Five Futures with AI Coding Agents

Steven L. Tanimoto
tanimoto@uw.edu
Paul G. Allen School of Computer Science and Engineering
Seattle, Washington, USA

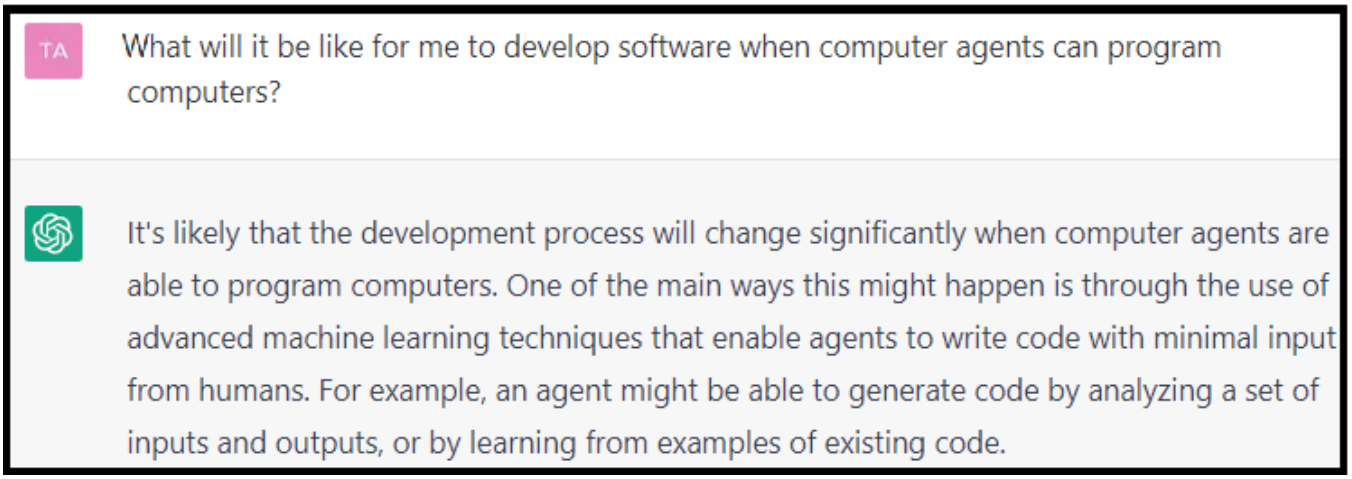

Figure 1: How will software development change?

ABSTRACT

Many computer programmers are beginning to use computational agents to help them develop software. This article raises questions about the nature of programmer-to-agent relationships. The author's intent is to foster thought that will help human programmers best prepare for such relationships and perhaps design the relationships, ultimately keeping their jobs and improving their programming experience.

CCS CONCEPTS

• **Software and its engineering**; • **Social and professional topics** → *Computational thinking*; **Computing profession**; *Intellectual property*; • **Computing methodologies** → *Artificial intelligence*; • **Human-centered computing** → *Collaborative and social computing*;

KEYWORDS

programming experience, artificial intelligence, coding agents, Copilot, ChatGPT, liveness, flow, future, software development, live coding, code golf, trust, verification, pair programming, interactive intelligent development environment, coding service, collaboration, soloing

<Programming>'23 Companion, March 13–17, 2023, Tokyo, Japan

ACM ISBN 979-8-4007-0755-1/23/03.
https://doi.org/10.1145/3594671.3594685

ACM Reference Format:
Steven L. Tanimoto. 2023. Five Futures with AI Coding Agents. In *Companion Proceedings of the 7th International Conference on the Art, Science, and Engineering of Programming (<Programming>'23 Companion), March 13–17, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3594671.3594685

1 INTRODUCTION

Since the arrival of Copilot[4] in 2021 and ChatGPT in 2022, impressive demonstrations of agents writing computer source code to a user's specifications have been viewed by a wider and wider audience. These demos beg the question of whether agents will replace human programmers any time soon. This paper does not attempt to answer that question, at least not directly. Rather, it considers another one: how will the existence of agents like these and ones even more powerful than these change the experiences of programming from what they are today?

This paper approaches the question by considering the basic nature of possible relationships between a human programmer and a computer agent. These particular kinds of potential relationships are then described in terms of characteristics commonly discussed in PX/nn workshops, such as flow and liveness.

1.1 Criteria for Comparisons

In any possible relationship between a computer agent and a human programmer, there are some basic questions that we can ask. The answers to these can help to map out a sort of landscape of the possible future experiences.

Balance of Work: How much of the programming process will be done by the human, and how much by the agent(s)?

Balance of Expertise: What knowledge and skills will be required of and exercised by each participant, human and computer?

Initiative: Which – human or agent – will typically be the initiator of sub-projects? Always the human, always the agent, or will there be a kind of mixed-initiative character to the collaborations?

Affect: Is it likely that the human will enjoy, appreciate, or look forward to participation in the collaboration – or not?

Power: Will the human or the agent have the upper hand in making design or other project decisions when they disagree?

Viability: Does some particular human-agent relationship seem to make sense in a particular context, such as the economics of software product development, the effective learning of computational thinking, or the exploration of a scientific hypothesis? This criterion is a meta-criterion, as it doesn't pertain directly to the nature of the human-agent relationship but rather to a judgment about whether a given relationship seems actually plausible, realistic, or defensible.

1.2 Disclaimer

This paper is primarily the result of some brainstorming. It's neither theoretical nor empirical. In a sense, it is science fiction, anticipating possible futures based on the premise that intelligent agents are becoming more capable and more ubiquitous in the experiences of human computer programmers. An in-depth survey of how programmers have already been using automated services and large language models in software development was recently published by Sarkar et al[10]. The present paper attempts to draw a somewhat wider circle around possible future styles of programming in a world with agents, and it does not present a wide review, but it is intended to stimulate discussion.

1.3 Spoiler

Here is the conclusion of the paper, if one can call it a conclusion at all. In the end, we can only conclude that there will be changes to the experiences of many programmers, and that we are likely to see quite a variety of evolutionary threads in experiences of programming.

2 SOLOING

Soloing is about human programmers acting alone and intentionally ignoring or rejecting the kinds of advanced tools and agents that might get in the way of free will, ownership, credit, or that might otherwise cramp the style of a human bent on being a purist.

There are cultures of programming today in which programmers accept tight constraints such as maximum length of source code, amount of memory used at runtime, or (commonly) maximum coding time. The practice of "code golfing" has a following in which programmers try to minimize the length of their source code, or try to fit the most sophisticated code into, say, a 140-character (early-format) Twitter message[3].

The metaphor implicit in "code golf" is that programming can be a sort of sport. As such, the activity is shielded from the market forces on the general profession of programming. When the automobile replaced horse-drawn carriages and riders on horseback as predominant means of transportation, the sports of horseback riding, jumping, racing, and polo did not disappear. In a way, they became more exotic and perhaps their participants are regarded today with more awe than in the past. Similarly, the future of soloing may become intensified in communities that react to the presence of coding agents by doubling down on their human abilities and their pride in exercising and displaying them.

In hackathons[6], programmers are typically given between one and two days to start and complete a programming project. Thus a 48-hour time limit is a good example of a tight time constraint for the coding. In programming contests, such as the International Collegiate Programming Contest, students must adhere to strict rules on what tools are to be used, as well as time limits[7].

In a typical livecoding performance[1], a human programmer sits on a stage in front of an audience and uses a laptop computer connected to a data projector and an audio amplification system and edits source code that is run continuously (even as the source code changes) to generate synthesized music. The audience experiences both the music and the display of the source code as it is edited. As an art form, livecoding is thus at least somewhat protected from agent-caused elimination.

For programmers who are practicing for or participating in these types of activities, software-creation agents such as Copilot may be irrelevant. Of course it is also possible that new games and contests arise in which such agents are permitted to be used as tools or as part of a controlled programming environment for the human participants. If agents are used as partners or teammates in these constrained activities, then "soloing" would not be the best term for such a future human role.

There is another argument and context for the persistence of soloing. We humans have a right and a need to express ourselves with languages, including with code. Programming is a means not only of producing software or of doing artistic or musical performances, but also of testing and realizing ideas, crafting expressions that have meaning and perhaps elegance, and of providing an alternative means of communication that may best suit any particular person whether by reason of ability, disability, convenience, or personal preference. Most programming languages such as Python, for example, have been designed for human coders, not for AI agents. (An exception might be Postscript, which although accessible to many human programmers, was primarily intended for word-processing back-end processors to create instructions for laser-printer engines.) As a legitimate form of human communication (e.g., storytelling by coding or live-coding animations), it seems inhumane to suggest that people shouldn't code anymore in an age of coding agents. Just because agents such as Alexa and Siri can talk, we don't expect that human speaking will go out of fashion.

We humans arguably have a right and at times, a need to express ourselves with code. The soloing modality is the dominant one for programmers today. The future for the existence of solo human programming looks good, and soloing in various forms is likely to persist regardless of how powerful AI agents get.

In terms of the criteria listed earlier, the balance of work, knowledge, initiative, and power are all on the side of the human in this

future. Presumably, affect is at an extreme because live coding (for example) is so much fun. If we quantify most fun as 0 and most annoying as, say, 10 on this same scale, then we can imagine the soloing future as being located at the origin of a space of futures spanned by the ranges of values for the answers to each of the criteria.

3 "ON THE SHOULDERS OF AGENTS" OR *FLOW*

I expect that most programmers will want to leverage the agents. Considering the agents to be tools will allow the programmer's IDE to be not just an interactive development environment but an interactive *intelligent* development environment. The IDE (or IIDE) will continue to be driven by the human programmer. However, the nature of services provided by the environment will be at a higher level than, say, what we have today in terms of code completion and code refactoring.

Programmers will use the new intelligent tools to take their creativity and productivity to the next level. Achieving the cognitive experience of "flow"[5] at a new, higher level of creative activity is a hope that today's programmers can adopt and aspire to achieve. Whereas flow for a programmer today may manifest itself in activities of writing assignment statements, fixing bugs due to uninitialized object references, and off-by-one loop index errors, flow for a future programmer might involve steps at the level of integrating a new code library, trying out an alternative user interface to product in beta, or inventing and coding up a new matchmaking algorithm in a supply-demand application scenario. What takes days today might take only minutes a couple of years from now.

Personally, I would like my programming environment to be "live" at level six[11]. At this level, the environment can anticipate my strategic programming goals, inferring my likely intentions as I work, offering me working components that I might need (either existing, newly customized or synthesized for the occasion) and then integrate them into my project as I give the go-ahead. I expect that such agents would help me both in achieving high-level goals and a personal state of cognitive flow.

Thus, the new agents might enable programmers to achieve flow at a highly creative functionality level rather than just at an implementational level. If the resulting flow enhances the humans' creativity, then there is hope for our society's future as a whole, if one believes David Brooks, columnist for *The Atlantic* [2].

> If a society is good at unlocking creativity, at nurturing the abilities of its people, then its ills can be surmounted.

If there are many agents helping out, the programmer's role could be that of a director of development. Like the conductor of an orchestra, who does not bow the violins or blow the horns, the director would not enter the assignment statements and type the semicolons but would use design terminology, gestures, and new protocols to control how code is built by agents to achieve the overall goal.

Where in the space of futures is this one? It could be almost anywhere. It's not at the origin, because then the agents wouldn't be helping out at all, and our human would be only achieving flow, if at all, at the conventional level. It's important for the human's affect to be near zero – more fun and less frustration. It seems important for the human to have decision-making power most of the time, because being consistently overridden would probably weigh heavily on the human's motivation and affect and contribute towards breaking or preventing flow.

4 COASTING OR "LET THEM DO IT"

When programmers let agents do most of today's work of software development, they might be "coasting" and thereby having an easier time of the job than they would have had before. Each of the phases of software development might or might not be performed by agents: requirements gathering and analysis, system design, detailed specification, selection of software frameworks and existing components, coding of new functionality, debugging, testing, performance analysis, documentation, and gathering beta-test feedback, release, post-release maintenance, etc. Whatever agents cannot do effectively or more economically than humans, will be left to the humans. However, at first glance, it looks like the humans will have a lot of help, and that they might therefore have an easier time.

4.1 They'll Do the Work, But then What?

Yet of course, when the agents get that good, the humans will be asked to up their game or do new tasks. What might these be, in addition to filling the gaps left by the agents? One answer is that the humans could "go meta" and become responsible for software development projects at a more abstract level, perhaps acting as managers, but performing tasks such as the following:

- Articulate the need for a product.
- Articulate the need to invest in the product.
- Integrate the product/project within the ongoing business of the organization in terms of balancing budgets and achieving consistency of corporate policy or image.
- Shop for the latest and greatest new agents and software engineering services.

So the "coasting" is on the programming side. The human just gets a new job to do – being a meta-developer. Matt Welsh argues that this will be the most likely future for programmers within 3-5 years[12].

4.2 Do You Trust Them?

A big question today for programmers getting help from unknown contributors and agents is how much to trust these "helpers" and the content they provide. Agents such as ChatGPT have often answered questions with convincing-sounding, but incorrect answers. If tomorrow's agents continue to be unreliable, a lot of work should go into methods (new or dressed-up though old) of detecting problems, validating code, analyzing provenance of code and gauging levels of trust in the agents themselves.

The existence of untrusted agents providing software is not new. The open-source community has been dealing with it from its early days. (The agents in this case have been human contributors.) Yet, the prospect of dealing with huge quantities of new auto-generated code from agents based on machine-learned language models calls for more research and development into validation and code-fixing

tools that can keep up with the onslaught. (Sarkar et al discuss this in sections 5 and 7.2 of their survey[10].)

Perhaps there will be future coding agents that always produce clean code, free of many kinds of problems such as syntax errors, security holes, malware, and dense or overly complex expressions and structures. Perhaps the trust issue will go away then. Then human programmers might be able to coast a little longer, if they don't want to be just testers and fixers.

5 "TEACH ME"

Human coders might need to and/or like to learn from AI agents. The premise for this future is that a human-agent relationship can be based on learning activities in which the agent is the source or purveyor of the knowledge or skills being transferred. Two key aspects of such a relationship are the form of interaction and the subject matter being taught. The interaction may be modeled after existing tutee-tutor structures in pedagogy, or perhaps some new structure involving new settings (e.g., virtual reality, or code visualization environments), communication modalities (new languages, metalanguages, types of diagrams, speech-text-effects mixes, etc.). The content being taught might be from the domain of software engineering and programming languages, or it might be lessons about how the AI works and performs its role in facilitating software development. Or perhaps the content is irrelevant to software engineering and is communicated solely to provide a break from the main slog of getting real work done or to build a form of camaraderie between human and agent. Of course a relationship such as this need not exclude other modes of interaction between the human and the agent. However, there seems to be a paradigm here in which a programmer's experience is enhanced by interaction with an agent.

Where is this future in the abstract space mentioned earlier? It could be in a lot of places. Today, even human tutors use alternative methodologies in tutoring their pupils: the initiative within a session maybe mostly the tutor's or mostly the tutee's. The knowledge balance is presumably that the tutor knows more than the tutee in the subject being taught, but it could be that the tutor knows a little more, a lot more, or actually a lot less, except in the very specific topic area for the session. Perhaps depending on the manners of the tutor and the degree to which is considerate towards the tutee, the tutee's experience may be positive or negative. Affect for the human thus is likely to depend on a combination of design features of the agent as well as the personality and individual needs of the human.

6 "TEACH THEM" OR HELP THEM SAVE YOUR WORLD

Our world has a lot of big problems, such as a climate that is getting less hospitable to human beings, growing pollution of the air, water and land with health-threatening substances, and the strong possibility of catastrophic wars. What will it take to solve these problems (if they are solvable)? If they are not really solvable, what are the best policies for reducing the risks or mitigating their effects?

For a problem such as fixing the climate, when one takes into account many of the relevant factors, the problem is so complex in terms of information, prediction, and identifying best possible decisions at all levels, that it would seem crazy not to be considering getting help from those powerful information-processing engines called computers. Then which computers? What software? Where do we start?

One answer is to find some good AI agents that can not only help keep track of all the relevant information, but integrate it into models that can lead to decision-making that consistently optimizes the prospects for humanity. Do such agents exist? Maybe not, or at least not yet. This is where human programmers might come in and teach some existing agents to become better at studying possible solutions to "wicked problems"[9]. Human programmers could help agents with both implementational techniques (e.g., how to integrate certain types of data into a general database) and strategic methods (ways to organize the overall structure of a possible solution, in terms of major enabling steps, policy components, and clear vision). Programmers would not be the only humans involved, of course. Human experts in science, engineering, policy, psychology, and other fields, no doubt, would be needed, especially if the agents we are talking about are like today's ChatGPT — good at chatting in a natural-seeming way, but unable to synthesize solutions to messy problems.

To refuse the help of advanced information technology in working to save the world does not sound like a road to success. Computers are needed. Of course, we'll need to be able to trust this technology or find ways to turn its recommended policies into safe and likely effective ones, but that is part of helping them help us save the world.

7 COMPARING THE FUTURES

Figure 2 shows a map intended to suggest one view of the space of possible futures. The map is based on two of the criteria mentioned in section 1: (on the horizontal axis) balance of work and (on the vertical axis) balance of power. These two criteria appear to be relatively independent of each other, although likely correlated with the others.

Figure 3 provides a matrix with a cell for each (relationship-attribute) pair. In each cell, a code is provided as a possible characterization of the relationship (or placeholder for such) in terms of that attribute. While there could be a discussion for each entry of the matrix and/or a discussion of each future compared with each other, let us use the matrix to focus here on two particular comparisons: (a) Flow vs. Coasting, and (b) Teach Me vs. Teach Them. One can argue that since Coasting may be an inherently unstable relationship, with the human vulnerable to being let go, we should, if we wish to protect humans from that fate, attempt to influence designs of programming tools and practices to favor the achievement of Flow. This should involve a reasonable balance of work between human and agent, a balance of expertise, and either a balance of initiative or a weighting of the initiative towards the human. Combining those with the human having the power to decide contentious issues, we can expect a positive affect for the human. Achieving flow will not only require these aspects but basic usability and reliability of the tools or services provided by the agent. The survey by Sarkar et al[10] covers studies of programmers using intelligent tools, and these lessons learned should influence the design of IIDEs.

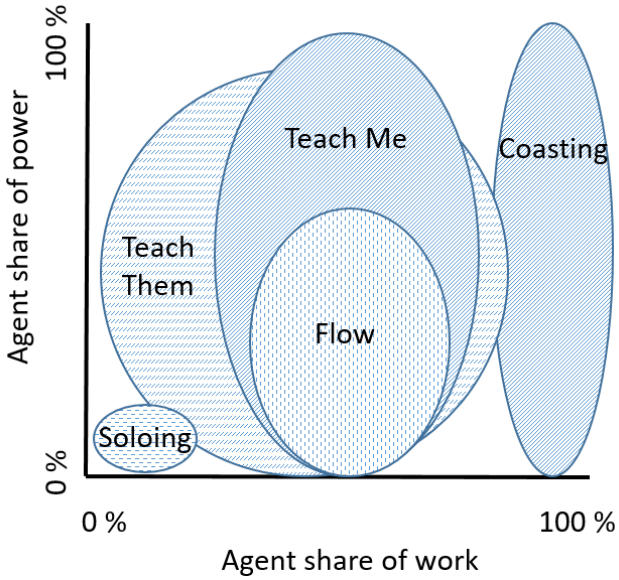

Figure 2: A map of the space spanned by two of the criteria in the introduction, showing the rough regions in which each of the five futures lie.

The Teach-Me future and Teach-Them future each involve the basic teacher-student or tutor-tutee relationship. The human programmer takes on the role of tutee in the first future, and the role of the tutor in the second. For a positive affect in the Teach-Me future, the human must be motivated to learn, and then satisfied by the learning that takes place. In the Teach-Them future, the human must be motivated to teach and then satisfied that the computer agent has learned the intended lessons. In the former case, the programmer may wish to be taught about an algorithm proposed by either party in the session, or about some other aspect of the software under construction or being considered for inclusion. The material to be learned should normally be relevant to the project being tackled. If irrelevant material is taught, it should be either to help keep the human-agent relationship effective, or serve as part of a break from the more demanding work of learning tough stuff that IS relevant. A human programmer might be motivated to pass on knowledge to an agent for the same reason an elder is motivated to teach younger people or to write books for them: to pass knowledge on to succeeding generations of people. The agent learning here is a proxy or a vehicle for the transmission of knowledge. A human might also be motivated to teach something to an agent if the agent might then be likely to be able to help with something important. Over time, we might expect agents to learn so much that the opportunities for humans to teach them things they don't already know may become rare. That may be true, but it probably will take some time to settle out. Even after that, the humans may wish to teach agents about their own preferences and personalities, so that the partnerships can be optimized.

The work in a teaching/learning relationship should be shared. Learning is easier when material is presented at an appropriate level and when questions can be well answered. Actual teaching is impossible if the learner fails to engage. On the other hand, balance of expertise, initiative, and power may vary in these relationships without necessarily killing them. The Teach-Me future seems comparatively viable; humans will need the ability to seek help within an IIDE, and the more complex the material, the more important it is to have an intelligent tutor, rather than simply a help-search mechanism. The viability of the Teach-Them future can be questioned, particularly if agents are primarily designed to perform specific services rather than learn new ones from their human IIDE partners. On the other hand, there is only a fuzzy boundary between instructing a system (i.e., providing a specification of behavior) and teaching it a skill or piece of knowledge. A feature of the latter is the likely persistence of the learned content across projects, i.e., its generality. Past research on intelligent tutoring systems can inform the design of IIDEs that support these teaching-and-learning futures. The systems could include learner models and advanced pedagogical capabilities[8].

Each of the many D entries in the matrix of Fig. 3 indicates not only that the rating of a future relationship on the particular axis "depends" on details of the relationship, but that an IIDE system essentially has a design variable there. Whether or not the agent will typically have most of the object-oriented programming expertise in a partnership setting could specified as Yes when the system is designed. Since there are many kinds of expertise, this would better be specified with a vector or a richer means.

8 ANTICIPATING THE FUTURE

8.1 What's Coming

The five futures suggested here are of course simply *some* of the foreseeable types of scenarios. In actuality, not only is it likely that human-agent relationships will differ in various ways from these, but will also be dynamic – changing in character from one session to another and even within a single work session. The agents may come and go such that there could be many agents in a session at a time, and some agents could play multiple roles at a time, acting as helpers, tutors, bosses, confidants, and secretaries. Humans may also be collaborating actively in teams with other humans, as well as with the agents. Thus the pair-programming model for which Copilot was designed is not the only human/agent structure we're likely to see.

8.2 Preparing to Live with Agents: Understand, Cope, Adapt, and Exploit

A final question that should be asked, in the context of anticipated changes in programming experiences is this: how should students and current software professionals rethink their plans for learning and working in programming and software development? There are at least two kinds of answers: (a) try to anticipate the schedule of new agent capabilities and services coming out and be ready to do other parts of the software-engineering work at the appropriate times, and (b) try to learn enough about agent technology itself to be able to ride the waves of change by contributing, mediating, anticipating, or knowing when to get out of the way.

Here is the author's suggestion to all people who rely on their own creativity in their lives, professionally or personally, including

Criteria for Comparisons

Five Futures	Balance of Work	Balance of Expertise	Balance of Initiative	Affect	Balance of Power	Viable?
Soloing	H	H	H	P	H	Y
Flow	B	B	H/B	P	H	Y
Coasting	C	B	B	D	D	D
Teach Me	B	D	D	D	D	Y
Teach Them	B	D	D	D	D	D

```
Key:
        H -- Human
        C -- Computer agent
        B -- Both
        D -- it Depends (on the design and the usage conditions)
        P -- Positive
        Y -- Yes
```

Figure 3: A matrix to help in considering how each of the five futures stacks up in terms of the six evaluation criteria.

computer programmers: first understand, then cope; next, adapt, and finally, exploit.

A. Understand: What are large language models? They are large information structures constructed through automated training using information from the Internet. They are particularly well adapted to predicting the natural flow of words in text and following associations among words. How are they operationalized? These models can be used to generate new text of several forms: poems, essays, computer source code, and others. They can also work with image synthesis methods such as Stable Diffusion to create intelligible renderings of particular situations or objects described in text – either real or imaginary. What are obvious applications? DALL.E 2 is being used to create graphic art, greeting cards, illustrations, cartoons, and more. ChatGPT is being used to correct ESL students' writing, compose essays, write or rewrite computer source code, and generate fiction. Many users are experimenting with new applications.

B. Cope Most people will not be directly affected immediately. But in the longer term, it's inevitable that AI-generated content of many kinds will be ubiquitous in the lives of all of us in modern civilization. We have already been flooded by bot-created news (and fake-news) stories, advertisements and political messaging. As it becomes more in-our-face, and as material becomes more honestly portrayed as artificial, we must learn to deal with for whatever it is, and it will be many things, including those already mentioned, such as news stories. How should we cope? For one thing, we should all hone our skills at estimating provenance. Is that a fact? Where did it come from? This starts with thinking critically, learning to seek out trustworthy information sources, and being sensitive to warning signs (see, e.g., Sam Woolley's The Reality Game) But not all AI will be bad. We can learn to enjoy the good stuff – appreciate the humor that occasionally comes from AI systems (see, e.g., Witscript by Joe Toplyn). Appreciate a beautiful art composition or poem, whether the source is artificial or human. In reality, most material likely to come out of bots is both artificial and human, in the sense that the large language models have been trained on human-created text and natural images taken by human photographers or art painted by human artists. Having been output as pixels or
synthesized through Stable Diffusion does not mean that there is no humanity in the resulting creation. It means that a sort of shuffling and remixing has taken place. Appreciate that you are living in an era of fundamental technological changes, analogous to the dawn of industrialization, photography, the advent of electrical power, flush toilets and modern plumbing, the automobile, the mobile smartphone, air travel, and antibiotic medicine. You'll be able to say, "I was there."

C. Adapt If you are a professional creative – one who relies every day on your creative abilities in order to make a living – it is possible that your profession will evolve. If you see this happening in your future, you may be able anticipate it and either go with the flow, or stay ahead of the crowd. When photography was invented, artists adjusted their activities to focus less on realism and more on interpretation, whether through impressionism, abstraction, or surrealism. Photography itself developed not only as a recording medium, but as an art form in its own right. As writing, translation, paraphrasing and poetry become increasingly automated, professional writers and editors may become "bosses to bots" who instruct them on what to write, how to tailor material, and what to re-write when it does not come out well enough the first time.

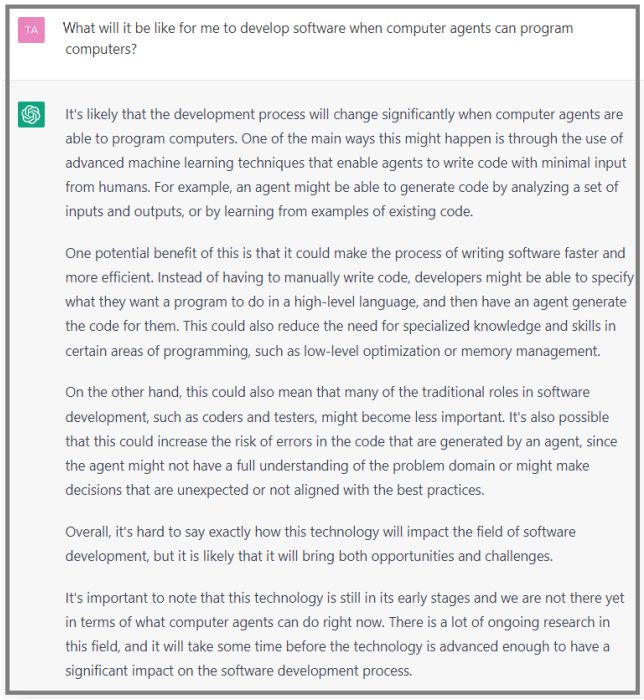

Figure 4: Full response by ChatGPT to a question about the future of software development.

D. Exploit Photography changed what painters did, but it opened up a field and a new profession: photographer. Technologies such as ChatGPT will change what writers do. Journalists are likely to spend more of their efforts on investigation and acquiring stories, and less time on wordsmithing the reports on those stories. A mystery writer may give increased attention to plot features and less to the word-by-word narrative. Completely new media may come out of the new AI technologies, including producing new interactive experiences in which a mix of human and robotic actors create dramatic environments in which playgoers can wander and talk. Educational materials authors will have new powerful tools with which to create rich, customized tutorials in STEAM subjects (science, technology, engineering, arts, and mathematics). Whether you are or are not a professional who is called upon daily to use your creative facilities to do your job, you will have fantastic new tools available with which to play. Create your first adventure novel. Produce a beautiful animated movie to share with others or to serve as a background for your own private meditation. Let AI help you visualize and share your thoughts, dreams, and aspirations.

8.3 Closing Words

For the record, the full answer of ChatGPT to the prompt shown in Figure 1 is provided in Figure 4.

In the PX community at least, these are exciting times.

ACKNOWLEDGMENTS

Thanks go to Edward Misback for providing comments on a pre-submission draft, and to the three anonymous reviewers who provided helpful suggestions. Thanks also go to the organizers of PX/23 and the active audience of participants.

REFERENCES

[1] Alan Blackwell, Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson. 2022. *Live Coding: A User's Manual.* MIT Press, Cambridge MA.
[2] David Brooks. 2023. Despite Everything You Think You Know, America Is on the Right Track. *The Atlantic* (January 2023).
[3] Wikipedia 2023. *Code Golf.* Wikipedia. Retrieved January 15, 2023 from https://en.wikipedia.org/wiki/Code_golf
[4] GitHub 2021. *Introducing GitHub Copilot: your AI pair programmer.* GitHub. Retrieved June 29, 2021 from https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/
[5] Mihaly Csikszentmihalyi. 1990. *Flow: The Psychology of Optimal Experience.* Harper, New York.
[6] Wikipedia 2023. *Hackathon.* Wikipedia. Retrieved January 15, 2023 from https://en.wikipedia.org/wiki/Hackathon
[7] ICPC 2022. *International Collegiate Programming Contest Rules.* ICPC. Retrieved January 15, 2023 from https://icpc.global/worldfinals/rules#heading-8
[8] J. Paladines and J. Ramirez. 2020. A Systematic Literature Review of Intelligent Tutoring Systems With Dialogue in Natural Language. *IEEE Access* 8 (2020), 164246–164267. https://doi.org/10.1109/ACCESS.2020.3021383
[9] Horst Rittel and Melvin Webber. 1973. Dilemmas in a General Theory of Planning. *Policy Sciences* 4 (1973), 155–169. Issue 2.
[10] Advait Sarkar, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? *Proceedings of the 33rd Annual Conference of the Psychology of Programming Interest Group (PPIG 2022)* (2022).
[11] Steven L. Tanimoto. 2013. A Perspective on the Evolution of Live Programming. In *Proceedings of the First International Workshop on Live Programming (LIVE).* IEEE Computer Society, San Francisco, CA.
[12] Matt Welsh. 2023. The End of Programming. *Commun. ACM* 66 (January 2023), 34–35. Issue 1. https://doi.org/10.1145/3570220