Cracking the code: Co-coding with AI in creative programming education

Martin Jonsson
martin.jonsson@sh.se
Södertörn University
Huddinge, Sweden

Jakob Tholander
jakobth@dsv.su.se
Stockholm University
Stockholm, Sweden

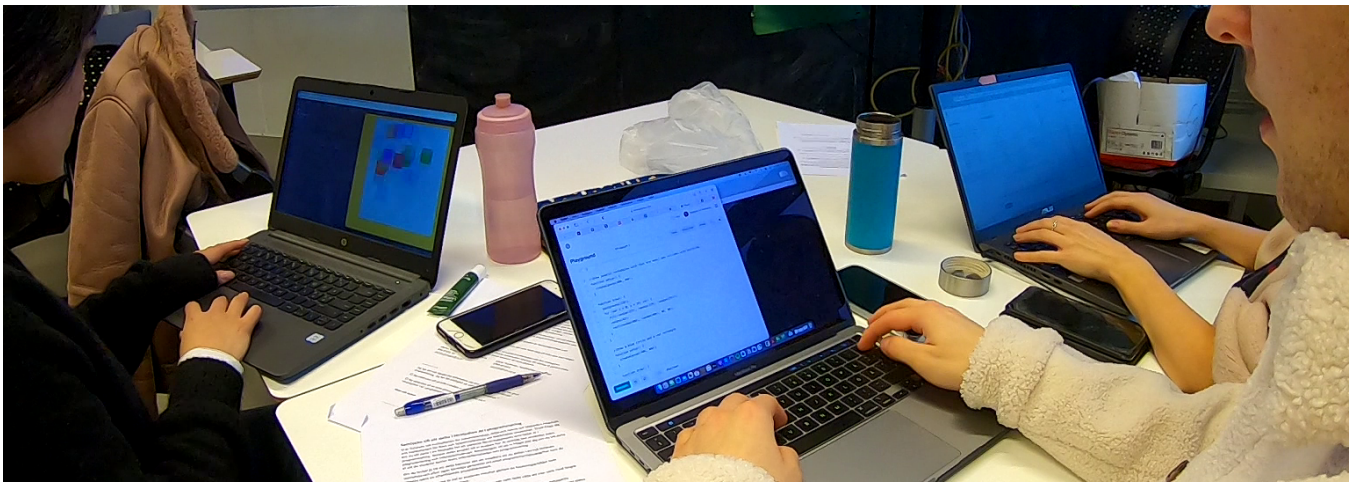

Figure 1: Participants engaged in co-creating computational graphics with the help of generative machine learning

ABSTRACT

This paper presents a study of a group of university students using generative machine learning to translate from natural language to computer code. The study explores how the use of the AI tool can be understood in terms of co-creation, focusing on the one hand on how the tool may serve as a resource for understanding and learning, and on the other hand how the tool affects the creative processes. Findings show how the participants search for a 'correct' syntax in their instructions to the machine learning tool, and how the inconsistent and erroneous behavior can work as a way to generate clues and inspiration for generating creative expressions. The notion of friction is used to describe how systems like this can serve to both lower thresholds for programming, and also interfere with the creative processes, encouraging reflection and exploration of alternative solutions.

CCS CONCEPTS

• **Human-centered computing** → *Interaction design process and methods*; **Natural language interfaces**.

C&C' 22, June 20–23, 2022, Venice, Italy

ACM ISBN 978-1-4503-9327-0/22/06.
https://doi.org/10.1145/3527927.3532801

KEYWORDS

GPT-3, generative machine learning, co-creation, programming, post-human design

ACM Reference Format:
Martin Jonsson and Jakob Tholander. 2022. Cracking the code: Co-coding with AI in creative programming education. In *Creativity and Cognition (C&C' 22), June 20–23, 2022, Venice, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3527927.3532801

1 INTRODUCTION

In digitally mediated creative practices, such as interaction design and various forms of digital fabrication, smart or intelligent tools are becoming increasingly common to support the design process. This development aligns with conceptual developments in design research around notions such as post-anthropocentric [13] and post-human design [37], co-performance [20], more-than-human design [17], and machine agency [28] which all suggest a reconsideration of the view that humans should be viewed as the only source of creative agency, to instead see various forms of interactive and digital technologies as having the agency to spur ideas or form a creative expression. With recent developments in AI and generative machine learning [12] these forms of co-creativity become even more topical as several of these technologies are explicitly designed to act as or resemble intelligent forms of co-actors, for instance in producing various forms of fictional text, presentation material, or programming code. In this paper, we present a study of how a system for AI-generated programming code can co-exist with, and be put to use in a higher education programming course for

interaction designers. In the study, the GPT-3 based Codex tool [10] for generating programming code based on natural language descriptions was introduced to a group of students attending a course on creative programming in interaction design. The students were asked to use the tool to work with assignments similar to those that they had worked with previously during the course. They used the tool to produce code that formed creative graphic expressions and animations. The activity thus represents a form of co-creative design activity between participants and the AI-based tool.

The study investigates questions concerned with co-creation in the particular context of learning programming as a tool for creative expression and design. Our findings shed light on a) traditional obstacles of programming such as syntax and coding errors in relation to the AI tool, b) how the participants perceived the relation to the AI tool and its role in designing their creative expressions, and c) how different framings of the AI-tool affect the understanding and interpretations of the system behavior. To frame the challenges and sometimes contradicting ways that the AI system affected the processes of creation and sense-making among the participants, we use and extend the notion of *friction* [11, 21]. This allows us to consider the system both as a tool for learning to program and as a co-participant in a creative process.

2 BACKGROUND

2.1 Generative machine learning

Recent development in AI research targeted at generative machine learning, or Generative Adversarial Networks (GAN) [12] has led to enormous improvements in the ability to generate original computer generated content. GAN-technologies have been used to generate images, audio and text in numerous contexts, for example realistic images of people that do not exist, and images based on textual descriptions. What has been particularly successful is the generation of textual content, primarily in the field of Natural Language Processing (NLP), but also to generate other forms of textual content such as structured datasets, website layouts, database queries, LaTeX equations, and not the least, programming code. Several successful generative machine learning systems for natural language processing (NLP) have been created in the last few years. The study that is presented in this paper makes use of a machine learning model called GPT-3 [7], developed by the company OpenAI. The GPT models are based on unsupervised learning where a very large neural network is trained with massive amounts of unstructured textual data scraped from the internet. GPT-3 has been described as a "few-shot learner" [7], meaning that it is highly versatile and can be adapted to a broad range of different domains, by just providing a few additional training examples that direct the model in a particular direction.

For this project, we are particularly interested in the ability of these systems to interpret and translate natural language into working programming code. This means that you can ask the system using natural language to produce programming code that performs particular functions, such as simple cases of drawing a circle, as well as more advanced programming expressions. This particular functionality is enabled by a variant of GPT-3, that goes by the name Codex [10]. The GPT-3 Codex has been trained on massive amounts of examples of programming code, from large online code repositories like GitHub. The GPT-3 Codex can thus perform tasks such as translating between different programming languages, translating from natural language to almost any programming language, or providing natural language explanations of code snippets. The complexity of the generated code is however limited in how well it translates natural language expressions into working and useful code.

2.2 Co-creation in design and programming

The idea of computers generating content in collaboration with human designers has existed for a long time. The notion of computer-aided design (CAD) and the first CAD-software tools were created already in the 1960s, to support and automate parts of the design process. Albaugh et al. [1] discuss CAD-systems as an example of a set of tools that can be categorized as "time saving systems" aiming at shortening the time or expertise required for each design iteration by delegating repetitive or time-consuming work to the machine, giving the users more space to focus on the creative aspects of design. A development of computer-aided design is the field of *generative design* [27], which often refer to tools or practices where algorithms are used to create design variations or to optimize existing design solutions. This approach is mostly used when working with 3D-models, within for example industrial design and architecture, where the models are modified by changing a set of predefined parameters. A branch of generative design is grammar-based techniques, where complex forms and patterns can be automatically generated from simple specifications [9]. Such an approach was for example used in work by Anderson et al. [4] where generative design was used to let novice users design electronic circuitry. Here the generative design functionality provided alternative suggestions for functional circuits based on high-level descriptions from the users. A majority of this type of computer aids supports design and creativity by lowering thresholds for participation by automating complex tasks, but they have also been used to support the exploration of a design space by providing examples of possible variations to new solutions, that can serve as inspiration for design [25].

Co-creation in relation to programming and programming education is also a fairly well-explored area, even if not often talked about in these terms. There are numerous tools designed to support and simplify various aspects of programming and programming practice. You could also consider the design of the programming languages themselves as creating support for co-creation between human and computer. A fundamental principle in computer programming is the use of abstractions and high-level representations. In high-level programming languages, hardware-specific commands are replaced with textual commands and a syntax that more or less builds on a formalized natural language syntax. There have been several attempts to create support for programming in natural language, for example by Miller already in 1981 [26], and later by Price et al. [30] These early examples however require that that the user adheres to a pre-defined syntax. This differs from the GPT-3 based system used in this paper, which is not restricted to a particular syntax or set of commands. Another strand of research and development concerns computer supported programming, where the computer provides "smart" support to the programming activity, such as finding ways

of providing automated feedback and repairing erroneous code to students learning how to program [39]. Another common form of support that is built into most IDE:s is code completion, where the editor can provide suggestions for how to complete a command or a set of commands, based on the input from the user.

2.3 Post-human perspectives on co-creation

The conceptual underpinnings of this paper rely on theories of the interplay between human and machine with a broad foundation in the research field of Human-Computer interaction, but with a particular focus on post-human theories, and the recent articulation of post-human design [13, 38]. This connects to the broader conceptual conversation in HCI on material and machine agency [3, 5, 17, 28, 33, 35] as being shared or distributed among humans, machines, and artefacts, and the reconsiderations of processes of design and interactivity that these imply. At its core, post-human perspectives on design rejects the traditional dichotomies between humans and machines in favor of a perspective on design in which humans and machines are considered to be co-creators in processes of ideation, design and making. These theories put to the fore how human agency must be understood as entangled with agencies that stem from non-human entities, e.g. as extensively elaborated in notions such as agential realism [5] and machine agency [28]. Of particular relevance for the present work are studies that question traditional categorizations of programming and creativity as relying on step-by-step models as going from design ideas - to digital representations - to machine execution, and instead view humans, materials, and machines as working in conjunct in co-creative and co-performative ways.

We argue that artificial intelligence systems are particularly interesting to explore as a form of non-human co-performer as they display properties that resemble those that would be ascribed to - or expected by - human actors, such as problem-solving and natural language interpretation. An important strand in post-human and post-anthropocentric interaction design has explored notions that challenge common expectations of AI systems to work as rational predictable actors, such as uncertainty [24], imperfection [18, 34, 36], and under-determination [1], and the consequences these have on the way we conceptualize and design tools and methods for these practices.

3 STUDY SETUP

The study was integrated into an ongoing course in Creative Programming for third-semester students in a bachelor's program focusing mainly on user experience and interaction design. The course is a six-week full-time course aiming at teaching the foundations of programming using javascript in a creative context, aiming for the creation of audio-visual and interactive expressions. The course makes use of the programming environment Processing and the javascript library *p5js* which is tailored for creative programming practices and commonly used by artists and other non-professional programmers. The study was presented in the fifth week of the course as a voluntary workshop on "programming with AI". 21 students chose to participate in the study, which was about half the class. The workshop lasted for two hours and consisted of an introduction to the AI tool, followed by independent work with the AI tool to produce code, and finally, a follow-up focus group interview.

The tool that the participants used was the OpenAI "playground" tool that allows for interaction with the GPT3 based models. For the study the Codex (davinci-codex) model was used, which is specifically trained to generate programming code. After being introduced to the tool, the participants were presented with two tasks to work with: First, they were asked to draw an ace of diamonds on the screen - a well-defined and seemingly simple task, that however hides some complexity related to rotating objects. The second task was the main activity in the workshop, and here the participants were asked to create a visually interesting expression - a task that was similar to some of the assignments they worked with previously in the course. To support these tasks, the participants had access to a few "presets" in the playground tool. These presets consist of a few lines of code providing example prompts to the GPT-3 model providing so-called 'few-shot learning' [7] examples, giving hints of the kind of output that are expected. In this case the presets consisted of link to the *p5js* javascript library followed by a simple program that draws a circle and a rectangle on the screen. The presets ended with a natural language description of what kind of expressions the tool should generate. In the first preset this line read: "Replace this line with instructions that makes the AI draw an ace of diamonds", as depicted in Figure 2. After each task, the participants were asked

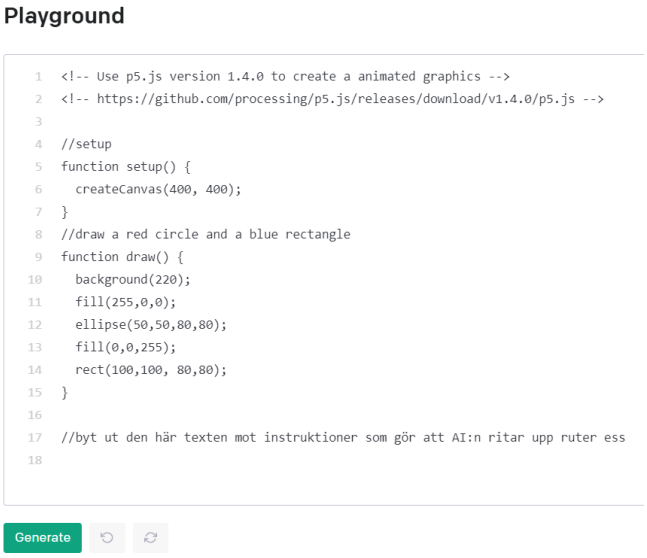

Figure 2: Screen capture of part of the OpenAI playground window (from openai.com) showing the preset text that was used in the first part of the study. The final line in Swedish translates to: "Replace this line with instructions that make the AI draw an ace of diamonds"

to write down their reflections on a number of questions about their experience, and after the final task also provide the final code they ended up with as a representation of "an interesting expression". The session ended with a group interview where the group was divided into two halves and asked to reflect on their experiences

based on some open-ended questions. The data that was collected and analyzed consists of, on the one hand, video recordings of the activities in the workshop and of the concluding group interviews, and on the other hand of the material from 13 projects that were handed in at the end of the workshop, consisting of programming code and written reflections to the questions described above. In the results section below, quotes from these answers refer to the respective projects as *P1-P13*. That the 21 participants resulted in 13 hand-ins was due to that several participants chose to work in pairs, and that a few participants decided to not hand in the material at the end of the workshop. Transcriptions of the data were analyzed using open coding and thematic analysis.

4 RESULTS

Of the 13 projects handed in after the workshop, 12 projects contained working executable code, and are depicted as screenshots in Figure 3. All but one of the artworks (P10) were animations that changed their appearances over time. Three of the artworks (P1, P7, P13) were interactive, reacting to mouse input. Neither the instructions in the assignment nor the preset examples mentioned animations or interactivity as explicit aims of the assignments. Throughout the workshop, the participants worked iteratively by using the Codex system through natural language expressions to generate javascript code representing those expressions, for instance, "draw two red squares in the bottom half of the screen". The Codex system would in most cases output syntactically correct code snippets that adhered to the format that was provided in the preset examples. The participants would then copy the generated code into a separate programming IDE where they could execute and try out the code, and modify it if needed. From the participants' accounts, we saw that manual modification of the code happened to a limited extent, and thus, participants primarily engaged in explorations of various natural language formulations for instructing the Codex system.

4.1 Inconsistencies and errors

Even though the Codex system throughout the workshop almost always provided syntactically correct and executable javascript code, the output commonly did not correspond to the participants' initial expectations or intentions. In many cases, the output was far from what they were aiming for, as illustrated by this quote: *"I only got flawed code. It had a hard time following simple instructions like 'draw 2 triangles that are mirrored on the x axis' sometimes it was stuck with some of the instructions in a test so it wrote 13 copies of the red triangle and it looked like it wanted to continue with that "* (P3). In other cases the output was more in line with what they aimed for, as illustrated here: *"For the most part, the code understood what I wanted it to do, but I also commonly got it half right. An example of this was that I asked for figures that change both color and size. At first I only got figures in different sizes, but after reformulating the sentence, it worked."* (P6)

In general, the participants expressed that they found that the system performed better when asked to draw basic shapes like circles and squares, and less well when asked to draw more complex things. For instance, *"it failed to draw things like bike, or a face, but managed to produce a fair representation of 'a mountain, using lines'*, see Figure 4. The participants commonly talk about the generated output in terms of being "correct" or "incorrect", or making "right" or "wrong" interpretations of their natural language instructions. Typically they would express themselves similar to this example: *"The code that was generated was incorrect since I gave it the instructions to 'draw a red square on 70 degrees' but what came out was a red triangle that wasn't tilted"* (P1). This view of how the Codex system interprets user instructions is based on a traditional algorithmic model where certain instructions correspond to specific reactions by the system. Interestingly, this is quite far from how the Codex system works and not how it was described by the instructor at the beginning of the workshop. Instructions to the system do not require following a specific syntax, but can be of an open-ended natural language character.

In many cases, the Codex system would also provide a different output if run twice with the same instruction. This kind of inconsistencies and unpredictability was often experienced as frustrating and confusing by the participants, but it also led them to explore modes of interaction with the system where they would repeat the instructions until they got a satisfactory output, as illustrated in this quote: *"At first the generator would not give me the right code and forgot to sketch the ellipse. I gave it another try and it worked."* (P9)

These examples of perceived misinterpretations and inconsistent behaviour bring along questions regarding how this kind of system should be framed within the context of programming as a creative practice. Considering the artificial intelligence system as a co-participant in a creative process, it would not be surprising if the same query would render different outputs, just like interactions with other participants would not always unfold in a predictive manner. Concerns about what should be considered a correct or incorrect response thus rely on the point of view that the system is looked upon, as an active contributor to the creative process or as a tool that would be expected to respond in accordance to the users' instructions.

4.2 Instructing the system and expectations of syntax

One of the most commonly recurring accounts from the participants concerned that they assumed that the instructions provided to the Codex system - such as *"draw a yellow circle"* - required a certain 'syntax' in how they would be formulated to achieve the desired output. This involved an assumption that to get the results that you want from a particular query you would need to formulate it syntactically correct, similar to that of a programming language expression. *"The code changes and is extensively affected by small edits in the formulations, so it is very sensitive to errors"* (P5). Consequently, the participants put significant effort into finding what they hoped would be the 'correct' formulation of the query that would lead to the intended outcome. A central aspect of how the participants used the Codex system concerned exploring what would be a working language to use in creating formulations that would lead to the generation of pieces of code that could be useful in designing their expressions. In the interviews with the participants, they said that they tried to adapt their instructions according to what they figured "the AI" would understand. These queries

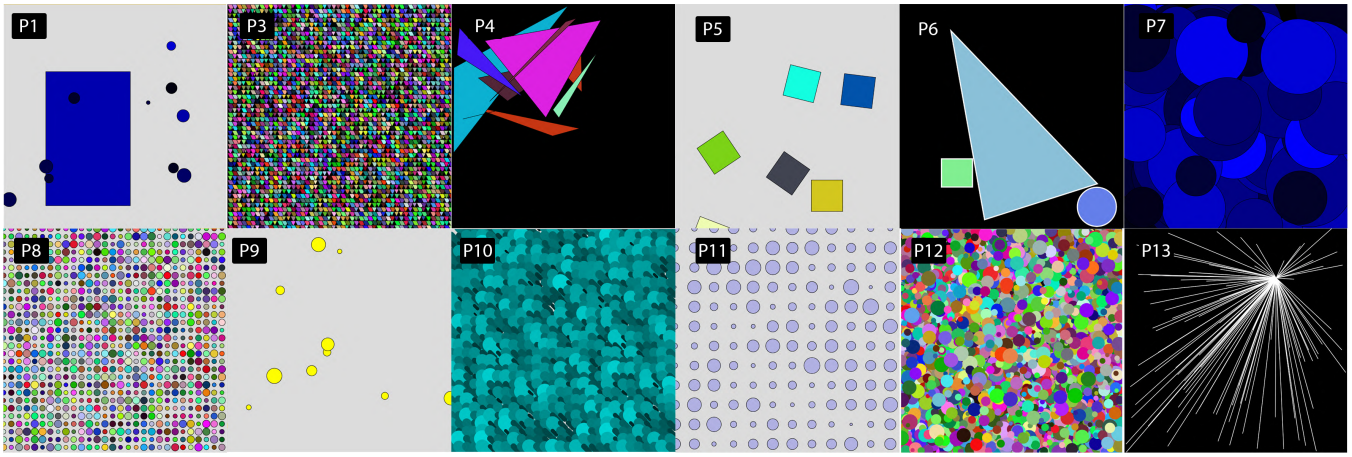

Figure 3: Image artworks generated by the participants in the workshop. A majority of the sketches were animated and some of them were also interactive, reacting to mouse movements. P2 is omitted since the project did not contain any working programming code

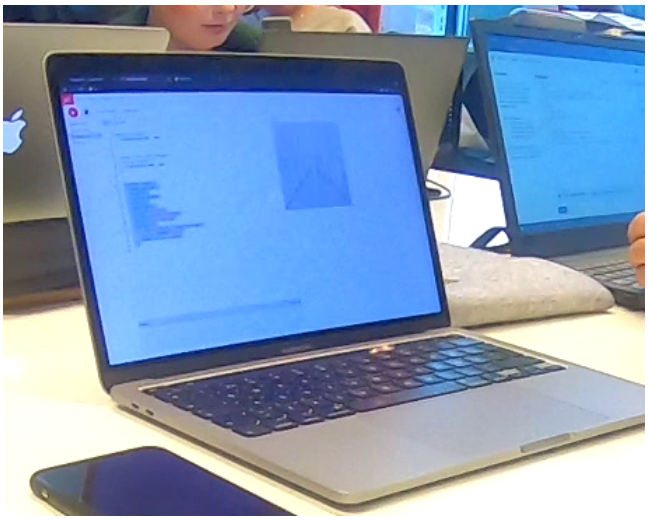

Figure 4: An early attempt at creating an expression by asking the system to create a 'mountain using lines', resulting in a triangular shape resembling a mountain, but also a set of (not asked for) animated vertical lines

ranged from everyday natural language expressions such as *"Could you draw a nice yellow circle"*, to more formal expressions such as *"draw circle yellow"*. These attempts involved making assumptions regarding the boundaries of what they figured that the system could understand and what pieces of code that the system was capable of producing.

These accounts resonate with previous research on interactions with voice-based conversational agents, such as a study by Luger and Sellen [22] where they found that the users of such systems systematically tested appropriate speech syntax in order to 'speak its language' and to create a dynamic mental model of its' capabilities. Additionally, these accounts also mirror Luger's and Sellen's identification that "When learning to use their CA, all participants described making use of a particular *economy of language*", where their participants dropped words other than 'keywords', reducing the number of words used, and searching for the most specific terms. These results are very similar to accounts that commonly appeared in this study, such as: *"Sometimes it requires that you use a simplified language in order for it to understand"* (P2). A quote from another participant highlights expectations that the system should be able to understand and interpret the underlying intention of different sentences with similar meanings: *"I can safely say after testing several phrasings that the result totally depends on how the sentence is formulated and how the computer generates code from that. Several times different sentences but with the same meaning resulted in different outcomes. I think that you have to find "code" words, i.e., the words that the system can understand and write based on that. To identify a pattern"* (P11)

In the post-activity interviews one participant accounted for the struggles to get one particular feature to work by experimenting with different wordings, concluding that *"on the time you spend figuring out what you need to say to it, you can just code it yourself."* These various accounts pinpoint some of the central challenges in the use and design of a system that should work as a co-participant that should render nuances and interpretations to a creative process that goes beyond a mere syntactical reading of an instruction, and at the same time being predictable and to some extent understandable in the output it generates.

4.3 Creative, generative, and conversational use

When looking at what role the Codex system played in the creative and explorative parts of the activity, some interesting perspectives can be identified. Some participants for example point out that the system promotes creativity by taking care of some of the difficulties of programming, leaving more room to focus on the end result, as in the following quote: *"In my case it was an effective tool as I have*

limited programming skills, which in turn limits what I can produce, creatively in this kind of tasks. So therefore it has been an effective way to promote creativity" (P11). A similar quote brings to the fore how the system may be useful to realize creative ideas: *"...when you know in your head how you want to design something but you can't really pinpoint how to write it yourself"* (P11).

While some participants approached the Codex system as a tool to be used to simplify or speed up the coding process, others treated it in a more 'generative' and open-ended fashion. Adopting such a generative approach involved using the system to create pieces of code based on a more loosely defined idea. Such use largely took the form of a *conversational process*, in which the system worked as a co-participant in a collaborative process of developing the idea, rather than as a tool that would serve a more well-defined purpose. In such generative use, the joint actions of both participants and system would constitute the shaping of a shared idea. These conversations involved manipulation and tweaking of the generated code in order to shape a creative expression. This entailed iterative processes of using the Codex system to generate code that was manipulated in subsequent actions of code generation and code manipulation in order to solve the task. This kind of use was mirrored in the participants' reflections: *"Sure, the code was useful in solving the problem on your own, it provided hints as to how the task could be solved."* (P1) or "*when you only have a rough idea of how to design something, but really cannot put the finger on how to write it yourself.*" (P11)

This kind of conversational use was also reflected in how some participants approached the exercise in a modular fashion. Typically, they would create snippets of code that would contribute to solving various parts of the expression they wanted to create. Programming here largely became about experimenting with and manipulating the instructions to the system, and also to some extent tweaking the actual code that was generated. For instance: "*It gave many different idea suggestions and interesting expression that it created on its own based on vague instruction. Lots of inspiring proposals that could be used to to complete/extend/build a foundation for ones own code*" (P7) This experimental and conversational approach could be understood in terms of having a *conversation with the material* [32], where the material can be attributed with certain agencies [14, 35] that shape the interaction.

As has been noted previously, the system often generated seemingly random output, that was not at all in line with the users' intentions, such as in this answer to a question about whether the generated code was correct or flawed: *"Both yes and no, yes because we actually got something working, but no because the expression wasn't what we had in mind. It did not recognize the shapes we wanted to use but generated circles or triangles instead"* (P4). Sometimes this partially random behaviour however led to interesting results, that in some cases were perceived as having qualities that exceeded the intended results: *"I can't say that the final became a unique expression but in an earlier stage when I randomly typed in a sentence that was generated by the AI it became a unique expression with rotating circles that changed color and created a lively pattern which thereafter could have been developed even more"* (P11). In the post-activity interviews the participants noted that the Codex system would *"contribute to a lot more creative solutions through its randomness"*. If we continue the framing of creative design as engaging in conversations with the material, parallels can be made with how Devendorf et al. [14] talk about how non-intuitive technologies and unpredictable and 'stubborn' design materials forces the user to negotiate her goals with the machine. However, some participants also raised the issue that the system merely generated a bunch of *"weird stuff"* without any concrete connection to the desired expression. This points to the necessity of considering the boundaries between experiencing the system as working in an almost random fashion vs seeing it as a co-actor that generates potentially useful ideas.

Another dimension that is commonly put to the fore in research on generative design, concerns aspects of *originality* and *authorship* [25], which here is expressed in terms of to what extent the participants saw the resulting expressions as authored by themselves or by the AI system. Here the opinions differ with respect to whether the participants viewed it as their own expressions, such as in this quote: *"I still think that it became like my own expression since it made everything like I told it. And also since I understand the meaning of the code"* (P8). Or if the AI system is the primary source behind the expression, such as in the following: *"The expression I finally ended up with was to the most part the AI's. I wrote 'fill the canvas with lines' it was the computer who drew the lines in different sizes and placed them in this pattern. I never gave any specifications so this was the computer"* (P10)

4.4 Support for reflection and learning

In the post-activity group interviews, the participants were asked to reflect on how useful tools like this could be for learning computer programming. On this note, one participant argued that the tool helps you not to get stuck in finding the exact programming language syntax, which makes the learning activity more enjoyable. As the system would always provide a potential starting point, it might be easier than to merely start from a blank slate. Others agreed that it is a potentially useful tool for novice learners, as you can have an experimental approach where you generate code that you then can modify and change on your own. As noted by one of the participants in the concluding interviews: *"It provided clues, it does not always give perfect code, but still some kind of working suggestions. It provides clues on how to structure things and what one would need to learn more of."*. Another participant compared working with the tool to how you work with pseudo-code in ordinary programming education, but where the tool allows you to quickly generate working code from the pseudo code, and that going back and forth between these is beneficial to learn the programming code syntax. A common view among the participants was however that the system's inability to correctly interpret the user's intentions was not beneficial to the learning situation. One participant framed it as for it to work in this context *"...it has to be flawless so that it won't generate code that does not contribute to what you try to achieve otherwise there is a risk that you learn it wrong."* And another participant put it like: *"It spits out so much and so random things... and as a beginner you think that these things are related to what you wrote"*

Despite the participants' view that the unreliable behavior of the system is problematic in educational settings, we could also see that these inconsistencies and flawed behaviour afforded a potential for reflection and learning. During the workshop session, there were

many occasions where the participants struggled with not getting the result they wanted, and where they had to closely investigate the generated code to see why it did not work, and how to potentially change it. We saw several cases in which participants scrutinized the generated code and said that it acted weirdly, since according to their interpretation it looked like it should. For instance, two participants tried to rotate a square 45 degrees by using the function "rotate(45)", but the code did not execute as expected. This confused them as they were quite convinced how "rotate" should work. They continued by experimenting with different numbers based on the errors of their initial trials, to figure out the correct usage of this function.

One could see this as encouraging a form of *reflection in action* [32], by creating an explorative situation of learning-by-doing and tinkering, [31] where activities of code generation is intertwined with activities of interpretation and meaning making. As noted by several participants in the interviews, for this kind of interpretative activities to become meaningful, a basic understanding of programming is required: *"I think that you have to know some programming in advance, so that you know that here it made a mistake, and correct it and take the things that are correct and use that"*. Ideally, the generated code should be a bit more advanced than the current level of the student, (or in the *zone of proximal development* [15]), but in the generative AI system used in this study, the level of complexity of the generated code is difficult to control. As noted by one participant: *"Sometimes it gets really complex with nested loops with many lines of code"*.

A final reflection with respect to learning concerns what the introduction of these kinds of tools does for the motivation to learn computer programming. As one participant puts it in the interview: *"If it generates the code for you, you are not motivated to learn how to do it yourself"*. Given the rapid development of AI in general, and generative machine learning in particular, it is not unrealistic to assume that the performance of these types of systems will increase. Here, one could argue that this paves the way for entirely new types of programming practices based on natural language interactions. So it might be a fair question to ask from a student's perspective, why do you need to know a particular complex programming syntax, when you can just tell the AI system what to create.

5 DISCUSSION AND CONCLUSIONS

The findings from the study show that there are conflicting views a mong the participants on how to understand and use a system such as this in the context of programming education, as well as on the different ways it may aid the practice of computer programming in a creative context. Traditionally, computer programming has been framed as a form of problem-solving practice that resembles scientific thinking, but more recently creative [6], embodied [19], and craft-oriented [8] perspectives of programming have contributed to a shift in this view. These bring up questions of how the practice of programming should be understood in an interaction design context and how a system that "co-creates" with the users should be used and framed. Clearly, if the responses of the Codex system are considered from the point of view of 'providing the code that the participants asked for', there are a considerable number of flaws and inconsistencies in how it works. However, as reflected in several accounts of the participants and as shown in previous work on post-human design, uncertainty, inconsistencies, or 'randomness', (e.g.,[24, 34]), the responses of the Codex system contributed to opening up for creativity and novel design ideas. Thus, when viewing this form of interaction as a co-creative or co-performative activity, the outcome cannot be determined based on a literal interpretation of how the participants initially expressed their ideas, but as a shared expression that unfolds in the interaction between user and system. In the following sections, we will discuss the findings in relation to three broader themes relating to co-creation with artificial intelligence: 1) learning to 'speak' the language to creatively use the system, 2) co-creation as a process of reducing or inducing friction, and 3) managing user expectations by considering how the system is framed within the creative process.

5.1 Cracking the code

Our findings point in several directions with respect to how participants perceive the usefulness and the value of this system. In many of the accounts from the participants, they claim that the success of getting their intended response from the system depends on their ability to formulate sentences that the AI understands, thereby suggesting that there is some kind of syntax that need to be figured out as one becomes more experienced in using the system. These accounts reflect research on voice-based conversational agents such as Alexa and Siri, showing how users tend to adapt and simplify their language in a way that is assumed that the system understands more easily. These studies highlight how natural language interfaces do not communicate their capabilities in the same way as traditional graphical user interfaces. Instead, users bring expectations of using natural language from other settings, most commonly from interaction with other natural language speakers, and adapt their interpretation of the system's response based on those [22, 29]. As noted above, while many participants became frustrated that the system did not generate code that worked in the fashion that they had foreseen, they also expressed value in the open-ended character of the interaction with the system. The open-endedness and unpredictability were thus interpreted positively as well as negatively. However, even if there is a value in that a system is not fully predictable - it might still be problematic when users expect that they have to figure out particular syntax when interacting with the system, when, in reality, there is none. As one participant said, *"it's like a mysterious system where you have to 'crack the code' in order to unlock the full potential of the system"*.

This particular mindset and understanding of the system can be attributed to that machine learning systems like the GPT-3 system used in this study can be described as 'black-box' systems, where it is inherently challenging to understand why the system behaves the way it does. Contemporary guidelines for the design of AI systems [2] generally strive for increasing the transparency of the system by providing explaining what the system can do, how well it does it, and why it did what it did. In the context of co-creation and creative design it is however not obvious that it would be more beneficial with a completely transparent system where capabilities and means of communication are clearly articulated. The unpredictive and inconsistent behaviour instead points to that the behaviour of the

system could be understood in terms of *animism*, as articulated by Marenko and van Allen [24]. In this context, animism should not be confused with anthropomorphism, as it is not a question of suggesting human-like capabilities and modes of communication. Instead, animism refers to a form of interaction that fosters the unexpected instead of prediction and linearity, and where conversations with things, rather than about or to things, take place. So learning to 'talk to the AI' from an animistic perspective is neither about conforming to a particular syntax, nor expecting a communication that resembles that of another human. To 'crack the code', becomes a matter of learning to treat the interactions with the system as contributions to a joint project of co-creation between human and machine, appreciating what can be experienced as unpredictability and inconsistency, rather than about unlocking what the machine is exactly capable of.

5.2 Reducing or inducing friction

As noted by Albaugh et al [1], systems aiming for co-creation with computers in creative contexts can be roughly divided into "time-saving systems", and "time-deepening systems", where the latter aims at disrupting or dehabituating an otherwise familiar practice, and supporting creative reflection. One way of rephrasing this is that the system may either reduce or introduce *friction* into the activity." Friction has previously been used in interaction design research for example by Laschke and Hassenzahl [21], where the notion of friction is described as a design quality that can be used to nudge users towards meaning-making and reflective use. Similarly, Cox et al. [11], talk about friction in terms of microboundaries that create obstacles, fostering a more mindful use of technology. Our findings show how the Codex system to some extent served both to reduce friction, by e.g. providing programming code with accurate syntax that they could use as a starting point to explore an idea, but also to induce friction, by providing code that performed differently than the users expected it to, thereby challenging them to expand, refine or rethink their idea.

Our findings show numerous accounts of how the Codex system may fill an important role in reducing friction in creative programming practices. Many participants highlighted that it was useful to get a 'starting point' through an automatically generated piece of code that executes and is syntactically correct. Some participants point out that this could be particularly useful for inexperienced programmers. To reduce friction, it is important that the system responds in a consistent and predictable way, and that it actually produces code in line with the users' expectations. From this point of view, the partially unpredictable behaviour of the system does not serve to lower friction, but was sometimes experienced to include flaws that would need to be mitigated by clarifying the system's behaviour.

However, our findings also show how the creative processes might also benefit from introducing friction into the process. This becomes particularly relevant in cases where the designer has a less well-defined or even vague idea of what the end result should be. In such a case, the system would instead play the role of co-exploring the design space and providing novel directions and resources in a creative process. Our findings provide several examples where the participants initially aimed for a particular expression, but where the system did not render exactly what they had hoped for. Instead, the system generated code with unexpected elements, a form of friction, that forced the users to reflect on and reconsider their actions and intent. In many cases, this steered the design exploration in new and often more creative directions. In the context of creative design, this reflects work by e.g. [23], [16] and [1], which have identified how aspects such as ambiguity, inconsistency, uncertainty, and randomness, may work as resources in a creative process.

These ways of experiencing the Codex system represent conflicting views on how creative work can be supported, either by reducing or inducing friction. Understanding friction from these views requires that designers find ways to balance aspects of automation with aspects of interference in the creative process. In some cases, it is preferable for the system to more closely align with the user's literal articulations, whereas in other cases it might be better if the system interferes or diverges from them in order to expand or open up a design space. We argue that the concept of friction can capture both these dimensions of using and designing these kinds of systems and thus become a potentially useful concept in the context of co-creation with artificial intelligence.

5.3 Framing the system - tool or co-creator

In line with post-human design perspectives, introducing systems of this kind in creative practices such as interaction design and programming involves repositioning the relations between humans and machines in the creative process. In particular, as opposed to viewing design and creativity with such tools as uni-directional processes governed solely by the actions and intent of the human, this requires that we consider the use of this kind of system as engaging humans and machines in co-creative dialogues. As our findings show, in a co-creative practice, we cannot assume that a system such as the one used here generates well-defined ideas or solutions, but rather these systems should be understood as creatively contributing to the unfolding of the design process. By considering creative practice from this point of view, we argue that we contribute by challenging the idea of the designer as the sole source of the ideas and intents that form a particular creative expression and thereby, expanding the space for the creative potential of this kind of tools. By *framing* the activities with this kind of tool in different ways we can engage users in attributing various degrees of agency to the AI tool [20]. From one point of view, the system resembles a passive tool that just acts on the command from the user. If the user asks it to draw a circle, it should draw a circle. Sometimes the system will understand the commands correctly and produce a correct response, and sometimes it will fail to do so. From another point of view, the system can be viewed as a co-performer or co-agent, which aligns with how GPT-3 systems actually work, referring to what can be labeled 'few-shot learning' systems [7], to which input is provided, and then the system autonomously generates novel output that is coherent with the provided input. From such a perspective, the issue is not to consider the behaviour of the system as correct or incorrect, but rather an issue of to what extent the system deviates from the original examples. Understanding the interaction with these can be described through Kuijer's and Giaccardi's notion of co-performance [20] to conceptualise the extent to which artefacts are capable of performing and exerting agency *together* with people. In contrast to notions of good or bad performance of a technological artifact, this shifts focus to the contextually situated

'appropriateness' of the joint human and artificial performances. In the context of this study, framing the interaction as a joint performance, shifts focus from how well the system output aligns with the user intention, to instead emphasize the resulting animated expressions as a joint achievement between user and system.

5.4 Conclusions

In this paper, we have shown how the sometimes inconsistent and imperfect behaviour of an AI-based system for generating programming code provided various challenges for the participants engaged in a programming exercise. However, it also worked as a creative resource and as a means to encourage reflection on the solutions that they were developing. To frame the challenges that a system such as this might introduce, we use and extend the existing notion of *friction*, as a way to address the need to balance the qualities of automation with that of interfering in the creative process. We believe that finding ways of dealing with these two aspects is crucial to successful use of generative AI for co-creation in creative settings. We stress the importance of *framing* when presenting activities to users, that involve generative AI for co-creation. By designing settings of use that involve a post-human design perspective, the users' understanding of what can be expected from it can be addressed, thereby putting generativity and unpredictability at the core, rather than a well-defined query-response system. Finally, our findings point to how in interaction with black-box systems such as the Codex, experimentation with language and syntax becomes a core part of the activity. In reflecting on the activity, participants displayed an understanding of the use of the Codex system as a matter of figuring out how to talk to the system to get the response they looked for. This suggests that introducing generative AI support for computer programming based on natural language interaction, although it might relieve the programmers from some of the struggles related to the syntax of the programming language, also introduces new struggles with identifying or inventing an effective 'syntax' in the interaction with the AI-system itself.

REFERENCES

[1] Lea Albaugh, Scott E. Hudson, Lining Yao, and Laura Devendorf. 2020. Investigating Underdetermination Through Interactive Computational Handweaving. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference (DIS '20)*. Association for Computing Machinery, New York, NY, USA, 1033–1046. https://doi.org/10.1145/3357236.3395538

[2] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300233

[3] Kristina Andersen, Ron Wakkary, Laura Devendorf, and Alex McLean. 2019. Digital crafts-machine-ship: creative collaborations with machines. *interactions* 27, 1 (Dec. 2019), 30–35. https://doi.org/10.1145/3373644

[4] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. Association for Computing Machinery, New York, NY, USA, 331–342. https://doi.org/10.1145/3126594.3126637

[5] Karen Barad. 2007. *Meeting the Universe Halfway: Quantum Physics and the Entanglement of Matter and Meaning*. Duke University Press. Google-Books-ID: H41WUfTU2CMC.

[6] Ilias Bergstrom and R. Beau Lotto. 2015. Code Bending: A New Creative Coding Practice. *Leonardo* 48, 1 (Feb. 2015), 25–31. https://doi.org/10.1162/LEON_a_00934

[7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]* (July 2020). http://arxiv.org/abs/2005.14165 arXiv: 2005.14165.

[8] Leah Buechley, Mike Eisenberg, Jaime Catchen, and Ali Crockett. 2008. The LilyPad Arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. Association for Computing Machinery, New York, NY, USA, 423–432. https://doi.org/10.1145/1357054.1357123

[9] Scott C. Chase. 2005. Generative design tools for novice designers: Issues for selection. *Automation in Construction* 14, 6 (Dec. 2005), 689–698. https://doi.org/10.1016/j.autcon.2004.12.004

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *arXiv:2107.03374 [cs]* (July 2021). http://arxiv.org/abs/2107.03374 arXiv: 2107.03374.

[11] Anna L. Cox, Sandy J.J. Gould, Marta E. Cecchinato, Ioanna Iacovides, and Ian Renfree. 2016. Design Frictions for Mindful Interactions: The Case for Microboundaries. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*. Association for Computing Machinery, New York, NY, USA, 1389–1397. https://doi.org/10.1145/2851581.2892410

[12] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. 2018. Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine* 35, 1 (Jan. 2018), 53–65. https://doi.org/10.1109/MSP.2017.2765202 Conference Name: IEEE Signal Processing Magazine.

[13] Laura Devendorf, Abigail De Kosnik, Kate Mattingly, and Kimiko Ryokai. 2016. Probing the Potential of Post-Anthropocentric 3D Printing. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS '16)*. Association for Computing Machinery, New York, NY, USA, 170–181. https://doi.org/10.1145/2901790.2901879

[14] Laura Devendorf and Kimiko Ryokai. 2015. Being the Machine: Reconfiguring Agency and Control in Hybrid Fabrication. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 2477–2486. https://doi.org/10.1145/2702123.2702547

[15] Tayebeh Fani and Farid Ghaemi. 2011. Implications of Vygotsky's Zone of Proximal Development (ZPD) in Teacher Education: ZPTD and Self-scaffolding. *Procedia - Social and Behavioral Sciences* 29 (Jan. 2011), 1549–1554. https://doi.org/10.1016/j.sbspro.2011.11.396

[16] William W. Gaver, Jacob Beaver, and Steve Benford. 2003. Ambiguity as a resource for design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03)*. Association for Computing Machinery, New York, NY, USA, 233–240. https://doi.org/10.1145/642611.642653

[17] Elisa Giaccardi and Johan Redström. 2020. Technology and More-Than-Human Design. *Design Issues* 36, 4 (Sept. 2020), 33–44. https://doi.org/10.1162/desi_a_00612

[18] Miwa Ikemiya and Daniela K. Rosner. 2014. Broken probes: toward the design of worn media. *Pers Ubiquit Comput* 18, 3 (March 2014), 671–683. https://doi.org/10.1007/s00779-013-0690-y

[19] Martin Jonsson, Jakob Tholander, and Ylva Fernaeus. 2009. Setting the stage – Embodied and spatial dimensions in emerging programming practices. *Interacting with Computers* 21, 1-2 (Jan. 2009), 117–124. https://doi.org/10.1016/j.intcom.2008.10.004

[20] Lenneke Kuijer and Elisa Giaccardi. 2018. 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*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173699

[21] Matthias Laschke, Sarah Diefenbach, and Marc Hassenzahl. 2015. "Annoying, but in a Nice Way":. 9, 2 (2015), 12.

[22] Ewa Luger and Abigail Sellen. 2016. "Like Having a Really Bad PA": The Gulf between User Expectation and Experience of Conversational Agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 5286–5297. https://doi.org/10.1145/2858036.2858288

[23] Betti Marenko. 2015. When making becomes divination: Uncertainty and contingency in computational glitch-events. *Design Studies* 41 (Nov. 2015), 110–125. https://doi.org/10.1016/j.destud.2015.08.004

[24] Betti Marenko and Philip van Allen. 2016. Animistic design: how to reimagine digital interaction between the human and the nonhuman. *Digital Creativity* 27, 1 (Jan. 2016), 52–70. https://doi.org/10.1080/14626268.2016.1145127 Publisher: Routledge _eprint: https://doi.org/10.1080/14626268.2016.1145127.

[25] Jon McCormack, Alan Dorin, and Troy Innocent. 2004. Generative Design: A Paradigm for Design Research. *DRS Biennial Conference Series* (Nov. 2004). https://dl.designresearchsociety.org/drs-conference-papers/drs2004/researchpapers/171

[26] L. A. Miller. 1981. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal* 20, 2 (1981), 184–215. https://doi.org/10.1147/sj.202.0184 Conference Name: IBM Systems Journal.

[27] James Mountstephens and Jason Teo. 2020. Progress and Challenges in Generative Product Design: A Review of Systems. *Computers* 9, 4 (Dec. 2020), 80. https://doi.org/10.3390/computers9040080 Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[28] J. Brian Pickering, Vegard Engen, and Paul Walland. 2017. The Interplay Between Human and Machine Agency. In *Human-Computer Interaction. User Interface Design, Development and Multimodality (Lecture Notes in Computer Science)*, Masaaki Kurosu (Ed.). Springer International Publishing, Cham, 47–59. https://doi.org/10.1007/978-3-319-58071-5_4

[29] Martin Porcheron, Joel E. Fischer, Moira McGregor, Barry Brown, Ewa Luger, Heloisa Candello, and Kenton O'Hara. 2017. Talking with Conversational Agents in Collaborative Action. In *Companion of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. ACM, Portland Oregon USA, 431–436. https://doi.org/10.1145/3022198.3022666

[30] David Price, Ellen Rilofff, Joseph Zachary, and Brandon Harvey. 2000. NaturalJava: a natural language interface for programming in Java. In *Proceedings of the 5th international conference on Intelligent user interfaces (IUI '00)*. Association for Computing Machinery, New York, NY, USA, 207–211. https://doi.org/10.1145/325737.325845

[31] Mitchel Resnick and Eric Rosenbaum. 2013. Designing for tinkerability. In *Design, make, play: Growing the next generation of STEM innovators*, M Honey and D Kanter (Eds.). Routledge, New York, NY, USA, 163–181.

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

[33] Lucy Suchman. 2017. Agencies in Technology Design: Feminist Reconfigurations*. In *Machine Ethics and Robot Ethics*. Routledge. Num Pages: 15.

[34] Jakob Tholander and Maria Normark. 2020. Crafting Personal Information - Resistance, Imperfection, and Self-Creation in Bullet Journaling. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376410

[35] Jakob Tholander, Maria Normark, and Chiara Rossitto. 2012. Understanding agency in interaction design materials. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2499–2508. https://doi.org/10.1145/2207676.2208417

[36] Vasiliki Tsaknaki and Ylva Fernaeus. 2016. Expanding on Wabi-Sabi as a Design Resource in HCI. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 5970–5983. https://doi.org/10.1145/2858036.2858459

[37] Ron Wakkary. 2020. A Posthuman Theory for Knowing Design. 14, 3 (2020), 12.

[38] Ron Wakkary. 2021. *Things We Could Design: For More Than Human-Centered Worlds*. MIT Press. Google-Books-ID: UiY6EAAAQBAJ.

[39] Ke Wang, Rishabh Singh, and Zhendong Su. 2018. Search, align, and repair: data-driven feedback generation for introductory programming exercises. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2018)*. Association for Computing Machinery, New York, NY, USA, 481–495. https://doi.org/10.1145/3192366.3192384