this article: the definition and relevance of code literacy.

4.1 Demographics of Respondents

The questionnaire started with a series of demographic questions to get an understanding of who completed the questionnaire and therefore also of potential over and under-representation of certain groups within the DH community.

We made the question about the respondent's gender an open text field to allow them to express their gender in the way they saw fit. As the vast majority filled either "female" or "male" , we coded the responses using four groups: female ($n = 166$ or 42%), male ($n = 208$, or 52%), other ($n = 7$, or 2%) and unspecified ($n = 25$ or 5%). The latter corresponds to respondents who left the field empty. From this we observe that female and male sexes seem to be equally represented in the sense that the respondents are not predominantly male or predominantly female. For gender, as for other demographic aspects, we cannot make any claims as to how representative this distribution is for the DH community, as it is hard to determine the boundaries of this community.

The majority of respondents work at universities and research institutes. We asked for the country of their current affiliation (in the case of multiple affiliations, we asked them to select one), using a controlled list of country names, see Figure 1. Just under half of all respondents are from Europe ($n = 190$ or 48%), with another large group ($n = 106$ or 27%) being from North America. Not all respondents filled in an affiliation and corresponding country ($n = 85$ or 21%). This puts some limitations on the representativeness of this survey, as Oceania, South America, and especially Africa and Asia are underrepresented (or not at all as in the case of Africa). Any claims we make should therefore be considered to apply more to Europe and North America than to the DH community more broadly.

We also asked respondents about their current role(s) in DH scholarship, and provided a list of roles (e.g. *academic*, *developer*, *librarian*, *administrative personnel*, *student*) as well as a free-text box. Respondents could select and/or enter any number of roles. The free form roles overlapped partially with the list of provided roles, so we coded them into five main groups: *researcher*, *student*, *IT specialist* (including developer, software engineer and technical support staff), *information specialist* (which includes archivists, librarians, curators, documentalists and data specialists) and *other* (including administrative personnel, editors, publishers and designers), see left side of Figure 2.

---

[13] For the GDPR, see: `https://gdpr-info.eu` For the Utrecht University guidelines, see: `https://www.uu.nl/en/research/research-data-management/guides/handling-personal-data#anonymise`

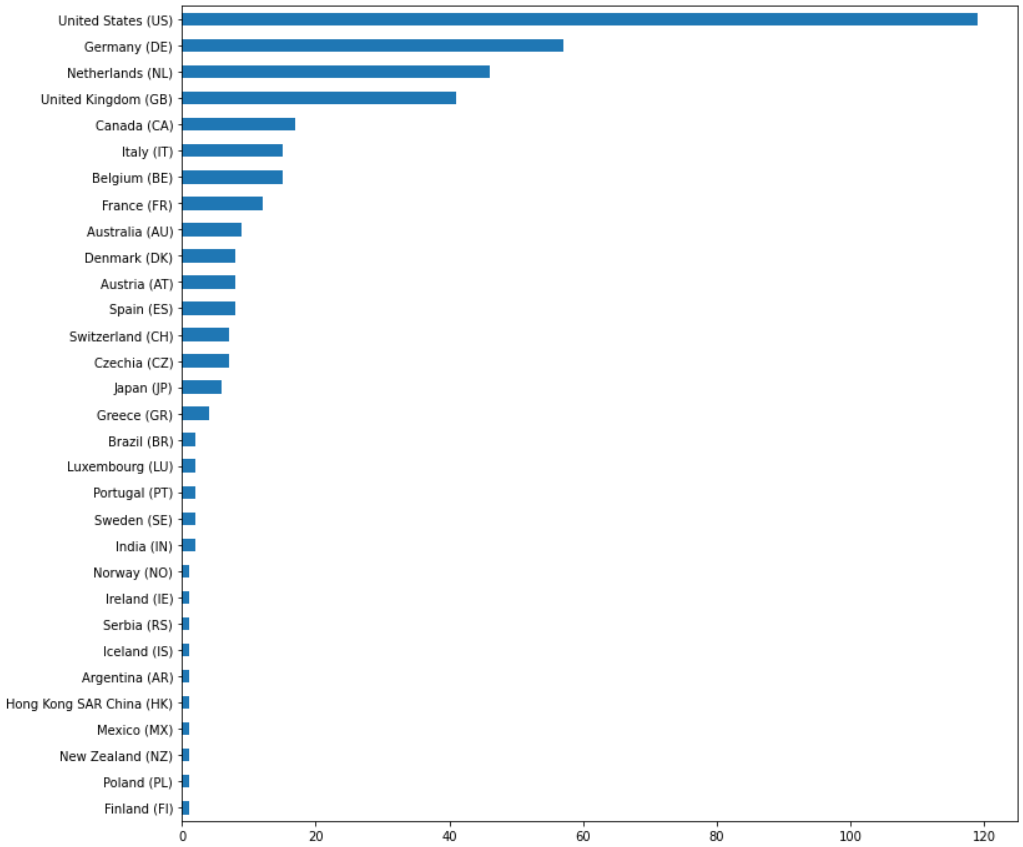

Figure 1: Distribution of respondents across countries.

The majority of respondents consider themselves *researchers* ($n = 301$ or 75%). The other two main groups, not surprisingly, are *IT specialists* ($n = 76$ or 19%) and *information specialists* ($n = 77$ or 19%). Most groups overlap substantially with the *researcher* group. For instance, 57% of *IT specialists* ($n = 44$) are also *researchers*. The other groups are more distinct, with only 19% of *information specialists* also having a role as *IT specialist* (vice versa is 20%). Altogether this suggests we managed to reach people with a diverse set of roles in the DH community.

Next, we asked respondents about the number of years of experience they have working with the field of Digital Humanities (see right side of Figure 2). Note that this does not necessarily correspond to their number of years of experience within their current roles, as some may have been a researcher or developer for decades but only started working in DH less than a year ago.

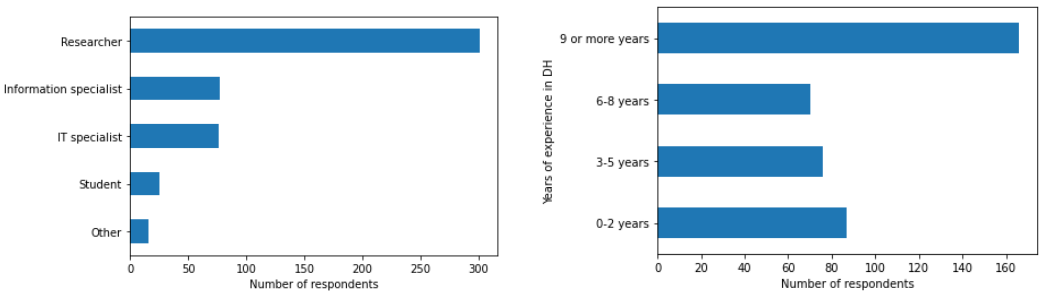

Figure 2: Distribution of respondents across roles and career phases in DH.

Almost half of all respondents has 9 or more years of experience in DH and less than a quarter, fewer than 2 years. It is difficult to establish how representative this is of the entire DH community. Our guess is that the more experienced contingent of the community is somewhat over-represented, as they are more likely to have come across our survey via the channels we used.

Respondents also have a wide range of academic backgrounds (see Figure 3), with large groups of respondents having a humanities background, specifically in *History*, *Language and Literary Studies*, or *Textual Scholarships*. (The latter includes Book History, Paleography, Scholarly Editing and Textual Criticism.) There are also many respondents with backgrounds in *Computer Science* and *Library and Information Science*.

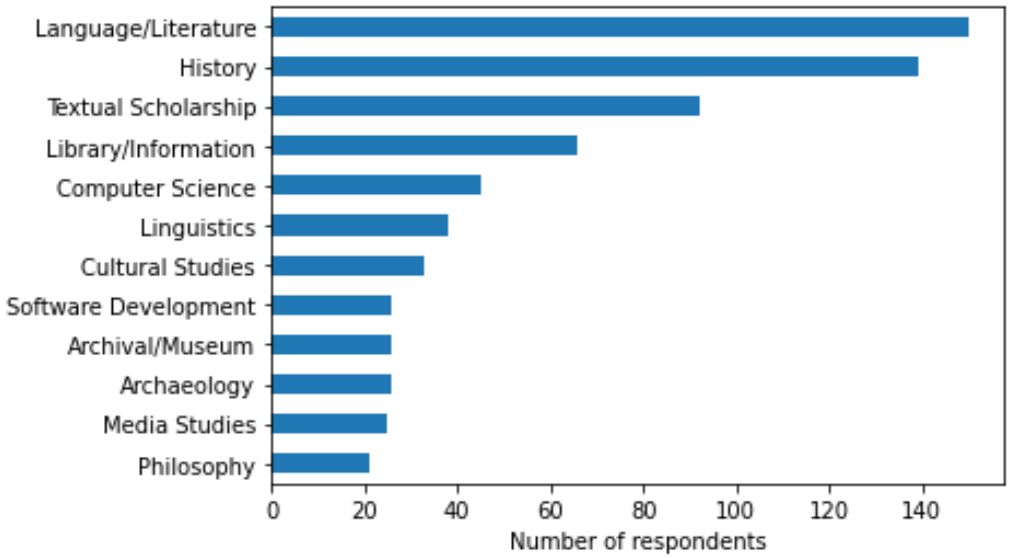

Figure 3: Academic backgrounds of respondents, where they could check multiple backgrounds from the ones we listed as well as add additional backgrounds in a free text field. The table lists only the backgrounds checked by at least 5% of the respondents.

4.1.1 Overlapping categories and significance testing

Before moving on to discussing how respondents define code literacy and answered further questions, we discuss ways in which we can meaningfully compare sub-groups of respondents along different demographic dimensions. To test the observed differences for statistical significance, we need to consider different tests for the variables that were coded with mutually exclusive categories - such as gender and career phase - and those that were coded with overlapping categories, like role or disciplinary background, where respondents could select multiple answers.

The former type of variables were tested using a one way ANOVA with Tukey Honest Significant Difference (HSD) post-hoc tests (Sachs, 2012, Ch. 7). For the latter type of variables, we consider two options. One is to make as many groups as there are combinations of answers. E.g. all respondents who indicated to have both a role as *researcher* and as *information specialist* are in a separate group from those who selected only *researcher* or only *information specialist*. In this way, no respondents belong to multiple groups, but we can still compare the groups for differences in how they define code literacy or answer other questions. There are two drawbacks to this approach. One is that it creates many groups, each with only a small number of respondents, resulting in many comparisons of pairs of groups, which are often too

small to establish statistical significance. The other issue is that even though this step establishes distinct groups, they can be hard to interpret. How should we interpret the difference between the group of respondents who are *researcher*, *information specialist* and *IT specialist* against the group who are *researcher* and *information specialist*, the group who are *researcher* and *IT specialist*, and those who only selected *IT specialist*, *information specialist* or *researcher*?

The other option we consider is to compare all respondents who selected a particular role or background, against who did not, using a χ^2 test. This has the disadvantage that we do not directly compare groups, but the advantage of more clearly interpretable test hypotheses and outcomes. E.g. are respondents with a role as *IT specialist* more or less likely to include *writing code* as part of the definition of code literacy than those who do not have this role? In the analysis below, we use the latter option. That is, we use the χ^2 test on the group with a particular attribute or response and the complement of that group.

4.2 Definitions and Interpretations

After the demographic questions, we asked respondents to provide their own definition of code literacy. As this was a required question, all 399 respondents provided a definition.

Below are a few examples of typical definitions given by the respondents:

> *"The ability to understand and write code and to use it to achieve some research goal."*
>
> *"The ability to read, write and use code."*
>
> *"knowledge and experience of solving problems through the use of programming skills"*

But there are also many definition that are more elaborate and rich:

> *"I'd say there's informal and formal code literacy. Formal literacy is what you find with colleagues in the sciences and engineering. There's a strong emphasis on style, standards, and efficiency. Tests and quality control are required. Then there are the programming historians, the self-taught, and highly pragmatic types. It's just amazing that it works at all. The code is an odd pastiche of cut and pastes from Stack Overflow. It's a game of persistence, self-doubt, and curiosity. I mostly work with informal code literacy to give people the practical skills of reading error messages and documentation, finding helpful solutions to problems, and making something that works but is by no means pretty or professional."*

To be able to analyse and compare these definitions, we coded them using open, axial, and selective coding methods (Corbin and Strauss, 1990). The open coding step was done individually by three of the authors. In the axial coding step, each of the annotators had already created a hierarchy for their own codes, so we compiled a list of all the hierarchical codes and derived a single hierarchy of codes. The overall hierarchy consists of five basic aspects, each with several sub-aspects. The basic aspects are *Communication*, *Code Type*, *Contextual Understanding*, *Level of Competence* and *Other*, see Table 1.[14] Apart from the five basic aspects, we added a code for responses that do not resemble a definition but, according to us, seem like a response to a different question. An example of this "I have none experience whatsoever, but I would like to change that." This is the case for 23 responses. One respondent only filled in a hyphen, possibly to be able to move on to the next question without providing any definition.

Further in the survey, we included a working definition of code literacy, and asked respondents if they wanted to answer the remaining questions using our definition or

[14] The full hierarchy of aspects, including the scope notes of each, is provided in Appendix A.

their own. We included this in case respondents were uncertain or not satisfied with their own definition. Our working definition was:

> Code literacy has to do with different levels of ability to recognise, interpret, and use code; not necessarily being able to create it yourself. In this context, code can refer to encoding (e.g. XML or MPEG) as well as program code (e.g. Python or R) to operationalise a process in concrete steps and actions.

Code	*Description*	*Scope note*
COM	Communication	Knowing how to communicate about code with others, either as a coder yourself, or with someone who codes for you, including communication about purpose, workings, role and implications, as well as teaching code literacy.
CT	Code Type	Whether the code explicitly refers to a specific type of code, either 1) code as encoding of documents or 2) performative code for e.g. processing of data.
CU	Contextual Understanding	Understanding that code sits in a context and how it relates to 1) research and operationalising research questions, 2) it's possibilities, limits and biases, 3) the culture and attitudes toward code, 4) the ecosystem around code of ethics, privacy, security, maintenance, documentation, versioning, licensing, practices, platforms and software, and 5) other aspects of context.
LOC	Level of Competence	Literacy is divided into 7 different levels of competence, from 1) understanding the basics of what code is and does, to being able to 2) read code, 3) write basic code, 4) modify existing code, 5) review code, 6) create and package code from scratch, and 7) understanding of the theory of computation and coding paradigms. Separate from those levels there is a sub-category for definitions that mention there are multiple levels of competence.
OTHER	Other aspects	Any code literacy aspects not covered by the above four aspects.
MIS	Misinterpretations	Responses that are not definitions but answers to a different question.

Table 1: The top level aspects of our axial coding of the respondent's definitions of code literacy.

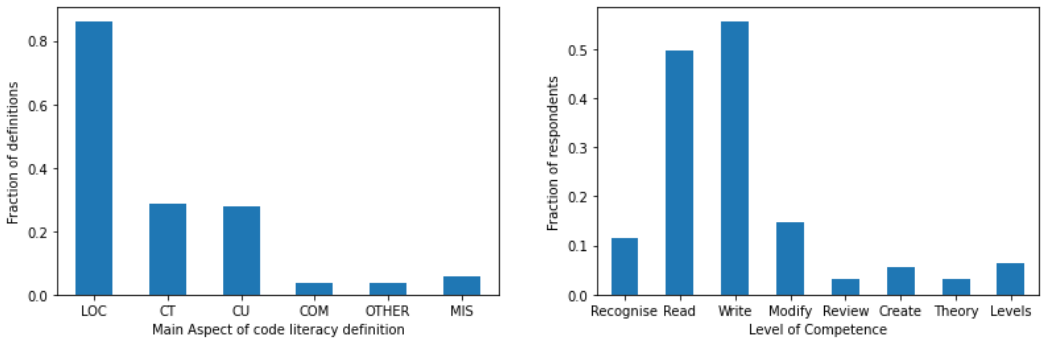

Figure 4: Fraction of definitions that include a main definition aspect (left) and that include sub-aspects of Level of Competence for all 399 respondents.

The distribution of aspects given in respondents' definitions is shown on the left side of Figure 4. The vast majority of respondents mention at least one of the levels of competence (*LOC*) as part of the definition of code literacy, such as the following two definitions:

> *"the ability to recognize, understand and write codes of any kind, e.g. markup languages, programming languages, annotations or any other kind of symbolic, rule-based/formalized means of communication"*
>
> *"the ability to read code to comprehend the purpose of functions, their syntax, and what output results; the ability to interact with code to improve its functionality, ether in terms of efficiency/clarity or in terms of research purpose; the ability to write (invididually or collaboratively) new code"*

Only around 28% mention code type (*CT*) or contextual understanding (*CU*). Communication (*COM*) is rarely mentioned (less than 4%). We note that our working definition did not include any aspects of contextual understanding (although it hints at operationalising research processes), but aspects of it were mentioned by at least 20% of respondents, in each of the 12 disciplines, with more than 20 respondents. There are significant differences between disciplines in how many respondents mention any aspects of *Contextual Understanding*.

Looking at the number of coded aspects in each definition, we found that most respondents mention one, two or three aspects (30%, 34% and 20% respectively), with the remaining 16% of definitions mentioning between four and seven aspects.

> *"being able to write and deploy code when you encounter problems that code could help you deal with; being able to read code of others; being able to read pseudo-code and series of formulae in publications; knowing how to use tools that support code development and implementation (version management tools, Pypi, etc.); being familiar with terminology used in discussing code and coding issues."*

We found that *IT specialists* mention more aspects in their definitions than non-IT specialists (2.8 versus 2.2 aspects, χ^2 with $p = 0.002$), but career phase has no significant impact.

4.2.1 Level of Competence

Level of Competence (LoC) is by far the most mentioned aspect in the code literacy definitions. Across gender categories, career phases, roles and disciplinary backgrounds, we find no significant differences in how often LoC is mentioned. This suggests that at least some sub-aspects of LoC are core elements of code literacy on which there is a broad consensus. Moreover, it reflects that discussions on the role of programming are still central to defining code literacy (Dobberstein, 1993) even though respondents differ when it comes to prioritising competences. Zooming in on the sub-aspects of *Level of Competence*, shown on the right side of Figure 4, reveals that the main aspects that respondents agree on are the ability to read ($N = 198$ or 50%) and write code ($N = 222$ or 56%). Digging further into the *read* and *write* aspects of definitions, we noticed that there is a substantial group ($n = 77$ or 19%) who mention *reading* code as a core competency of code literacy, but not *writing* code. This group of definitions is not associated with any specific disciplines. Many of these definitions emphasise that at a minimum, those involved in DH research should be able to read and get the gist of what a piece of code does, without necessarily being able to write such code. Examples include:

> *"Being able to read and understand, not necessarily write, code"*
>
> *"Understanding of how code works. Not necessarily how to code something specific yourself, but you*

are able to understand how code is build and how to read it".

On the other hand, 101 respondents (25%) mention *writing* code but not *reading* code, indicating that for another substantial group, the best or most logical approach to learning to code is by doing. Examples are:

"Ability to write/correct code that runs successfully, eventually"
"The ability to write practically usable code in some computer programming language"
"be able to write code to solve problems"

Overall, these results suggest that the DH community is still largely split on the question if code literacy implies possessing programming skills (Ide, 1987; Ramsay, 2013b; Rushkoff, 2010). While a majority supports the contention that literacy means writing of code, a still substantial group of respondents would disagree, foregrounding other ways of engaging with programming languages.

This prompted us to think about the different ways in which code literacy can be taught or how different types of learning materials can be offered. For some groups of students and scholars, or within some curricula, it might make sense to focus first and foremost on how code relates to aspects of their discipline, how researchers translate between research questions and following the logic of code, thereby emphasising how code fits in the research *process*, while for others, the best way might be to start with writing straightaway. For instance, people who already have some experience writing basic code, might improve their skills (e.g. with additional elements or a different language or paradigm) more efficiently through writing. Alternatively, the nature of research in some disciplines is more directly translatable to computational thinking and breaking down the process into procedures with explicit steps, making it more relevant to learn writing small bits of code from the start. While for other disciplines, the connection between their research process and code might be more complex, in which case, learning about the role and possibilities of code might require focusing on its relevance and potential, and engaging with examples of code that have been successfully applied in their discipline.

The other aspects in the *levels of competence* are mentioned by only few respondents, and represent more advanced perspectives on code literacy.

4.2.2 Contextual Understanding and Code Type

Next, we zoom in on the two other main aspects mentioned regularly in definitions, *Code Type* and *Contextual Understanding*. First, since only 28-29% of respondents mention these aspects, we want to know if these respondents differ from the others in terms of demographics, background, literacy or career phase.

Among the groups of roles we created, IT specialists more often also think about other aspects (code type and contextual understanding). We assume that IT specialists are most involved in working with code, and as noted above, they have more elaborate definitions than most others.

For example, this definition is one of the most "dense" in terms of elements (given by an academic researcher):

"I distinguish:
- a basic understanding of the principle of coding and the affordances and limitations for one's discipline, necessary to collaborate with a computer scientist or software developer
- the capacity of executing predefined rules in a learning environment
- mastering a computer language and knowing how to write code to execute particular actions

- knowing several computer languages that are relevant to a specific domain and the specificities of each one of them
- being all round with computer languages that cover the whole eco-system of digital humanities , so that you can create an infrastructure".

As well as this one provided by an IT specialist:

> *"The ability to conceive and implement a software solution to solve a specific problem in an adequate and efficient manner. The ability to understand and augment solutions implemented by others. The ability to use common infrastructure tools needed in software development (text editors or IDEs, version control)."*

Only 28% of respondents made explicit reference to type(s) of coding in their definitions (i.e., encoding or processing). When they did, some mentioned both, some mentioned only one. Textual Scholars are more likely to mention *encoding* than non-Textual Scholars (16% versus 6%, χ^2 with $p = 0.009$) and respondents who have a background in software Development are more likely to mention *processing* than respondents who do not (58% versus 24%, χ^2 with $p < 0.001$).

Within *Contextual Understanding*, the *possibilities, limitations* and *biases* are more likely mentioned by respondents with a background in *Archival & Museum Studies* (27% versus 8%, χ^2 with $p = 0.003$), *Library and Information Science* (17% versus 8%, χ^2 with $p = 0.003$) or *Linguistics* (21% versus 8%, χ^2 with $p = 0.002$).

Given the emphasis on contextual understanding in the (academic) literature (Clement, 2012; Dobberstein, 1993), the relatively low number of mentions of this aspect (mostly coming from IT specialists and *not* researchers) was somewhat of a surprise. It suggests that the community could benefit from expanding its notion of code literacy beyond discussing competencies, or programming languages, and instead perceive literacy as the understanding of how code interacts with/relates to the humanistic framework in which it operates(Clement, 2012).

4.2.3 Code Literacy Level of Respondents

As people with several years of coding experience may have different definitions than people who have never directly interacted with code before, we asked respondents to score themselves on a five-point scale as to how code literate they consider themselves to be, given their own definition of code literacy (see left side of Figure 5).

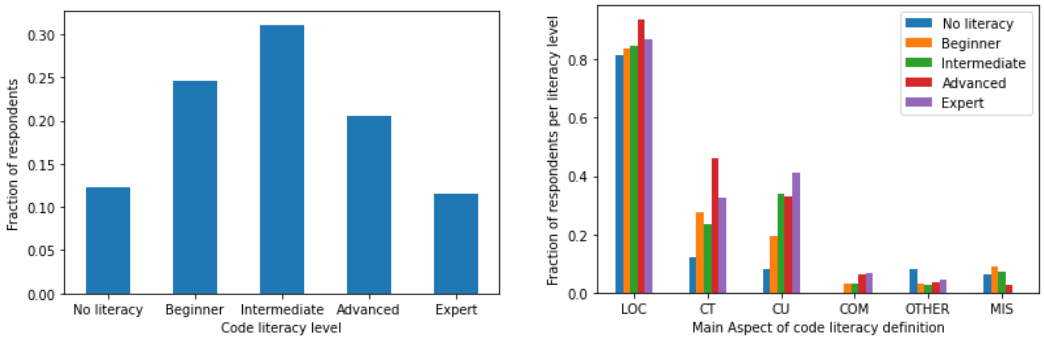

Figure 5: Distribution of self-reported code literacy levels among all 399 respondents (left), and of main aspects of definitions per code literacy leve.s

The five-point scale has the following labelled levels: *No literacy*, *Beginner*, *Intermediate*, *Advanced* and *Expert* (ConcordiaUniversity, 2011).

The distribution of the responses is close to a normal distribution, with the most frequent level being *Intermediate*, and frequencies tapering off towards the extremes. The *Experts* are the smallest group with $n = 46$ respondents.

There is an association between gender and literacy level. Women are more likely to consider themselves at *beginner* level than men (32% versus 18%, Tukey HSD with $p = 0.008$), while men are more likely to consider themselves at *advanced* (11% versus 29%, $p = 0.001$) and *expert* level (4% versus 17%, $p = 0.001$).

There is also a clear association between role and literacy level. *Information specialists* are less likely than non-information specialists to consider themselves *advanced* (10% versus 23%, $p = 0.02$), while, as is to be expected, IT-specialists less frequently have *no literacy* (0% versus 15%, χ^2 with $p = 0.001$) or be at *beginner* (11% versus 28%, $p = 0.003$) or *intermediate* (21% versus 33%, $p = 0.05$) level than non-IT specialists, but more likely to be at *advanced* (37% versus 17%, $p < 0.001$) or *expert* level (32% versus 7%), $p < 0.001$).

There are also significant relationships between disciplinary background and code literacy level, which we argue has implications for how code literacy is best taught within different disciplines. Respondents with a background in History, are more often than non-Historians at the level of *no literacy* (20% versus 8%, χ^2 with $p = 0.001$) or *beginner* (32% versus 21%, $p = 0.02$), and those with a background in Language and Literature are less likely to be at *expert* level than those without (7% versus 14%, $p = 0.03$), while those with a background in Linguistics are less likely to be at *beginner* level than those without (5% versus 27%, $p = 0.007$). These differences are visualised in Figure 6.

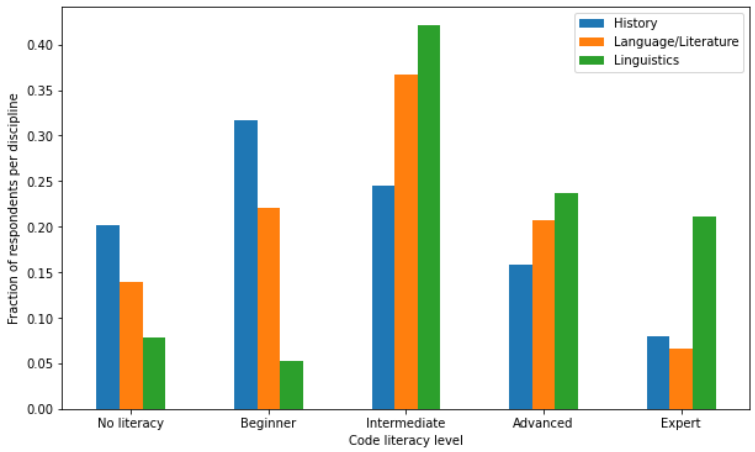

Figure 6: Code literacy level of respondents with a background in History, Language and Literature, and Linguistics.

One possible explanation for the differences between these humanities disciplines is the different research methods and data they use. Linguistics has a long history of computational approaches to analysing speech and textual utterances using quantitative methods (Hajic, 2004), so perhaps the step of translating methods and techniques to code is relatively small and requires no significant change in mode of thinking. Whereas for historians, the established methods of archival research and making associative connections through close reading of heterogeneous documents are perhaps less easily translated into quantifiable and computational steps. It may therefore be more beneficial for historians to see examples of how computational processes have been translated into a recognisable part of historical research. This can help both the

understanding of the potential and limitations of code for their discipline, as well as introduce "thinking in code" or "thinking through code" as a relevant mode of thinking.

Interestingly, although historians and language and literature scholars score themselves lower on code literacy than many others, they form the two largest disciplinary groups in this survey, each represented by over 140 respondents. This could potentially be related to representation and self-selection bias (see Section 4.1). Again, as expected, respondents with a background in Software Development or Computer Science are rarely at the level of *no literacy*, *beginner* or *intermediate*, and are more often at the level of *advanced* or *expert*.[15]

The level of code literacy of respondents is clearly associated with the definitions (right side of Figure 5). The majority of respondents at all levels agree that *Level of Competence* is part of the definition, but *Code Type* and *Contextual Understanding* are mentioned more frequently by respondents with higher levels of code literacy. Respondents at Advanced level are significantly more likely to mention *Code Type* (41%), specifically *Processing*, than those at No Literacy (8%, Tukey HSD with $p = 0.001$), Beginner (19%, $p = 0.003$) and Intermediate (34%, $p = 0.04$) levels. The same goes for *Contextual Understanding*, which respondents at No Literacy level are significantly less likely to mention (12%) than respondents at Intermediate (23%, Tukey HSD with $p = 0.005$), Advanced (46%, $p = 0.02$) and Expert (33%, $p = 0.003$) levels, and those at Beginner level are significantly less likely to mention it (28%, $p = 0.04$) than those at Expert level.

From this we speculate that more direct experience with code provides a richer vocabulary to talk about code, a wider perspective on how code relates to the research questions and processes, and that writing and interacting with code brings people into contact with the larger ecosystem of coding languages, interpreters, data formats, versioning and management of code, aspects of ethics and privacy and other elements.

This does not mean that *Contextual Understanding* is a more advanced concept that should be taught at a later stage in the curriculum. We argue that more experienced coders are more aware of the importance of the context in which code is created and used. This also ties in with the earlier observation that foregrounding Contextual Understanding would help establishing a more nuanced and broader perception of code literacy in the DH community.

To conclude our analysis of the definitions of code literacy, there are three important dimensions to consider when talking about code literacy: *Level of Competence* is the main dimension that respondents across all career phases, roles, disciplinary backgrounds and levels of code literacy mention, while *Code Type* and *Contextual Understanding* are more recognisable to those with some experience in using or creating code. The different perspectives of what code literacy is, also suggest that there is no one-size-fits-all approach to teaching it, but that it makes sense to differentiate between different disciplines and roles. We will address the question of how to teach and incorporate code literacy in curricula in a follow-up article.

4.3 Importance of Code Literacy

We turn now to the second research question. How important is code literacy as part of DH scholarship? After giving their own definition of code literacy, we asked respondents to consider how important they think code literacy is for digital humanities

[15] We leave out percentage and P-values for readability.

scholarship, provided them with five different levels of importance as well as the option *Don't know*. The five different levels are: *Not important at all*, *Somewhat important*, *Important*, *Very important* and *Crucial*. The distribution of responses is shown on the left in Figure 7. Only 5 respondents indicated that they did not know (1%), and only 21 (5%) answered *not important at all*, which means 373 respondents (93%) consider code literacy to be at least *somewhat important*.

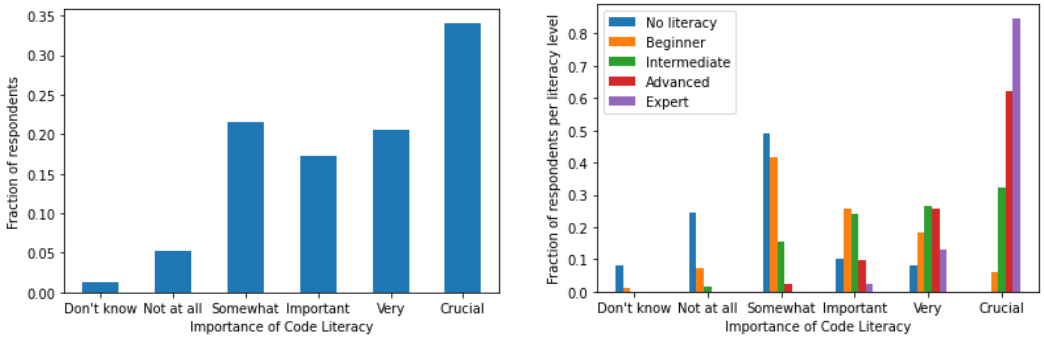

Figure 7: Distribution of the level of importance of code literacy for DH scholarship, among all 399 respondents (left) and across the five code literacy levels (right).

There are differences across background disciplines. The disciplines with the highest percentage of respondents who consider code literacy not important at all are Media Studies (12%), History (10%) and Cultural Studies (9%). So across all disciplines in our survey, the vast majority agree code literacy has a place in DH scholarship.

Historians are significantly more likely to consider code literacy as *not important at all* compared with non-historians (10% versus 3%, χ^2 with $p = 0.004$) and significantly less likely to consider it *very important* (12% versus 25%, $p = 0.002$). Respondents with backgrounds in Archival and Museum studies are significantly less likely to consider code literacy *crucial* than those without these backgrounds (7% versus 36%, $p = 0.006$). On the other hand, linguists are significantly more likely to consider it *crucial* than non-linguists (55% versus 32%, $p = 0.007$). Again we see the strong contrast between historians and linguists. Their difference in perceived importance might be related to their difference in levels of code literacy.

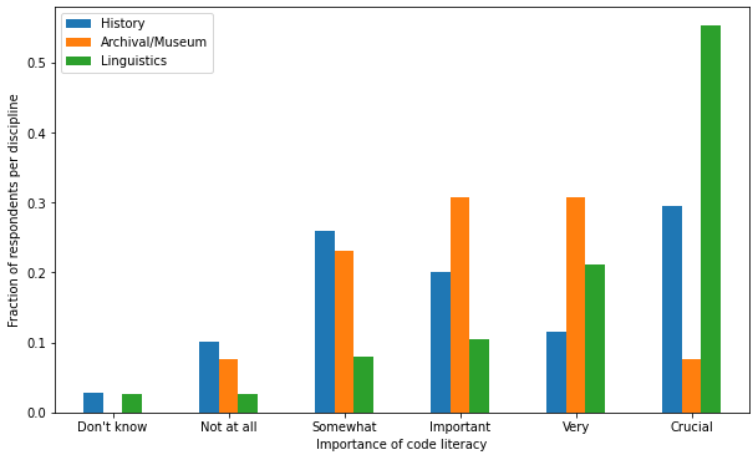

Figure 8: Distribution of respondents on importance levels per discipline for historians, linguists, and specialists in archival and medium studies.

Apart from these three humanities disciplines (represented in Figure 8), the only other disciplines with statistically significant differences are computer science and software development, which, not surprisingly, score lower on *somewhat important* (7%, χ^2 with $p = 0.02$ and 0% with p$p = 0.01$ versus 23% respectively), but higher on *very* and *crucial* (respectively 60% and 77% versus 31%, $p < 0.001$). Keeping in mind the possible self-selection bias mentioned in Section 4.1, the results suggest there is a broad consensus across a wide range of disciplines, within the humanities and beyond, that code literacy is important to many DH scholars.

5 Discussion

This paper analysed the results of a questionnaire on the definition and importance of code literacy in digital humanities. Our coding and analysis of the 399 definitions of code literacy reveals a complex, multi-layered and multi-faceted perspective on code literacy; from the basic skills of reading, interpreting, writing and using code, to publishing and reviewing code, to the different contexts in which code is created and used – the research context, the ecosystem of hardware, software, and communities and conventions – as well as the societal context relating to ethics, privacy and bias.

We found that these different aspects are related to the experience levels of respondents, with more code literate respondents providing more elaborate and nuanced definitions. This suggests that code literacy is not a single level that one reaches at a certain point, but is a set of skills that people continuously improve and extend within a particular context.

The order in which these skills are best learnt and developed is not necessarily related to the level of literacy of the respondents that mention them. The context of research in which one translates between research questions and methods on the one hand, and how that can be expressed, modelled or performed via code on the other hand, is mentioned most by respondents who identify as more code literate. However, we argue that this contextual understanding should be learnt early on, as it is the most directly relevant aspect for integrating code into DH scholarship. Many coding practitioners in the (digital) humanities, humanities researchers, and sociologists of code literacy seem to have gravitated to a similar attitude, although they differ of opinion on how to define this "contextual understanding".

A large part of the respondents think that code literacy is important for DH scholarship. However, the particular distributions we see regarding who thinks code literacy is important should also inspire questions about the reasons *why* we consider code literacy to be important. It seems safe to assume though, that the perception that code literacy is important in DH is on the rise. And yet, many respondents are dissatisfied with their own level of code literacy. This prompts the question as to whether there is currently a gap in academic curricula regarding digital humanities? And if so, would enhancing code literacy fill this gap? And what should such teaching look like according to our respondents? Questions such as these will be addressed in follow-up publications in which we will analyse and discuss the remaining questions and responses of the questionnaire.

References

Theophilus Azungah. Qualitative Research: Deductive and Inductive Approaches to Data Analysis. *Qualitative Research Journal*, 2018.

BenAmi Blau. Computer Literacy: What and Who? *Personnel*, 62(12):7–8, 1985.

Helen J. Burgess and Jeanne Hamming. New Media in Academy: Labor and the Production of Knowledge in Scholarly Multimedia. *DHQ: Digital Humanities Quarterly*, 5(3), 2011. URL http://digitalhumanities.org/dhq/vol/5/3/000102/000102.html.

Elizabeth Callaway, Jeffrey Turner, Heather Stone, and Adam Halstrom. The Push and Pull of Digital Humanities: Topic Modeling the "What is digital humanities?" Genre. *Digital Humanities Quarterly*, 14(1), 2020. URL http://www.digitalhumanities.org/dhq/vol/14/1/000450/000450.html.

Monica Chin. Kids Who Grew Up With Search Engines Could Change STEM Education Forever, 2021. URL https://www.theverge.com/22684730/students-file-folder-directory-structure-education-gen-z.

Tanya Clement. Multiliteracies in the Undergraduate Digital Humanities Curriculum: Skills, Principles, and Habits of Mind. In Brett D. Hirsch, editor, *Digital Humanities Pedagogy: Practices, Principles and Politics*. Cambridge, UK: Open Book Publishers, 2012. ISBN 9782821854031. URL http://books.openedition.org/obp/1656.

ConcordiaUniversity. Computer skills: Levels of proficiency, 2011. URL https://www.concordia.ca/content/dam/concordia/services/hr/docs/employment/guides/proficiency-computer-skills.pdf.

Juliet M. Corbin and Anselm Strauss. Grounded Theory Research: Procedures, Canons, and Evaluative Criteria. *Qualitative Sociology*, 13(1):3–21, 1990.

Adam Crymble. *Technology and the historian: transformations in the digital age*, volume 1. University of Illinois Press, 2021.

Michael Dobberstein. Computer Literacy for the Rest of Us. *Computers and the Humanities*, 27(5-6):429–433, September 1993. ISSN 0010-4817, 1572-8412. doi: 10.1007/BF01829393. URL http://link.springer.com/10.1007/BF01829393.

Amy Earhart, Matthew Kirschenbaum, Bethany Nowvisky, Katherine Harris, Vika Zafrin, Patrick John Murray, and Daniel Allington. Doing DH vs. Theorizing DH, 2016. URL https://dhanswers.ach.org/topic/doing-dh-v-theorizing-dh/#post-436.

Matthew K. Gold, editor. *Debates in the Digital Humanities*. 1. University of Minnesota Press, 2012a. doi: https://doi.org/10.5749/9781452963754.

Matthew K. Gold. Introduction: the Digital Humanities Moment. In Matthew K. Gold, editor, *Debates in the Digital Humanities*, pages 9–16. Minneapolis, MN: University of Minnesota Press, 2012b.

Matthew K. Gold and Lauren Klein, editors. *Debates in the Digital Humanities*. 5. University of Minnesota Press, 2019. doi: https://doi.org/10.5749/9781452963785.

Jan Hajic. Linguistics meets exact sciences. *A companion to digital humanities*, 42:79, 2004.

Susan Hockey. Workshop on Teaching Computers and the Humanities Courses. *Literary and Linguistic Computing*, 1(4):228–229, 01 1986. ISSN 0268-1145. doi: 10.1093/llc/1.4.228. URL https://doi.org/10.1093/llc/1.4.228.

Simon Holland and Gordon Burgess. Beauty and the Beast: New Approaches to Teaching Computing for Humanities Students at the University of Aberdeen. *Computers and the Humanities*, 26(4):267–274, August 1992. ISSN 0010-4817, 1572-8412. doi: 10.1007/BF00054272. URL http://link.springer.com/10.1007/BF00054272.

Nancy M. Ide. Computers and the Humanities Courses: Philosophical Bases and Approach. *Computers and the Humanities*, 21(4):209–215, 1987. ISSN 0010-4817, 1572-8412. doi: 10.1007/BF00517809.

Kenneth M. King. Evolution of the Concept of Computer Literacy. *Educom Bulletin*, 20 (3):18–21, 1985. URL https://eric.ed.gov/?id=EJ331749.

Ellen Konar, Allen I. Kraut, and Wilson Wong. Computer Literacy: With Ask You Shall Receive. *Personnel Journal*, 65(7):83–86, 1986.

M.C. Marino. Critical Code Studies, 2006. URL http://www.electronicbookreview.com/thread/electropoetics/codology.

Liliana Melgar, Mari Wigham, and Marijn Koolen. Programming Humanists – What Is the Role of Coding Literacy in Digital Humanities and Why Does It Matter? In *DH Benelux Conference 2019*. University of Liège, 2019. URL https://2019.dhbenelux.org/.

Nick Montfort. Exploratory Programming in Digital Humanities Pedagogy and Research. In *A New Companion to Digital Humanities*, pages 98–109. John Wiley & Sons, Ltd, 2015. ISBN 978-1-118-68060-5. doi: 10.1002/9781118680605.ch7. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118680605.ch7.

Douglas Noble. Computer Literacy and Ideology. *Teachers College Record*, 85(4): 602–614, 1984.

Robert L. Oakman. Perspectives on Teaching Computing in the Humanities. *Computers and the Humanities*, 21(4):227–233, 1987. ISSN 0010-4817, 1572-8412. doi: 10.1007/BF00517811. URL http://link.springer.com/10.1007/BF00517811.

Heather L. O'Brien and Lori McCay-Peet. Asking "Good" Questions: Questionnaire Design and Analysis in Interactive Information Retrieval Research. In *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval*, pages 27–36, 2017.

James O'Sullivan, Diane Jakacki, and Mary Galvin. Programming in the Digital Humanities. *Digital Scholarship in the Humanities*, pages 142–147, October 2015. ISSN 2055-7671, 2055-768X. doi: 10.1093/llc/fqv042. URL https://academic.oup.com/dsh/article-lookup/doi/10.1093/llc/fqv042.

Michael Piotrowski and Mateusz Fafinski. Nothing New Under the Sun? Computational Humanities and the Methodology of History. In *Proceedings of the Workshop on Computational Humanities Research (CHR 2020)*, volume 2723, pages 171–181, Amsterdam, November 2020. CEUR Workshop Proceedings. doi: urn:nbn:de:0074-2723-3. URL http://ceur-ws.org/Vol-2723/short16.pdf.

W. James Potter. The State of Media Literacy. *Journal of Broadcasting & Electronic Media*, 54(4):675–696, November 2010. ISSN 0883-8151. doi: 10.1080/08838151.2011.521462. Publisher: Routledge.

Stephen Ramsay. Programming with Humanists: Reflections on Raising an Army of Hacker-Scholars in the Digital Humanities. In Brett D. Hirsch, editor, *Digital Humanities Pedagogy: Practices, Principles and Politics*. Cambridge: Open Book Publishers, 2012. ISBN 9782821854031. URL http://books.openedition.org/obp/1642.

Stephen Ramsay. On Building. In Edward Vanhoutte, Julianne Nyhan, and Melissa Terras, editors, *Defining Digital Humanities: A Reader*, pages 243–246. Ashgate, 2013a.

Stephen Ramsay. Who's In and Who's Out. In Edward Vanhoutte, Julianne Nyhan, and Melissa Terras, editors, *Defining Digital Humanities: A Reader*, pages 239–242. Ashgate, Surrey, UK, 2013b. URL https://web.archive.org/web/20121015012254/http://stephenramsay.us/text/2011/01/08/whos-in-and-whos-out.html.

Douglas Rushkoff. *Program Or Be Programmed: Ten Commands for a Digital Age*. OR Books, 2010.

Anne B. Ryan. Post-positivist Approaches to Research. *Researching and Writing your Thesis: a Guide for Postgraduate Students*, pages 12–26, 2006.

Lothar Sachs. *Applied statistics: a handbook of techniques*. Springer Science & Business Media, 2012.

Maria Spante, Sylvana Sofkova Hashemi, Mona Lundin, and Anne Algers. Digital competence and digital literacy in higher education research: Systematic review of concept use. *Cogent Education*, 5(1):1519143, January 2018. doi: 10.1080/2331186X.2018.1519143. Publisher: Cogent OA.

Dara Tafazoli, Ma Elena Gomez Parra, and Cristina A Huertas Abril. Computer literacy: Sine qua non for digital age of language learning & teaching. *Theory and Practice in Language Studies*, 7(9):716, 2017. ISSN 1799-2591. doi: 10.17507/tpls.0709.02. Publisher: Academy Publication Co., Ltd.

Robert S. Tannenbaum. How Should We Teach Computing to Humanists? *Computers and the Humanities*, 21(4):217–225, 1987. ISSN 00104817. URL http://www.jstor.org/stable/30207392.

Rob Timans, Paul Wouter, and Johan Heilbron. Mixed Methods Research: What It Is and What It Could Be. *Theory and Society*, 48:193–216, 2019.

Joris J. van Zundert and Tara L. Andrews. Qu'est-ce qu'un texte numérique? a New Rationale for the Digital Representation of Text. *Digital Scholarship in the Humanities*, 32(suppl_2):ii78–ii88, 08 2017. ISSN 2055-7671. doi: 10.1093/llc/fqx039. URL https://doi.org/10.1093/llc/fqx039.

Joris J. Van Zundert, Smiljana Antonijević, and Tara L. Andrews. 'blackboxes' and true colour — a rhetoric of scholarly code. In Jennifer Edmond, editor, *Digital Technology and the Practices of Humanities Research*, page 123–162. Cambridge, UK: Open Book Publishers, 2020. URL https://www.openbookpublishers.com/product/1108.

Edward Vanhoutte, Julianne Nyhan, and Melissa Terras, editors. *Defining Digital Humanities: a Reader*. Ashgate Publishing, Ltd., 2013.

Annette Vee. Understanding Computer Programming as a Literacy. *Literacy in Composition Studies*, pages 42–64, 2013.

Annette Vee. *Coding Literacy: How Computer Programming Is Changing Writing*. Software Studies. The MIT Press, Cambridge, July 2017. ISBN 978-0-262-03624-5.

Staten W. Webster and Linda S. Webster. Computer Literacy or Competency? *Teacher Education Quarterly*, pages 1–7, 1985. URL `https://www.jstor.org/stable/23474573`.

Barbara M. Wildemuth. 14. sampling for extensive studies. In Barbara M. Wildemuth, editor, *Applications of Social Research Methods to Questions in Information and Library Science*, pages 116–128. ABC-CLIO, 2009.

Robert M. Zemsky. Numbers and history: The dilemma of measurement. *Computers and the Humanities*, 4(1):31–40, 1969. ISSN 00104817. URL `http://www.jstor.org/stable/30199320`.

6 Appendix A - Code Literacy Definition Aspects

Code	*Description*	*Scope note*
LOC	Level of Competence	Different levels of competence of interacting with code, from recognising, reading and basic writing, to increasingly complex aspects of these. Competencies thereby have different levels.
LOC-1	Recognize	The ability to recognize code as code, having an understanding of the general purpose of code, e.g. that code is used to tell a processor what actions to perform (processing) or to structure the content of a document (encoding)
LOC-2	Read and apply	The ability to understand the syntax of a coding language and to read a specific bit of code and figuring out what it does or what it conveys. This can include understanding data structures (like arrays, hashes, ...), databases, api's, and control-flow aspects like loops and conditionals. This also includes knowing how to apply it, e.g. by changing a few parameters or variables.
LOC-3	Write	The ability to write syntactically correct code in a specific language (processing or encoding). Correct means that it can be executed without error, but says nothing about its quality or organisation. Use this when a definition only speaks of 'writing code' without specification. We assume this level is also applied in definitions that say something like 'knowing at least one programming language'.
LOC-4	Repurpose (copy-paste/libraries), edit/modify	The ability to identify a relevant piece of existing code or code libraries and incorporating it in ones own code, or adjusting it to ones own purpose and context (beyond changing a few parameters or variables and running the repurposed code as a tool).
LOC-5	Review/evaluate	The ability to review code to decide if it corresponds to its creator's intended use and purpose, the ability to evaluate the quality of code
LOC-6	Create, test, improve and deploy	The ability to create code from scratch to solve a concrete task (either to process data or to encoding documents)
LOC-7	Paradigms and theoretical aspects of computation	The understanding of various programming paradigms and how they differ in terms of modelling and extending core programming concepts. This refers to aspects like the differences between object-oriented and functional programming, or declarative versus procedural. Theory of computation includes references to computability, P vs. NP complete, ...
LOC-M	Different levels of literacy	Use this for answers that explicitly refer to different levels of code literacy. For example: "basic literacy is being able to read and understand code, intermediate literacy is being able to write code, advanced literacy is being to write high quality code."

Table 2: Code literacy definition aspects related to Level of Competence.

Code	*Description*	*Scope note*
CU	Contextual Understanding	Understanding that code is not created or used in a vacuum, but is used within a wider context of the research it is part of, cultural attitudes and conventions and other technological aspects like environments and platforms.
CU-1	Transforming research problems to computation	Understanding how a research question or problem can be divided into increasingly smaller questions or problems, which can be more directly addressed through code or software. People mentioning this aspect might focus more on code or on the research question.
CU-2	Potential, limits, biases	Understanding how specific technologies and tools are relevant to a research problem, what they can and cannot do, when it's appropriate to use them, and how they can create, propagate or exacerbate biases in the data. This is not about social/cultural aspects of code, but about the context of the coder and their intention.
CU-3	Attitude	"The attitude towards learning to code and using and creating code, as well as to its value for and role in research. This includes things like being open-minded and confident in accepting that you maybe don't know what you're doing or that you make a lot of mistakes. This also includes attitudes around autonomy and ownership as well as attitudes towards finding help (e.g. don't be afraid to ask for help or copy from others)."
CU-4	Code ecosystem (ethics, security, privacy, maintenance, documentation, versioning, licensing, good practices)	The understanding and ability to handle aspects of ethics and privacy, security, to maintain, document and version of code, understanding licensing and open-source aspects. Knowing about the software ecosystem around code (code editors, repositories) and knowing how the web works (servers, domains, sites, etc). Knowing how to find help (in documentation but also on forums and via colleagues).
CU-5	Other references to context	Any aspect of code context that is not covered by the first four aspects
CT	Code Type	Whether the code explicitly refers to a specific type of code, either code as encoding of documents or as in processing of data.
CT-1	Encoding	A form of coding that segments and structures the content of a document (be it text, image, audiovisual or data) and identifies elements of interest
CT-2	Processing	A form of coding that is performative, in the form of processing instructions that can be run to perform some task
COM	Communication	Being able to communicate about code and collaborate with others through code (includes a.o. code sharing, pair programming, co-designing specifications, co-designing a methodology), this could also include teaching and explaining of code to others.
Other	Other aspects	Anything not related to code literacy
MIS	Misinterpreted	Misinterpretation of the question. For answers where it's clear they answered a different question or least give no definition.
NA	Not Applicable	For answers that are not captured by any of the codes

Table 3: Code literacy definition aspects related to Contextual Understanding, Code Type, Communication and aspects not related to code literacy.