Craft Ethics - Aiming for Virtue in Programming with Generative AI

Martin Jonsson
Södertörn University
martin.jonsson@sh.se

Jakob Tholander
Stockholm University
jakobth@dsv.su.se

Abstract

This paper analyses some aspects of the profound shifts in programming practice and education about by the advent of generative AI (GenAI). As GenAI tools become increasingly integrated into programming environments, they offer an approach to programming that bypasses significant aspects of the meticulous syntax-focused processes inherent in traditional programming. Instead, these tools enable a more immediate transition from problem articulation to automated solution generation, reducing the need for traditional forms of iterative problem-solving and careful focus on coding details. This paradigm shift not only challenges the foundational skills taught in programming education but also raises ethical concerns regarding aspects such as interpretability, authorship and accountability of the produced code. This involves a reevaluation of programming education and practice, suggesting a need for a reorientation to emphasise ethical and interpretative skills in programming with GenAI. Based on a series of studies on GenAI-supported programming, this paper highlights aspects relating to control, agency, and design for ethical deliberation in the evolving practices of programming with GenAI. To move towards such practice, we propose a set of design challenges based on the concept of "Craft Ethics," which emphasizes virtue, quality, and a thoughtful approach to programming and design. These challenges integrate traditional craftsmanship values into GenAI practices, ensuring that the ethical and qualitative aspects of programming are renewed and enhanced.

1. Introduction

Generative AI (or GenAI)tools based on Large Language Models such as GPT-4, Gemini, or Llama, have reached a level of functionality where they are capable of generating various forms of working programming code ranging from advanced code-completion to writing code snippets based on natural language input, as well as advanced debugging, reviewing, and rewriting code. These tools are rapidly being taken into practice both among professional programmers as well as in a range of educational settings and have given rise to renewed discussions of how programming finally may reach the point of being democratized and accessible to a broader audience, mirroring the hopes of early high-level languages such as Smalltalk and Logo. More recently, Krings et al, (2023) for instance, envision what the future of programming might be like when AI tools become powerful enough to generate working programming code based on open-ended specifications, allowing for '*everyone to be programmers*', potentially turning programming into a practice that is more about developing ways to support humans in "how to best educate the machine" (Welsh, 2023), than about writing code in traditional ways. The new programming tools based on generative AI are certainly impressive but are also limited in many ways, and we still need to better understand how they are used in actual practices of programming, and what new kinds of interactions and collaborations need to be supported in the emerging processes of co-creation and co-coding between humans and AI (Jonsson & Tholander, 2022). In their current form, interaction with these tools is primarily conducted through various forms of text or chat-based natural language interactions which provides powerful opportunities for open-ended forms of interaction (Prather et al., 2023). However, studies have pointed to the need to also explore and understand the new practices that emerge with the use of these kinds of tools, and how the interactions and practices are shaped by the properties and behaviours of both the underlying models and the programming tools that leverage their functionalities to the end users. (Denny, Kumar, & Giacaman, 2023) (Jeuring, Groot, & Keuning, 2023) (Jonsson & Tholander, 2022). The new programming practices and the new ways to create digital interactive materials also challenge established truths about what are good and virtuous

ways of programming, and how to achieve good quality and desired behaviours of the generated code materials. Based on research in this area, we will discuss some new emerging phenomena relating to GenAI-assisted programming, and identify a set of ethical challenges relating to the use of such tools in various programming activities. To elaborate on these challenges we will articulate programming practice as a form of *craftsmanship* based on concepts from theories on craft and design practices, such as design judgment, and craftsmanship of risk. Finally, we will outline a *Craft Ethics* for GenAI-assisted programming, highlighting stances and approaches that might be beneficial for mitigating some of the identified challenges.

2. Background

An increasing amount of studies have explored the effectiveness of novel AI tools such as CoPilot for generating working programming code and how they integrate with and influence various aspects of the practices of programming, such as learning (Jeuring et al., 2023), productivity(Bird et al., 2023), use (Kazemitabaar et al., 2023), correctness and performance (Denny et al., 2023), etc. Bird, et al (2023)for instance noted how programmers using CoPilot spent less time analysing code errors, but at the same time seemed to develop less of an understanding of how or why a particular piece of code worked the way it did. They also noted that the perceived productivity among users of CoPilot was actually higher than it actually was. Relatedly, several other studies, (e.g. (Prather et al., 2023) (Barke, James, & Polikarpova, 2022)) show that users of Generative AI tools for programming were required to put additional efforts into actions such as instructing and articulating how the code should work, as well as on reading, reviewing, debugging and tweaking the code that has been generated, while less efforts were put into spending time on writing actual lines of code. This has been argued as pointing to a potential larger shift in the nature of future practices of programming, making programming less about writing code from scratch, and more about being able to interpret, understand and tweak code generated by AI systems. This has even somewhat provocatively been phrased as that we are facing "the end of programming" (Welsh, 2023), suggesting that the future of programming will change from programming being a process of writing syntactically and logically correct lines of code, to a process of being skilled in expressing and instructing the system in ways that makes it generate the code that one wants. A few recent examples of studies have explored how to design for novel kinds of programming with Generative AI may look beyond current interaction models, for instance, through prototypes that use the ability of ChatGPT to generate code based on visual and textual sketches of a program (Lewis, 2019) (Ban & Hyun, 2020) and how to create programming assistance through conversational interactions when programming using LLMs (Ross, Martinez, Houde, Muller, & Weisz, 2023).

2.1. Emerging Programming Practices

A limited number of studies have specifically addressed issues and specific patterns of use and interaction that emerge with generative AI tools in programming, and to identify emerging design opportunities (Jayagopal, Lubin, & Chasins, 2022) . Barke et. al. (2022) suggest that interaction with AI-based programming assistants is *bimodal*. Firstly, it involves what they call an *acceleration mode*, denoting how the tool simplifies and speeds up the process of writing a particular piece of code, and secondly, it involves an *exploration mode* denoting how the tool supports the programmer to explore different options when hesitating on where to go next. In a similar fashion, Jonsson and Tholander (2022) discuss the notion of *friction* and point to two ways that it can be understood in processes of programming with code generation tools. These tools may work to *remove some of the friction* that a novice programmer experiences in programming, e.g. simplifying through the generation of initial suggestions and ideas, and alternative solutions to various problems. They may also *induce friction* by slowing down the process, suggesting unexpected routes that make users reflect and explore alternative solutions. Similarly, Prather et al (2023) identified two interaction patterns in novice programmers use of CoPilot. The first, called *sheparding,* describes how students worked through continuous tweaking of prompts that guided Copilot through making its auto-generated code proposals to close up on a working solution, rather than creating code from scratch. The second, called *drifting*, regarded how students tried to make use of the system's suggestions despite not getting closer to a working solution, thereby drifting between various

suggestions and eventually getting lost. A form of interaction called *slow accept* was also identified, through which users typed in the suggested code snippet themselves, rather than merely accepting it. This allowed them to better understand the purpose and workings of the proposed code. This growing number of empirical studies of the usage of generative AI for programming have started to identify and conceptualise the different ways that incorporate these technologies in their practices However, there is still limited understanding of how these kinds of interactions are shaped by the specific properties of the user interfaces, and the opportunities of novel designs for alternative forms of use.

2.2. From Programming Tools to Programming Partnerships

Some of the findings of previous studies have raised discussions about how these shifts in the nature of programming practice and interaction may fundamentally recast the relations between programmers and the technologies they are entangled in the creation of programming code. Both Welsch (2023) and Jonsson and Tholander (2022), point to how programmers have to give up parts of their autonomy as programmers and hand over control over the coding process to the AI tool (Bird et al., 2023). Programming tools and practices would then require the design of ways to support new forms of partnerships between human and artificial partners, new kinds of *pair-programmers*, in which the AI is considered an active collaborator, and not merely a non-agentive tool (Lawton, Grace, & Ibarrola, 2023) . Such a reframing of the relations in programming practices requires a reconsideration of each actor's contribution to and role in the collaborative process. As in other forms of human-human collaboration, this is not always what can expected or specifically designed for. All actors in various ways and to different extent contribute to bringing the overall process forward. This might involve a perspective where code develops through shared and collaborative efforts, including the wrongs and rights, mishaps and points of view of either actor. Conceptually, this aligns with notions of co-creativity and co-creation that have emerged in research in perspectives on HCI, which frame how interactive and smart technologies may be considered as co-participants in creative processes (Wakkary, 2021), rather than as mere tools controlled by human actors. This further relates to conceptual and theoretical work in HCI (Devendorf & Ryokai, 2015), around notions such as more-than-human design, and co-performance (Kuijer & Giaccardi, 2018) and machine agency (Pickering, Engen, & Walland, 2017) which suggest a reconsideration of the view of humans as the sole source of creative agency, to also see various forms of interactive and digital technologies as sharing the agency to spur ideas and creative expressions. These theories highlight the entanglement of human agency with agencies originating from non-human entities, implying that programming and design are inherently *relational* forms of action. Programming and design are then not to be viewed as exclusively human actions, but co-constituted in entanglements of human and artificial agencies. We argue that the intersection between current technical developments and these theoretical perspectives on human-machine relationships opens up for interesting discussions around responsibility, control, and ethics in the new emerging practices of GenAI-assisted programming.

3. Ethical Challenges in GenAI-assisted Programming

As has been shown above, generative AI tools have the potential to radically redefine what programming is and how to do it. A large part of the research on GenAI-assisted programming, focuses on how to increase performance and productivity, reducing errors, and creating a more effective collaboration between humans and AI. What has been less discussed are the ethical ramifications of this shift of practice. Some concerns have been raised, for example, the risk that programmers might lose their jobs (Welsh, 2023), or that these new tools will have limited accessibility for broader groups of potential users, increasing gaps both between groups in society, as well as globally between the global north and developing countries. In the following, we will omit ethical issues on this societal macro level and instead focus on some challenges that arise on the level of the individual users and the emerging practices of GenAI-assisted programming.

As an increasing amount of the code the programmers produce is automatically generated, an inherent consequence is that the programmer has less control over the way the code is written and how it executes. For cases such as basic auto-completion for setting up simple programming constructs, this might not be problematic, as the code is easy to read and interpret. However, as these tools get more advanced, we

can assume that the generated code will be more challenging to read and, as all programming code, hide and abstract away important aspects of the code. This will give less control over the code and require the programmers to trust how the system is built and how it is executed. While large software development projects have similar consequences in that there is not one single individual in control of every piece of functionality in the system, we argue that when we to an increasing extent rely on AI-generated code, we will encounter unknown consequences of how it executes. We have identified four particular ethical challenges that may occur in the concrete GenAI-based programming practices: 1) dealing with errors and imperfections, 2) explainability and accountability of code, 3) bias and drifting in interaction, and 4) methods and approaches for bias and testing

3.1. Errors and Imperfections

Being able to interpret and find errors in generated code is reinforced by the fact that it is well-known and broadly acknowledged that despite rapid improvements, LLM-based tools still generate textual output that is far from perfect. Text generated by LLMs is commonly flawed or even totally wrong, and the same goes for the generation of programming code. A recent study from Kabir et al. (2024) shows that 52% of ChatGPT answers to programming questions contain incorrect information, but that users still prefer using ChatGPT for getting answers due to the comprehensiveness and well-articulated style of language often used. Concerning errors in generated programming code, Weisz, et al (2021) discuss how programming differs significantly from other forms of text-based practices, since small errors in programming may have much more significant consequences compared to small errors in a piece of argumentative or fictional text. Errors in the logic of the code may fundamentally change its workings, thus requiring that any piece of generated code always needs to be thoroughly reviewed. In Barke et al's (2022) study of novice programmers, they point to a need for a shift in programming practice from writing code, to reading and interpreting generated code, and how this enables students to work at a higher level of abstraction and spend their cognitive effort on thinking about the semantics of the program, rather than on details of the syntax and logic. Vaithilingam et al, (2023) as well as Jonsson and Tholander (2022) further discuss the consequences of the imperfection of the code that tools such as CoPilot or ChatGPT in relation to the design of novel forms of interaction for working with such code generation. Importantly, these studies found that despite code generation being imperfect, it still proved useful for the ongoing problem-solving and creative process, for instance as starting points when being stuck or as 'pieces of code to think with' in order to carry the overall process forward. It is likely that the performance of the LLM-models will improve, resulting in fewer errors and unexpected code. This is obviously a positive development, but it might lead to an over-reliance and poorly grounded expectations on the quality of the code that AI tools may generate, leading to the users omitting critical reflection and evaluation of the generated output. Moreover, this also builds on an assumption that the code is generated on a well-defined intention of the final execution. However, many programming projects are iteratively driven, in which the code works as a co-creative tool in developing the idea of what is to be designed and built.

3.2. Explainability and Responsibility

A commonly discussed problem with respect to AI and especially large language models, is that these systems are opaque, giving no explanation as to why a certain input generates a particular output. Even the researchers who designed these systems often have a hard time explaining the intricate links between input or output, or as bluntly put by (Welsh, 2023): *"Nobody actually understands how large AI models work."* This raises issues regarding who is responsible for the effects of the outputs of the AI system. To what extent can the user be in control of the output and the process? In the case of programming, the user might not have the abilities and training to fully understand the generated code or it may rely on code or data generated elsewhere. This raises the issue of how to design mechanisms in these systems that allow programmers to interpret and understand code in a fashion that allows them to be accountable for the behaviour of code. This concerns both an understanding of the workings of the generated code as such, but also involves challenges in how the AI tools interpret the instructions and input provided by the user, and how the generated output aligns with the user's ideas or more or less well-articulated

intentions. Partly this can be understood as a problem that can be mitigated by more powerful and better-tuned AI models, but there are dimensions of this problem that cannot be solved by engineering. The power of these tools resides in that they can translate a short instruction or a few lines of code into a solution rich with details that go far beyond what was specified in the input. If every aspect and detail that should be included in the generated output has to be specified in the input instructions, the gains and usefulness of the tools become comparable to just writing the code on your own. Projecting the usage of GenAI tools into the future, it is not hard to imagine usages in which the AI-generated output is too detailed or complex to fully comprehend even for expert users. If such code causes harm or expresses unwanted norms, it raises concerns regarding the responsibility for the emerging effects. This would call for more rigorous and efficient testing methods. However, as is well documented for instance in critical algorithm studies, the effects of these systems are not intentional or predictable beforehand but emerge out of interaction with users and the data that they operate upon.

3.3. Bias and Drifting

Bias in AI and machine learning systems is a well-known issue, extensively discussed concerning its ethical implications. Bias typically arises when training data is flawed due to prejudices related to gender, race, and other factors, leading AI systems to reproduce these flaws. In the context of GenAI-assisted programming, bias can also be significant. If the AI is trained on biased historical code bases, these biases can be reflected in its suggestions. Training data that is not diverse and predominantly includes code from specific demographics may cause the AI to favour certain coding styles or solutions. The architecture and algorithms used can further introduce bias by prioritizing specific data patterns. This may affect code completion and the recommendations that are generated, potentially reflecting bias or stereotypes thereby limiting the opportunities for novel and creative solutions. Additionally, the AI tools might generate code that is less accessible to certain user groups if it is optimized based on specific demographics, making it less usable for other groups. Bias can also manifest in the AI's performance across different development environments, particularly if certain coding environments or tools are more represented in the training data. Lastly, if the AI is predominantly trained on code from a specific region, it may fail to suggest best practices common in other regions. The potential biases of GenAI programming tools may play out in different ways in different patterns of interaction, for instance in relation to Prather et al's (2023) notion of *drifting* - a phenomenon where the AI-generated code gradually drifts away from the users' initial intentions. Drifting illustrates how a biased GenAI-programming tool can become ethically challenging, as the activity must always be understood as a joint effort between human and AI actors, and where the activity and generated outcomes is a result of a mangle of agencies (Pickering et al., 2017) attributed to both humans and the tools, models and user interfaces involved in the interaction.

3.4. Control and Testing

In a recent, still unpublished study comparing novice programmers with experienced programmers in their use of GenAI tools, a number of differences emerged concerning how the two groups manage control and testing. One critical difference between novice and experienced programmers concerned their previous coding experiences and how these influenced how well they could evaluate and test the code that the tools generated. As one would expect, the more experienced the users were, the more elaborate they were in their judging of the workings of the code that was generated. Less experienced users on the other hand, primarily evaluated the code by *executing* it in order to be able to experience and reflect upon how the output behaved in relation to what they had expected. To conceptualise this, two dimensions of interaction and use were proposed; *code-focused* versus *execution- and experience-focused* interactions. The code-focused interactions align to a large extent with existing programming practices and often involve the careful writing of prompts or structuring of sections of pseudo-code that the tools would use to generate a specific piece of code, including the interactional efforts required to understand and interpret the code that was generated. This also aligns with how previous studies (Barke et al., 2022) have argued for how generative AI tools contribute to a shift in the focus of programming practices from primarily involving various processes oriented to the writing of code, to the increased

engagement of competencies of skillfully reading, interpreting and tweaking generated code. The novice users more commonly engaged in interactions that were *execution- and experience-focused*, involving the writing of prompts, articulation of ideas, or expression of intentions that had the purpose of getting the tool to generate a piece of code that would result in a particular kind of execution - such as a specific visual interaction on the screen - without expecting an explicit idea of the specific characteristics or qualities of the actual code that the execution would be based upon. Projecting the use of GenAI tools for programming into the future, it is not unreasonable to assume that the textual code representation will be less important and less in focus than in current programming practices. This shift in how code is evaluated and tested might be ethically problematic as the focus on execution omits the scrutiny and detailed examinations of all the generated materials.

4. A Turn to Craft

Welsch (2023) claims that generative AI provides a "seismic shift" for people's relation to computation and programming, replacing predictable static processes, governed by instruction sets and decidability, with an understanding of computation as temperamental, mysterious and unpredictable, more similar to humans than we have seen before. Likewise, generative AI tools offer an approach to programming that bypasses significant aspects of the meticulous syntax-focused processes inherent in traditional programming. Instead, these tools enable a more immediate transition from problem articulation to automated solution generation, reducing the need for traditional forms of iterative programming practice and careful focus on details of syntax and logic. This paradigm shift raises ethical concerns regarding aspects such as authorship and accountability of the produced code, and how programming processes are controlled. In these emerging programming practices, there is a need to establish common understandings of what constitutes good practice, how quality is maintained, and how to aim for virtue. Programming practices have traditionally been characterised as cognitive endeavours rooted in logic and scientific scrutiny, highlighting computational thinking as a particularly critical competence, such as decomposition of problems into parts and working systematically towards a solution. An alternative, albeit not necessarily contradicting way of describing programming practice, is through the notions of crafting, and craftsmanship. This perspective on programming has previously been put to the fore by the software craftsmanship movement (Sundelin, Gonzalez-huerta, Wnuk, & Gorschek, 2021) that emerged in the early 2000s (McBreen, 2002) as a response to the industrialization of software development. This view characterises programming not merely as a technical-logical endeavour but with resemblances to art practices that require a deep commitment to quality, detail, and material. A recent literature study on the software craftsmanship movement (Sundelin et al., 2021) summarizes the core ideas behind this movement, highlighting both a focus on particular qualities of software architectures, such as simplicity, minimalism, and layered architectures, as well as a focus on processes and organisation of work highlighting iterative design with a strong focus on testing and iterative refinement. Cultural dimensions are also highlighted, such as values relating to professionalism, like pride, humility and accountability. As noted in previous chapters, GenAI-supported programming might result in practices where the care to detail is replaced by higher-level design considerations and automated manufacturing processes with a more direct route from high-level instructions to a final result. Such a process runs the risk of missing out on important considerations and attendance to the details that shape key qualities of what is being created. Therefore, we propose that the notion of software craftsmanship need to be reconsidered and reframed to align with the new directions that programming practices are taking. In order to do this we have to articulate the specific aspects of craftsmanship in line with the novel AI tools and ways of working with them, and how these can be stimulated through novel tools and methods.

4.1. Design judgment and reflection in action

Many of the ethical challenges outlined in the previous chapter highlight the need for judgment and critical evaluations to be integral parts of the programming practice. In his seminal work on the Reflective Practitioner, Donald Schön (1987) describes how professional designers continuously engage in a *reflection-in-action*, referring to the process by which they think about and critically analyze their actions while they are performing them. This concept emphasizes the importance of real-time reflection

to adapt and respond to complex and dynamic situations effectively. Unlike traditional forms of reflection that occur after the fact ("reflection-on-action"), reflection-in-action involves immediate, intuitive problem-solving and the ability to adjust one's approach on the fly based on the emerging circumstances and feedback. Nelson and Stolterman (2014) further elaborate on such designerly ways of reflecting in action by proposing the notion of *design judgment* as an intrinsic and important skill in design practices. Design judgment is described as a series of both conscious and subconscious judgments and deliberations on a broad range of different aspects. Parts of these judgments can be described as intellectual judgments, which could concern actively making sure that the design results live up to a "desiderata" of stated quality criteria and standards. Another dimension of design judgment is described as design volition, focusing on using one's own will to pursue desired ends, referring to a particular form of judgment-making related to the processes that bring new things into existence. These types of judgments typically do not rely on a science of measurement to determine an objective outcome. Rather it is the ability to gain subconscious insights abstracted from experiences and reflections from situations that are complex indeterminate, indefinable and paradoxical. Design judgment is here described as a process of taking in the whole, in order to formulate a new whole. Nelson and Stolterman identify design judgment as comprised of the following particular kinds of judgments:

- *Framing judgment* – defining and embracing the space of potential design outcomes and the direction that the design process will initially take
- *Default judgments* – a nearly automatic response to a triggering situation representing a form of "bodily knowing" and an application of high-level skill without conscious deliberation
- *Appreciative judgment* – determining what is to be considered background and what requires attention as foreground, assigning importance to some things and considering other aspects as part of the context.
- *Appearance judgment* – Aesthetic judgments concerning the material substance and temporal experience or the fundamental character of the design.
- *Quality judgments* – deliberations relating to craftsmanship, connoisseurship or artistry, for example highlighting precision and skill in crafting and shaping materials. Quality judgments can be understood as a quest for excellence in the making of things.
- *Instrumental judgments* – The choice and mediation of means within the context of prescribed ends. These judgments take technology into consideration and concern techniques and what instruments to use to determine what are realistic possibilities.
- *Navigational judgment* – making the right choices in an environment that is complex and unpredictable. Securing the desired state of affairs by staying on track and proceeding in the right direction, knowing when to follow the rule book and when to leave it aside.
- *Compositional judgment* – bringing things together in a relational whole, and concerns aesthetic, ethical as well as sensual considerations.
- *Connective judgment* – establishing interconnections among things so that they create functional assemblies, creating synergies and emergent qualities.
- *Core judgments* – The cases where we "know what is right" without being able to argue in a rational way. A composite of meanings and values, formed during the experience of living.
- Mediative judgment – balancing the different types of designer judgments to orchestrate how the whole should be brought together.

Even though programming has often been described as a craft practice, working with code and interactivity as design materials, all aspects of design judgment as described above might not be immediately applicable to computer programming. Many of the judgments are however relevant and a subset of these judgments will be a fundamental contribution to the forthcoming suggestion of a Craft Ethics for GenAI-assisted programming.

4.2. Workmanship of Risk

David Pye's work, *The Nature and Art of Workmanship* (Pye, 1968) from 1968 explores the philosophy and practice of craftsmanship, emphasizing the importance of skill and human touch in creating objects. Pye here paints a romanticized image of traditional craft and craftsmanship as a reaction to the increasing industrialisation of design and mass production of goods. He distinguishes between "workmanship of risk," where the quality of the outcome is directly influenced by the maker's skill and decisions during the creation process, and "workmanship of certainty," where the outcome is predetermined by machine processes, factory production, and automation. This distinction can be directly applicable to the case of AI-generated code which could be described as articulating a "design" or description of the wanted output, that is provided to the machine, which automates the production of the code. For such processes, you are required to know exactly what qualities you want when you start the manufacturing process, as they cannot emerge as part of the process. Pye instead advocates a craftsmanship of risk, where the end result continuously depends on the mastery as embodied in persons, reflected in a capacity for judgment, and expressed in problem-finding rather than problem-solving. But the main concern is taking care and retaining control of the work process, and as a result of this, accepting responsibility for the end result. This stance towards making with machines, as well as coding with AI, reflects a more ethical practice, where the programmer becomes an integral part of the entire programming process.

4.3. Making with Head, Heart and Hand

Cheatle & Jackson (2023) explores traditional pre-digital craft practices and highlights qualities of craft practices to be embraced in digital creative practices, such as craft's holistic ways of making with *'head, heart, and hand'* and craft's distinctly collaborative and embodied practice styles. Their conceptualisation of craft accounts for ways of making that are based on a rich and dynamic concept of personhood, a 'whole self' approach, where mindfulness and cognitive processes are combined with emotional dimensions and the physical and skilful workings of the body in equal measure. "The heart" in this understanding of craft represents a driving force that moves work forward, founded on respect for labour, hard work, self-reliance, community support, perseverance, and excellence in work. The heart here symbolizes a type of personal investment that sustains work through challenges and failures, as well as the pursuit of building a better world through care, commitment, trust, responsibility, respect, and knowledge. This embodied and holistic perspective on making, to some extent aligns with the accounts of design judgments accounted for above, referring to a "bodily knowing" based on internalised experiences.

4.4. Towards a Craft Ethics in GenAI-Assisted Programming

Based on how virtue, ethics and judgment are articulated in relation to craft and design practices, we provide a first outline of a new ethical framework, *Craft Ethics* for GenAI-assisted programming, in an attempt to highlight the need for a common understanding of *craftsmanship* and *judgment* for the emerging practices involving programming with generative AI-based tools. Craft ethics builds on the care given to details and quality that are intrinsic to most craft and design practices, and the conscious and unconscious judgments and deliberations that take place in these processes. As part of this framework, we propose an alternative understanding of design judgment, in terms of *programming judgment*. Central judgments in GenAI-assisted programming are: *Framing judgments* - Being clear of the aims and directions and being able to articulate them in the co-creative activities with the AI tools. *Quality judgments* - Continuously monitoring and evaluating the joint performance between human and AI in a quest for excellence. *Instrumental judgments* - an internalised understanding of the capabilities and limitations of the AI tools. *Navigational judgments* - Monitoring the co-creation process with a critical awareness towards drifting in unwanted directions. *Co-creative judgments* - critically monitoring how different

agencies come into play in the joint activities, and how the interplay works towards a whole. Another dimension of the Craft Ethics framework we refer to as *riskful programming*, encouraging practices that emphasize flexibility, creativity, and adaptability, often at the expense of predictability and control. In riskful programming with GenAI, the programmer accepts full responsibility for AI-generated outcomes and is forced to engage intimately with the co-creative process to mitigate the potential harms that might appear. The final dimension of the Craft Ethics framework is *programming with head, hands and heart*, highlighting programming as an embodied practice with intrinsic ethical dimensions, with an overarching ambition of programming as a pursuit of building a better world. A craft ethics framework could be understood as a set of guiding principles to support practitioners in how to engage in this particular form of programming activities in an ethical and sustainable way. The framework could also be used to inform the design of new tools and user interfaces for GenAI-supported programming, for example making room for and encouraging programming judgment and riskful programming approaches.

5. Conclusions

In this paper, we have tried to outline some aspects related to the new emerging practices concerning GenAI-assisted programming, as well as a number of ethical challenges related to these practices. The need for a new understanding of craftsmanship for programming with generative AI tools was highlighted, discussing the need to support design judgment in different forms. An initial draft of a craft ethics framework was formulated, consisting of a) programming judgment, b) riskful programming, and c) programming with head, hand and heart.

6. References

Ban, S., & Hyun, K. H. (2020, March). 3D Computational Sketch Synthesis Framework: Assisting Design Exploration Through Generating Variations of User Input Sketch and Interactive 3D Model Reconstruction. *Comput. Aided Des.*, *120*(C). Retrieved 2024-05-02, from https://doi.org/10.1016/j.cad.2019.102789 doi: 10.1016/j.cad.2019.102789

Barke, S., James, M. B., & Polikarpova, N. (2022, October). *Grounded Copilot: How Programmers Interact with Code-Generating Models.* arXiv. Retrieved 2024-01-24, from http://arxiv.org/abs/2206.15000 (arXiv:2206.15000 [cs])

Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., & Gazit, I. (2023, May). Taking Flight with Copilot. *Commun. ACM*, *66*(6), 56–62. Retrieved 2024-01-30, from https://dl.acm.org/doi/10.1145/3589996 doi: 10.1145/3589996

Cheatle, A., & Jackson, S. (2023, October). (Re)collecting Craft: Reviving Materials, Techniques, and Pedagogies of Craft for Computational Makers. *Proc. ACM Hum.-Comput. Interact.*, *7*(CSCW2), 250:1–250:23. Retrieved 2024-02-21, from https://dl.acm.org/doi/10.1145/3610041 doi: 10.1145/3610041

Denny, P., Kumar, V., & Giacaman, N. (2023, March). Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (pp. 1136–1142). New York, NY, USA: Association for Computing Machinery. Retrieved 2024-02-29, from https://doi.org/10.1145/3545945.3569823 doi: 10.1145/3545945.3569823

Devendorf, L., & Ryokai, K. (2015, April). Being the Machine: Reconfiguring Agency and Control in Hybrid Fabrication. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 2477–2486). New York, NY, USA: Association for Computing Machinery. Retrieved 2021-12-21, from https://doi.org/10.1145/2702123.2702547 doi: 10.1145/2702123.2702547

Jayagopal, D., Lubin, J., & Chasins, S. E. (2022, October). Exploring the Learnability of Program Synthesizers by Novice Programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (pp. 1–15). Bend OR USA: ACM. Retrieved 2024-05-02, from https://dl.acm.org/doi/10.1145/3526113.3545659 doi: 10.1145/3526113.3545659

Jeuring, J., Groot, R., & Keuning, H. (2023, November). What Skills Do You Need When Developing

Software Using ChatGPT? (Discussion Paper). In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research* (pp. 1–6). Koli Finland: ACM. Retrieved 2024-04-25, from https://dl.acm.org/doi/10.1145/3631802.3631807 doi: 10.1145/3631802.3631807

Jonsson, M., & Tholander, J. (2022, June). Cracking the code: Co-coding with AI in creative programming education. In *Proceedings of the 14th Conference on Creativity and Cognition* (pp. 5–14). New York, NY, USA: Association for Computing Machinery. Retrieved 2023-11-30, from https://dl.acm.org/doi/10.1145/3527927.3532801 doi: 10.1145/3527927.3532801

Kabir, S., Udo-Imeh, D. N., Kou, B., & Zhang, T. (2024, May). Is Stack Overflow Obsolete? An Empirical Study of the Characteristics of ChatGPT Answers to Stack Overflow Questions. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (pp. 1–17). New York, NY, USA: Association for Computing Machinery. Retrieved 2024-05-29, from https://dl.acm.org/doi/10.1145/3613904.3642596 doi: 10.1145/3613904.3642596

Kazemitabaar, M., Hou, X., Henley, A., Ericson, B. J., Weintrop, D., & Grossman, T. (2023, November). How Novices Use LLM-based Code Generators to Solve CS1 Coding Tasks in a Self-Paced Learning Environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research* (pp. 1–12). Koli Finland: ACM. Retrieved 2024-04-25, from https://dl.acm.org/doi/10.1145/3631802.3631806 doi: 10.1145/3631802.3631806

Krings, K., Bohn, N. S., Hille, N. A. L., & Ludwig, T. (2023, April). "What if everyone is able to program?" – Exploring the Role of Software Development in Science Fiction. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (pp. 1–13). Hamburg Germany: ACM. Retrieved 2024-05-02, from https://dl.acm.org/doi/10.1145/3544548.3581436 doi: 10.1145/3544548.3581436

Kuijer, L., & Giaccardi, E. (2018, April). Co-performance: Conceptualizing the Role of Artificial Agency in the Design of Everyday Life. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (pp. 1–13). New York, NY, USA: Association for Computing Machinery. Retrieved 2022-01-05, from https://doi.org/10.1145/3173574.3173699

Lawton, T., Grace, K., & Ibarrola, F. J. (2023, July). When is a Tool a Tool? User Perceptions of System Agency in Human–AI Co-Creative Drawing. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference* (pp. 1978–1996). New York, NY, USA: Association for Computing Machinery. Retrieved 2024-02-29, from https://doi.org/10.1145/3563657.3595977 doi: 10.1145/3563657.3595977

Lewis, C. (2019, April). Why can't programming be like sketching? In *Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming* (pp. 1–6). Genova Italy: ACM. Retrieved 2024-05-02, from https://dl.acm.org/doi/10.1145/3328433.3338060 doi: 10.1145/3328433.3338060

McBreen, P. (2002). *Software craftsmanship: The new imperative*. Addison-Wesley Professional. Retrieved 2024-05-30, from https://books.google.com/books?hl=sv&lr=&id=C9vvHV1lIawC&oi=fnd&pg=PR13&dq=%5B57%5D+Pete+McBreen.+2002.+Software+Craftsmanship:+The+New+Imperative.+Addison-Wesley.&ots=pQ_x2wbVfN&sig=J9QWqgl4n8pmugreoPI1pl_d0TM

Nelson, H. G., & Stolterman, E. (2014). *The design way: Intentional change in an unpredictable world*. MIT press. Retrieved 2024-05-30, from https://books.google.com/books?hl=sv&lr=&id=lr34DwAAQBAJ&oi=fnd&pg=PR9&ots=UFve8ln4P5&sig=kJYyFDkUsZ7Ygq0Q38vmMWu_O_s

Pickering, J. B., Engen, V., & Walland, P. (2017). The Interplay Between Human and Machine Agency. In M. Kurosu (Ed.), *Human-Computer Interaction. User Interface Design, Development and Multimodality* (pp. 47–59). Cham: Springer International Publishing. doi: 10.1007/978-3-319-58071-5_4

Prather, J., Reeves, B. N., Denny, P., Becker, B. A., Leinonen, J., Luxton-Reilly, A., ... Santos, E. A.

(2023, November). "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans. Comput.-Hum. Interact.*, *31*(1), 4:1–4:31. Retrieved 2024-02-20, from https://doi.org/10.1145/3617367 doi: 10.1145/3617367

Pye, D. (1968). *The Nature and Art of Workmanship*. Cambridge University Press.

Ross, S. I., Martinez, F., Houde, S., Muller, M., & Weisz, J. D. (2023, March). The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (pp. 491–514). Sydney NSW Australia: ACM. Retrieved 2024-04-25, from https://dl.acm.org/doi/10.1145/3581641.3584037 doi: 10.1145/3581641.3584037

Schön, D. A. (1987). *Educating the reflective practitioner: Toward a new design for teaching and learning in the professions*. San Francisco, CA, US: Jossey-Bass. (Pages: xvii, 355)

Sundelin, A., Gonzalez-huerta, J., Wnuk, K., & Gorschek, T. (2021, September). Towards an Anatomy of Software Craftsmanship. *ACM Trans. Softw. Eng. Methodol.*, *31*(1), 6:1–6:49. Retrieved 2024-05-30, from https://dl.acm.org/doi/10.1145/3468504 doi: 10.1145/3468504

Vaithilingam, P., Glassman, E. L., Groenwegen, P., Gulwani, S., Henley, A. Z., Malpani, R., … Yim, A. (2023, May). Towards More Effective AI-Assisted Programming: A Systematic Design Exploration to Improve Visual Studio IntelliCode's User Experience. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (pp. 185–195). Melbourne, Australia: IEEE. Retrieved 2024-02-29, from https://ieeexplore.ieee.org/document/10172834/ doi: 10.1109/ICSE-SEIP58684.2023.00022

Wakkary, R. (2021). *Things We Could Design: For More Than Human-Centered Worlds*. MIT Press.

Weisz, J. D., Muller, M., Houde, S., Richards, J., Ross, S. I., Martinez, F., … Talamadupula, K. (2021, April). Perfection Not Required? Human-AI Partnerships in Code Translation. In *26th International Conference on Intelligent User Interfaces* (pp. 402–412). New York, NY, USA: Association for Computing Machinery. Retrieved 2024-01-30, from https://dl.acm.org/doi/10.1145/3397481.3450656 doi: 10.1145/3397481.3450656

Welsh, M. (2023, January). The End of Programming. *Commun. ACM*, *66*(1), 34–35. Retrieved 2024-04-30, from https://dl.acm.org/doi/10.1145/3570220 doi: 10.1145/3570220