# Purpose, Transparency and Challenge: Self-taught programming in adolescence

**Jan Dittrich**
University of Siegen
jan.dittrich@uni-siegen.de

**Felienne Hermans**
Vrije Universiteit Amsterdam
f.f.j.hermans@vu.nl

## Abstract

Many programmers will agree that programming is a unique activity, which, although it has been compared to writing, gardening and painting, is unlike other things humans do. But if it is unlike other activities, how do people learn to be a programmer and what affected them in this process?

We start at the beginning of people's programming careers, when they first start to program and interview 10 developers about their experiences. Based on the interviews, we suggest three core themes that affected the programmer's early learning experiences: Purpose of activity, transparency of technology and meaningful challenges.

## 1. Introduction

A large part of programmers in the workforce today learned programming in a *'self-taught'* way, having started to program at a young age before the internet was widespread. Learning programming not in school, but with help of instructional media is still common: The 2024 StackOverflow survey allowed to report several ways of learning programming. Over 82% learn from "other online resources" while the percentage of people having learned programming in school is about as high as the use of physical media like books.[1] The use of physical media is even higher for programmers between 35-44(59%) and between 45-54(68%), indicating that older programmers might be even more likely to be self-taught.

We are interested to learn more about the phenomenon of self-teaching as it seems to be an unexplored topic in programming education. We wonder: How did childhood experiences of programming come to be? What made it possible for teenagers to start their professional development at such a young age? And how do the views of self-taught programmers shape their theories and practices today in their careers in technology? That is what this paper sets out to explore by conducting semi-structured interviews with 10 self-taught programmers who learned programming in their teenage years.

Using thematic analysis, we find three main themes in our interview data. Participants seem to have been driven by a clear **Purpose** in programming; each had a self-selected goal in mind that they wanted to reach; often making a game, sometimes making a website. This purpose was not only technical but also embedded in a community; peers or parents enabled the love for programming, in material ways, with access to computers or computer books and magazines, or immaterial by providing encouragement and support. Many participants also mention being bored, having nothing else to do, which might have contributed to the need of finding a purpose.

A second theme which enabled programming in childhoods is **Transparency**: in the eighties, programming was much more easy to observe than it is now; BASIC games shipped with their source, and in the nineties, web pages' markup was often easy to read and to understand. We recognize that in today's digital world, the feeling that programming is a thing you can learn, that the codes that they show will be one day understandable to you, might seem much further away than 3 or 4 decades ago, despite an abundance of learning materials.

The final theme is the **Challenge** of doing something difficult. Participants recognized that programming would be challenging, like a puzzle or like running a marathon, and they specifically sought out that challenge. Our participants saw that trying, and then succeeding in something hard, was an interesting

[1] https://survey.stackoverflow.co/2024/developer-profile#2-learning-to-code

reward in itself. Important part of this challenge was that is was self-imposed, sought out and not forced by someone else.

## 2. Background and Related Work

A seminal work on the role of computers and programming in childhood is Sherry Turkle's *The Second Self* (Turkle, 2005), discussing computers and programming as a tool as well as a way to think about questions of life, death and what it means to think. Seymour Papert's *Mindstorms* (Papert, 1980) was published five years prior and details how and why children should learn to program using the language LOGO which allows to draw by code. A similar focus on drawing was discussed by Goldberg by using smalltalk (Goldberg & Kay, 1977). This tradition of using smalltalk in programming education continues until today with smalltalk-inspired languages like Scratch. Notable about the culture surrounding these tools and their makers is a focus on child centered education, outside of traditional classroom teaching. Papert writes in his book "Mindstorms": "I believe that working with differentials [a gear train used in cars] did more for my mathematical development than anything I was taught in elementary school." Teachers thus only have a small role to play in his educational vision. As such, our participants learning outside of school fits the viewpoints of the day.

We are, however, less interested in the tools used for learning, these have been studied extensively in its earlier days (Solomon et al., 2020; Battista & Clements, 1986), see (Lockwood & Mooney, 2018) for a comprehensive overview of integration of Computational Thinking into K-12 education. In addition to programming in schools, we have also explored how programming in taught in out of school clubs in prior work (Aivaloglou & Hermans, 2019).

In this work, however, we want to explore learning of programming entirely outside of educational contexts involving teachers. We aim to find out more about what enabled learning as children for people who are adults today, how context, practices and values enabled their learning, and how that learning shaped their ideas about programming.

The fact that all our participants are use he/him pronouns is indicative of the time period that we study. Particularly relevant for the timeframe in which our participants learned programming is also the history of early home computers and their user communities. Here, the masculinization of computing is also discussed, for example in the marketing of the popular C64 home computer (Juul, 2024) and the shift towards gaming becoming more excluding towards girls and women (Kirkpatrick, 2017). Factors that lead to more participation of women in computer science programs was later researched by Fisher and Margolis (Margolis & Fisher, 2001).

## 3. Methods

To explore the experiences of childhood programming, we interviewed 10 people that taught themselves programming, which we recruited through (then) Twitter. We conducted semi-structured, online interviews with all participants. We subsequently analyzed their answers using thematic analysis (Braun & Clarke, 2019), to explore our overarching research theme of self-directed programming education in teenage years.

### 3.1. Interview Protocol

In each interview, we asked these questions:

1. Can you retell how you learned programming?
2. What is being self-taught?
3. Do you consider to still be a self-teacher, if you learn new things now?
4. Is there a difference between self-taught people and not self-taught people?
5. What is the effect on you now?

| ID | Age | Occupation | Pronouns | Residence | Origin |
|---|---|---|---|---|---|
| P1 | - | - | He/Him | - | - |
| P2 | - | - | He/Him | - | - |
| P3 | 50-59 | Freelance programmer | He/Him | Belgium | Belgium |
| P4 | 30-39 | Freelance programmer | He/Him | Germany | Argentina |
| P5 | 50-59 | Freelance programmer | He/Him | UK | UK |
| P6 | 30-39 | Researcher | He/Him | UK | UK |
| P7 | 40-49 | IT consultant/software engineer | He/Him | Netherlands | Netherlands |
| P8 | 40-49 | Software Developer | He/Him | Netherlands | Netherlands |
| P9 | 30-37 | Dev-Ops/System Engineer | He/Him | Netherlands | Netherlands |
| P10 | 20-29 | Software developer | He/Him | Czech republic | Czech republic |

*Table 1 – Overview of personal details of the 10 participants*

6. What is needed/required for self-teaching?

With Questions 1 to 3, we specifically explored the devices and tools that our participants used, whether their direct community (parents, family members, friends) provided help, or acted as role models, and how other media sources like books, magazines or video tutorials contributed to learning. Question 4 sought to understand how participants see *self-taughtness*, and how they imagined others to see it. With Question 5, we aimed to explore how their childhood experiences shaped their current thinking in their profession (which are often in IT or in adjacent occupations). It also sought to encourage participants to reflect on their experiences as related to peers with a different pathway into computing. Questions 6 was deliberately broad and open to allow referring to all people, media, free time and personal traits.

### 3.2. Data Analysis

All 10 interviews were automatically transcribed by the video conferencing software ZOOM, which we used to conduct the interviews. We also recorded audio, and used the audio where needed to correct and extend the automatic transcripts.[2]

We analyzed the improved transcripts by using thematic analysis (Braun & Clarke, 2019), identifying themes for short sections of each interview which we then discussed. Each of the two authors initially coded five interviews alone (P6 to P10), after which we compared notes and grouped the themes for those first five interviews into categories. From there we individually coded the remaining five interviews, but in the same room, so we could continuously communicate about the found themes. Through iterative refinement of the categories for the remaining interviews, we identified core themes that could be found across several participants.

### 3.3. Participants

Our participants all use he/him pronouns, come from different countries, but mainly from the EU, and hold a variety of jobs related to programming. Table 1 gives an overview of the participants and their backgrounds.

## 4. Results

The goal of this paper is to understand what constitutes being self-taught, and what enabled this between roughly 1980 and 1995. From the interviews we conducted, we constructed three large themes: Purpose, Transparency and Challenge.

Note that additions to quotes between round brackets are context added by the authors. In some cases we have also shortened quotes, indicated by …, and removed words like ‘ehm’. Our analysis resulted three main themes that we elaborate upon in the next sections.

[2] which was very often needed, the term ‘self-taught’ does not seem to be included in many datasets the ZOOM speech to text algorithm is trained on, and the non-native English of both interviewees and interviewers often tripped up the tool, too

## 4.1. Purpose

Participants all spoke about finding a *purpose* in programming, a thing that they felt both drawn to from within, and enabled to do by outside factors.

### 4.1.1. A thing to build: Games and Websites

Having something concrete in mind to (re)create contributed to participants having a purpose, and for most of our participants, what they wanted to create were games.

Rather than serving a singular simple aim that could be summarized of "*having fun while playing a game*", games also served as motivation for programming and as a topic to bond over with peers.

Our participants pointed out that using the computer for games was obvious, and further elaborated: "*This was what these little home computers were for, nobody did anything serious with them*" (P5). Similarly, P4 pointed out that the activities on the computer were "*playing games or doing programming*". The choice between these activities was even seen to be baked in the hardware: "*I mean, unless you shove a cartridge in there with some cool Konami game, you immediately get a prompt for programming or loading stuff from disk or tape[...]*" (P7)

Thus, it is not surprising that programming was often approached through wanting to create games: "*...of course you have to try to create games.*" (P3). P4 initially wondered "*How do people make these games?*" and P6 asked his father "*How do I make a computer game?*" continuing that "*... his answer was basically to give me a book from his bookshelf on BASIC.*". Two participants, P6 and P10 also mentioned the customization of games.

Games were not just a means for interaction with the computer alone. P8 says his MSX microcomputer club "*was more a games exchange.*" and P7 told that one purpose for exchanging with peers was to get new games: "*Basically, you bought a box of floppy disks and then you went to a friend that you copied whatever new stuff he had lying around.*". Thus, games also serve as means to build social ties.

Not all participants were interested in building games though. Younger participants like P2 also mentioned making websites, but in all cases the participants had a thing in mind they wanted to build.

### 4.1.2. Boredom

However, just the goals of making a game or a website might not have been enough to start programming. Games and websites might have kept children busy *playing* the games or *visiting* the websites rather than creating them.

Participants frequently described that interacting with computers was a welcome way to spend free time, and that at one point, all levels of a given game would have been completed.

P4 told us: "*So at night I didn't have anything to do, because I lived also far away from my friends*" and "*There was nothing more to do; either I was doing playing games or doing programming.*"

These feelings are very much in line with the experience of the first and second author of this paper. P9 contributed to this topic as well: "*I mean back ... we didn't have an Internet or anything, so there wasn't a whole lot to do.*" P10 says his self teaching was enabled by "*Various kinds of discoveries . . . And a lot of free time*".

P6 also described that, as a young teen, in those days, there weren't so many things you could do: "*You know with that age, what activities were available back then? If you're home at, let's say like 10 o'clock at night, and you're not out with your friends, then what have you got? Maybe like TV or reading a book. Or playing games with my brother.*"

P9 stressed that, because the computer was so limited, making games was needed: "*And mostly, I pretty quickly got bored with what the machine could do . . . So I pretty quickly figured out that there had to be a way to make machine do more ... so I took it from there and I started in a QBasic*" and added "*I want to emphasize the boredom part . . . That's the main motivator for almost anybody*".

Programming was the only way to create new games or programs when all the games one owned were

played, and there was no distraction from Netflix or TikTok.

It is interesting to think about how learning this way is difficult in today's world where kids never reach 'the end of content'. P9 explicitly reflected on this topic: "*That which is increasingly absent in this modern world is boredom*".

#### 4.1.3. Social Permission

A final part of purpose is that people around you allow, or even encourage you to do an activity: Participants talked about being part of a community, that both helped them to get started and to keep going. In other words, they experienced what we call 'social permission' to pursue their interests.

P1 said that, in addition to access to book and materials, access to "*people that will actually answer questions that you run into*" is most important for self-directed learning. He further mentioned that it is not only learning from others, it is also sharing the love for programming, stating he thinks that "*the availability of people actually excited for this work*" is what gets kids going. P3 confirmed this, says that "*friends at school were the main source of information.*"

Magazines played a role in creating small communities of peers. P5 remarked "*And we did read magazines. A lot of the magazines were actually really brilliant.*" P8 also mentioned magazines: "*MSX magazines that also had listings, which my neighbor also liked, so he typed in the listing, and I tried- I solved the bugs in those programs.*" Here, the media provides the code that he and his friend then collaborated on. P9 told that he started to learn alone, but "*As time went on, I got to know some other kids who were a couple years older, who were more experienced with programming and they basically taught me some other stuff.*" and P7 remarked that "*. . . there was classmate who's dad did some LOGO. . . And maybe we also had a machine running LOGO in school.*"

There were also mentions of working alone at least for some time: P9 said he did not have support, contrary to other people: "*I noticed some friends of mine had parents who would guide them . . . That's what I missed. I basically had to rely on my own curiosity.*" P4 also described programming alone for a long time, and longing to find a group to learn together with: "*I could not find in my class somebody that that I could ask. . . because I was the one who knew about programming.*", eventually finding peers in his university career: "*Finally. . . I found the place where where I was among people at my level, so that I could improve.*" P6 told us: "*I don't think I knew anyone else that was doing it in primary school*", but that he found peers with similar interest in secondary school.

Most interaction with peers are about programming and computers, but descriptions about working together on code are rare. The direct interaction with code itself is mostly a solitary activity.

#### 4.1.4. Parents

It was not just peers who made our participants feel that programming was a good activity to pick up. Parental support has played large role in enabling programming enthusiasm. In almost all interviews parents, most often fathers, play a large role in shaping the interest of participants in interacting with computers.

There were repeating patterns of how children got access to computers and how their parents interacted with their kids. First of all, there was direct support received from parents in the form of buying the computer needed for programming, which at the time were quite rare. Sometimes the computer was brought into the house for different reasons, for example for a parent's jobs (P6, P7, P9, P10), but often they were bought specifically for children to use or even to program on (P1, P3, P5). P5 expresses that the computers were often bought for children directly: "*I can't think of anyone whose computer was their dad's or their mom's. It was a child's thing.*" In addition to the computer itself, parents also provided educational materials like books or magazines.

Parents were usually computer users themselves. They did not teach or encourage programming directly, but they often had positive feelings about technology and computers. The situation seems to have been positioned at participants at a sweet spot between not being forced to do programming by overexcited

parents, but also not being discouraged by parents who were very critical about computer use.

For example, P4 described his father working for the government overseeing "*digitizing of processes, and he may have seen the benefits of that … But I think he saw this [digitization] coming and that is was important.*" P1 told about modifying an existing English language program to work in Dutch saying "*my dad was very concerned as well as proud.*" P2 stated: "*My dad though, he was more interested in technology [than P2's mom], he is an electrician*", P10 describes his father similarly: "*His work was technical, but he was not dealing with software, he was a hardware technician*".

P6 describes "*at one point I went to my dad and I said: "How do I make a computer game?" and his his answer was basically to give me a book from his bookshelf on BASIC.*" He goes on to describe how his father had bought the book for himself but never really explored it in depth.

## 4.2. Transparency

Many participants in our study indicated they started programming with a wish to recreate something they knew. Older participants (P6, P4, P3) mention playing games and wanting to build those, slightly younger participants like P2 mentioned making websites.

It seems to have been important to have access to the code behind running programs, and to change that code and learn from it, in two ways. Firstly, seeing a completed game or website clearly showed what learning to program could bring, helping to keep enthusiasm going ("It must be possible to do this"). Secondly, having access to the source code made it possible to get started by adapting programs rather than building new ones from scratch, a sort of Use-Modify-Create model *avant la lettre* (Franklin et al., 2020).

For most of our participants, 'succeeding at programming' did not mean reaching a mathematical definition of a solved problem—something like correctly reversing a list or finding a certain substring in a piece of text—but being able to successfully figure out a way of replicating parts of another program, like a realistically bouncing ball (P1) or rounded corners of resizable boxes on websites (P2). After that success, some participants would tinker with the solution to make it more uniquely theirs. This is a recognizable experience for the second author, who while in high school had together with a friend decided their goal was to program a rotating cube in BASIC. The goals was self-selected, and there was no clear ceiling, when the rotation was done, we added the challenge to make it interactive, to add texture and so forth. It was not finished, in our minds, when we graduated high school.

## 4.3. Challenge

The final large theme that we suggest based on the interviews is enjoying a challenge. Of course, this relates also to purpose, as doing a challenging thing is much more likely to give a sustained purpose than something that can be completed quickly and without effort.

### 4.3.1. Programming as puzzle solving

There can be many different reasons to be excited about learning programming: It can be driven by a concrete goal, like making a game or a website, but we also found that many of our participants were, at least in part, driven by learning something complex because it was intellectually rewarding.

P1 makes this distinction very concrete: "*I don't think I wanted to create as much as I wanted to understand what was going on. So it was this great, great puzzle*".

The reward was in the programming itself and not other factors: "*Even though it was challenging, it was a lot of fun, figuring things out on your own, when you succeed*"(P2). The accompanying frustration was seen as enjoyable, P2 continues: "*Most people could probably run a marathon just by being stubborn enough. But most people don't do that, because they know it's a lot of pain. They kinda need to enjoy that pain, I suppose*". P9 expresses a strong focus on solving problems: "*You really want to understand why there is a problem ... It's almost like a sort of Pavlovian reaction, like: computer gives problem, I need to fix it*".

Consequently, getting and using a permanent result was not relevant. P5 describes that he and his friends

wrote games "*all the time*" while also saying that "*they were rubbish.*". That the games worked meant that the puzzle was solved: "*You play it for five minutes and then you... then the next night you do another one*" (P5). P4 similarly describes that "*It [building games] was just for me, it was much- interesting to make the games than to play the games.*". P10 mentions that he liked programming for its own sake, and not for external reasons: "*being able to program as a kid wasn't something that that gave me my social credit, which I think has changed in recent years*".

#### 4.3.2. Exploration

Participants mentioned learning through exploration. Often, our participants find that this is not just the way they learned, but a requirement for being a programmer. P1 explained "*I do consider to [figuring things out] be a prerequisite for being a good programmer. Because if you can figure things out, you can program.*"

P3 stated that what drove his interest is "*Interesting things and explore those*", which meant "*Lots of experimentation and not following rules but experimenting*". This is tied to the feeling of hacking or breaking things, which can be attractive. P4: "*...we will basically try to break Windows [The Operating System], putting websites and folders in the desktop. ... So that's basically like like mashing stuff until it worked.*"

Learning by exploration is related to tinkerability and we doubted, whether we would put it together in one category. Ultimately, we decided to describe them as two separate aspects: Tinkerable artifacts are compatible with most school systems; even if you would have a teacher-led form of teaching, a teacher could show students, step by step, how to recreate games or websites and then have kids pick a program to replicate, adapt and change it.

Exploration, however is not so much about what you do as an activity, but about how you do it: From your own motivation, by finding just the right amount of information and then going back to work. This form of learning aligns more with Papert's way of constructionist teaching in which kids build something that has meaning to them by figuring out how to do that. This model of learning is not compatible with explanation by a teacher. After all, Papert said that "*by explaining something [to someone] you take away the ability for a child to discover it*" (Papert, 1980).

#### 4.3.3. Obsession

Some of our participants named being 'obsessed' with the computer and not doing any other things, which fits modern day stereotypes about people interested in programming (Lewis, Anderson, & Yasuhara, 2016). P10 said: "*I was completely obsessed with programming, so I didn't need a reason to continue doing it. As opposed to anything else, it was just natural.*" P5 told: "*unless we were doing the exam coursework, we were writing games. All the time. All the time.*"

This focus can lead to neglecting other activities, as P8 described: "*My whole school effort went down, and I was only at the computer, when I was 12 or 13, I think.*" He also mentioned how special it feels to be programming: "*... and that's the thing I love about the computer, because you dive into it and you get emerged in the technology and you discover things. That's ... yes, it's special*".

There are also related, less extreme attachments to programming. P4 stated that being interested is the only way for him to learn "*I cannot stay focused if I am not interested ... A tight feedback loop and a problem is the only way I can stay seated and not move.*"

#### 4.3.4. Solitary Activity

Despite the fact that some encouragement in the form of special permission seems to have been needed, this does not mean that the act of programming itself was always done in pairs of groups; the contrary seems to have been true: programming was often done alone, but talked about with friends or parents, or about showing them running programs, or talking about ideas for projects.

P6 for example mentioned not having people to program with: "*... I don't think I knew anyone else that was doing it in primary school... Only when I got to secondary school in that second year, so I would have been be 12 or 13.*"

Some participants even explicitly mention that programming with other people was not desirable for them. P10 says that "*the programming interest that . . . I had was very much a solitary activity, which it didn't have to be, but for me it just made the most sense.*". P3 states that "*You learn a lot from working in a team, I know. But for me it was never that interesting.*"

Several participants mentioned that they believe that there are certain types of people drawn to programming, even though these were different traits. P9 names being an introvert: "*I'm not the most outgoing type . . . so I spend a lot of time with the computer*", while P8 mentions creativity: "*I think I'm a creative person and I learned programming by- because I'm creative.*"

## 5. Discussion

### 5.1. Purpose

There were several sources of motivation and purpose that our participants drew from: The immediate motivation was usually to create games, and, for a younger participant, creating websites. These motivations were, however, enabled by several factors, including the social permission and support by parents and peers as well as having time to explore the use of computers.

Games were frequently referred to by our participants and also described as being as one of the two things that 80s microcomputers could be used for, the other activity being programming. These activities were not separate: Programming could create games, and enthusiasm for games could motivate to do more programming. Gaming was the factor that eased socialization around computing topics, and aside of games itself, young people also exchanged magazines which often included code listings to create games.

The connection of programming and computer games has a long history: Even when computers were still expensive expert tools at universities and government institutions, programmers created games for them. Later, the association of microcomputers with games was also the result of marketing these as both toys and educational tools to boys (Juul, 2024) Other creative uses might be less prominent, but also have a long history, for example using them as creative tool in computer generated art, for making music or for creating textile patterns.

The focus on programming and games as well as the social legitimization of these activities was only possible under the assumption that it was okay for kids to be bored, to have free time and to spend that time at computers: Boredom was mentioned as a major factor. This makes sense, as a lot of activities around learning programming need time and do not yield immediate results: Searching for a mistake in code can easily take hours and taking this time would be less likely if the life of our participants would have been more directed at either learning for school or spend in extracurricular activities away from their computer.

One thing to note is that purpose, contrary to the other two large themes, is not as strongly connected to programming. Teenagers do pick up music, reading, drawing or other activities with similar fervor. However, childhood programming forms a more direct pathway into an occupation, and thus young programmers are more likely to still work in the field of programming. As such they are bringing their purpose into their adult lives at a larger proportion then to people that used to like drawing or music as a kid. This might help us understand some of the culture of programming including a focus on doing hard things better (Hermans & Schlesinger, 2024).

### 5.2. Transparency

Today, the connection between games and code is not obvious: Games do not come with their source code and while creating games is still a possibility, it is not a common activity to do at ones own computer let alone smartphone or tablet. Often, such activities are actively prevented by the technology: Running custom code on your game console or on your iPhone is difficult and an exception. Websites today do still provide inspectable code, but since it is mostly machine generated rather than directly written by programmers, it is almost impossible to find out how the website works by inspecting the code.

This, however, was different for our participants: The explorability of how a game, or any other computer program works, was much stronger in the 80s and 90s: Programs often came as source code that could be read by users and later, website's source code could be inspected. Program code for games was shared in magazines and one participant mentioned that the creation of games was more fun than actually playing it (P5).

It is notable that participants learned from examples rather than mere trial-and-error: Engagement with coding could also be replicating code from a magazine or trying to replicate an example. Knowing that it was created with code that they had access to, had a twofold effect. Firstly, it meant that one could learn how another programmer archived a certain effect (the bouncing ball, the rounded corners of boxes...), and secondly, it was certain that somehow, with enough effort, one could replicate the effect.

The examples used were not designed to be used for education: That code of games or websites could be read and replicated was part of the medium itself. This does not exclude use structured learning materials, but made learning by self-selected examples possible. In particular, wanting to replicate something or getting some code running can provide a motivating challenge.

### 5.3. Challenge

There are many potential ways to engage with code, one of them being the challenge: Being strongly motivated by overcoming a difficult problem that, in the context of our research was usually self-imposed. It might feel very different, though, if a computer program that "should work" just does not: The computer can become an antagonist (Turkle, 2005, 102) and the artifacts seems to impose the challenge.

What participants programmed were usually copies or derivatives of already existing programs. The models from which to copy can be directly provided by listings in magazines and books: You could manually copy a game by typing the source code. However, examples can also be taken from other computing experiences, for example interesting looks of websites (P2) or existing games (P8: "*You saw in a game, and now you could do it yourself.*"). This is similar to honing one's drawing skills by creating drawings of fan-favorite characters. To work as an example to be copied or extended, programmers must assume that they actually are able to do this: A simple text adventure might be plausibly created by a single person, but a somewhat up-to-date-looking 3D action game is known to be the work of many people with different specializations and thus not a plausible thing to copy – though it might still serve as motivation to be, at one day, part of such a production.

At a first glance, Challenge might feel odds with Transparency, as the former focus on hard things, while the latter makes things more accessible. We however believe they strengthen each other. The transparency of code made it possible for budding programmers to see what they were up against, which looked challenging, but also doable.

Our analysis confirms several known phenomena: The supportive parents (Margolis & Fisher, 2001, 94), in our research particularly the fathers, providing both the hardware and parts of the legitimization of the hobby of programming, the solitary and often obsessive interaction with machines over longer times (Turkle, 2005, 191) and the motivation to find out "how it works" by reproducing it (Turkle, 2005, 74). All of these are male-associated stereotypes. Furthermore, all our participants used he/him pronouns, none she/her or they/them, with a majority coming from Europe and being typically between 30-50 years old, many of them fitting well in the cohort to whom (or their parents) microcomputers were marketed as an educational toy for boys. (Juul, 2024)

Thus, the question needs to be posed of what we think we can learn from our participants for the future of education and design of tools to be more inclusive. On one hand, we want to understand the past experience of being self-taught better. The idea of the self-taught programmer is a powerful concept that still does influence both education and legitimization of roles until today, so we hope to make sense of the current state of education and tools by our qualitative study. On the other hand, we think we take aspects from the learning experiences of our participants and find our more about both, what makes programming interesting and motivating while wanting to make it interesting and motivating. By

interrogating these aspects, we would like to see how the powerful trope of the "self-taught-programmer" might help to understand what can be motivating about learning programming for a broader audience, too. This necessarily involves not to replicate the past, but take what we learned as a starting point for re-interpretation.

## 6. Implications for programming education in schools

As we mentioned in Section 4, children in our study were enabled in their learning by *social permission*, parents encouraged the learning of programming by providing books and computers (particularly since computers were an investment back in the day). However, there was a balance, as parents were not actively pushing programming, since they were not programmers and did not know programming themselves. Furthermore, contexts were created in which kids were interested, giving them a purpose.

How can our results inform educational practices and tools for modern programming education, especially in the context of schools rather than private bedrooms?

### 6.1. Purpose: Concrete Goals

What we see in these interviews is that once these kids had in mind a thing they wanted to create, and something, both in the online or offline world that they wanted to be part of giving them a purpose. With that purpose, their motivation remained extremely high even though they also describe getting stuck and feeling frustrated.

This has implications for programming education, since lessons typically do not contain a lot of *purpose creation*, or at least not in the first stages. Specifically in programming in a textual language, extensive instruction in the intricacies of a programming language syntax are needed to even create simple programs. Some students never reach a good enough understanding of programming details to create programs, a concept known as the *syntax barrier* (Denny, Luxton-Reilly, Tempero, & Hendrickx, 2011; Gilsing, Pelay, & Hermans, 2022).

Because of the complexity of languages, programs in education start small with the canonical ‘Hello World’ program. It takes a long time before learners exposed to examples of what would be possible with programming in the medium to long run. There are exceptions of course: Systems like SonicPi or Processing are focused on different goals (music and art) and as such do take learners along in the path of possibilities. However, they still require complex syntax that distracts. Teaching methods such Use-Modify-Create model (Franklin et al., 2020) aim to start with a concrete goal which children adapt, but as the programs must be understandable for learners, they remain simple in many cases.

While not all kids might be (so strongly) motivated by purpose and goals, one could imagine that more children would be if programming education placed more emphasis would be placed on what programming is *for*.

### 6.2. Transparency: Modern tools lack it

In the interviews, we find that it was easy to inspect and change the code that creates the software. The programming languages were relatively simple and systems allowed for easy inspection.

This accordance could be called ‘tinkerability’: the option to look at something (in our interviews: a game or a website), and to look inside the code. This transparency motivates by demonstrating and recreation of the artifact is possible; the purpose is building something like the explored artifact, the puzzle is figuring out how the current code works and how to change it.

Sadly, in the modern digital world it is much harder to explore common technologies when compared to the 80s and 90s. Websites are no longer text with some added markup which is easy to read. Today, much code is hard to read and machine-generated by other tools. This prevents users from understanding the source code but for the companies creating the websites it enables particular programming workflows, smaller file sizes and defense against blocking of advertisements. Games no longer ship with their source code, game consoles prevent users (without deep technical knowledge or expensive tools) to run their own games.

Despite efforts to merge them, such as the work of Bret Victor[3], current programming environments almost always separate written code from the running program in most cases, while our results stress that that division limits the possibility to explore and learn in the broadest sense.

Our findings point to the need for simpler languages for education, with less and not more features, which make the connection between code and execution model more, and not less transparent. Such a move might be contrary to the assumption that visual programming tools such as Scratch are the best first tool for children to start in their programming journey. Scratch's interface does not enable or encourage 'inspection', you cannot click a block and ask what it does. Scratch is not only opaque, but also supports complex language features—also termed *high ceiling* (Resnick et al., 2005)—and nowadays of course also features AI options such as text-to-speech via Amazon Polly [4]. These features might increase purpose and challenge, but decrease transparency even more, as they lean on increased abstraction of the machine.

While browsers do still support features to inspect code, today's web pages are in most cases not understandable by inspecting their source: Much of the markup will be generated server-side, without showing how it is generated. Executable JavaScript code will also often be generated by other tools (like TypeScript) and then, in addition, be minified, with whitespace and variable names stripped from the code to save bandwidth. While the code formally is accessible, it is barely possible to read, even for experts, since how the code was originally written and how it appears to the user are so different.

In other words, the tools that people are creating for programming education and the tools that professionals use for programming both no longer support the transparency that we find is needed for self-teaching. This is a hard problem to address, a teacher could of course create an old-style web 1.0 website, which would be achievable to analyze for learners, but it is unlikely that this would motivate learners, since they are used to more complex websites. The first author, for example, did not learn programming as a child, since they only knew programming in QBasic, which in the 90s already looked old.

### 6.3. Challenge: Programming tasks as puzzles

Of the three main themes, challenge might be the hardest to integrate into current educational practices and tools. In each group of learners, some will crave hard problems to sink their teeth into (often associated with status, see (Hermans & Schlesinger, 2024)). Other students however might feel insecure, or not interested in the topic, and as such night not be excited by challenge. These are much more likely students with lower prior knowledge and lower self-efficacy, i.e. girls and kids from lower social economic families. If we want to entice them to like programming, challenge is unlikely the way.

It would be possible to integrate challenge into formal education. Challenge can be compatible with many education systems, because in schools lessons are often structured around solving problems with different levels of difficulty. However, these challenges are in most cases given by teachers rather than self-imposed and it remains an open question whether similar excitement and motivation can be created in that fashion. We also encounter the population issue: a potential problem of focusing eduction on challenges can be that great difficulty and overcoming it is emphasized, thus potentially only focusing on the most advanced and competitive learners.

One fruitful related concept here can be that of a *puzzle*. Puzzles are often created by another person, like a teacher giving tasks in school. Puzzles also do not need to be very difficult to be enjoyed. Think of Sudoku or crossword puzzles: Solving them is not the heroic work of lone geniuses but can happen at the kitchen table and be a social activity ("Do you know a word with 5 letters that means...?").

Using a puzzle-model for giving programming tasks has limits: Unlike the challenges of our participants, the puzzles in education are not self-imposed, and thus less connected to activities that learners find

[3] https://worrydream.com/LearnableProgramming/

[4] https://aws.amazon.com/blogs/publicsector/scratch-and-aws-educate-build-new-activity-for-annual-hour-of-code/

personally exciting. A related problem is that it is easy for experienced programmers to come up with programming puzzles that are exclusively abstract and math-like, while learners might be interesting in other ways of using code, for example to create visual patterns or sounds.

In summary, while it might be possible to motivate some learners with challenge, it will not be easy to replicate the success of self-driven motivation in a classroom for all learners, since what one child finds interesting and challenging, might be confusing and frustrating to another.

## 7. Concluding Remarks

Our goal for this paper was to find out what being self-taught was like for people who were children in the 1980 and early 1990s. We did this based on interviews and thematic analysis. Our goal was to find out what shaped their experiences and enabled them being self-taught as well as what we could learn from these experiences to shape today's tools and programming education.

Based on the interview data, we constructed three main themes: Purpose, that is having a motivation for programming; Transparency, the possibility to inspect how other programs work and to tinker with them; and challenge, a motivated engagement with difficulties.

We discussed how these findings connected to existing research and how the insights could be helpful for helping kids to learn programming today, including those who do not self-teach but get in contact with programming first in the classroom. One interpretation of the results might be to double down on existing ideas of self-guided discovery without guidance which would most likely benefit mostly boys already motivated about learning programming. However we want to emphasize that the basic principles of Purpose, Transparency and Challenge can be also interpreted in a less gendered way: Purpose does not need to be utilitarian or focused on the technology for its own sake. Transparency also means a focus on a powerful yet small and thus understandable feature set and focus on community exchange of code and how-tos rather than only results. Challenge, while on the first glance heroic and conflictual can also be understood as puzzle-like: An activity that follows rules, has an initial state and a solution and which can also be done collaboratively.

Our works shows that we can gain insights from the past experiences of programmers today, while, instead of merely repeating what worked for them in the past, build more inclusive tools and pedagogies based on these insights.

## 8. References

Aivaloglou, E., & Hermans, F. (2019). How is programming taught in code clubs? Exploring the experiences and gender perceptions of code club teachers. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (pp. 1–10).

Battista, M. T., & Clements, D. H. (1986, January). The effects of Logo and CAI problem-solving environments on problem-solving abilities and mathematics achievement. *Computers in Human Behavior*, *2*(3), 183–193. Retrieved 2019-06-07, from http://www.sciencedirect.com/science/article/pii/0747563286900026 doi: 10.1016/0747-5632(86)90002-6

Braun, V., & Clarke, V. (2019, August). Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health*, *11*(4), 589–597. Retrieved 2021-04-02, from https://doi.org/10.1080/2159676X.2019.1628806 (Publisher: Routledge _eprint: https://doi.org/10.1080/2159676X.2019.1628806) doi: 10.1080/2159676X.2019.1628806

Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011, June). Understanding the syntax barrier for novices. In (pp. 208–212). ACM. Retrieved 2019-09-07, from http://dl.acm.org/citation.cfm?id=1999747.1999807 doi: 10.1145/1999747.1999807

Franklin, D., Coenraad, M., Palmer, J., Eatinger, D., Zipp, A., Anaya, M., ... Weintrop, D. (2020, August). An Analysis of Use-Modify-Create Pedagogical Approach's Success in Balancing Structure and Student Agency. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (pp. 14–24). New York, NY, USA: Association for Com-

puting Machinery. Retrieved 2025-02-26, from https://dl.acm.org/doi/10.1145/3372782.3406256 doi: 10.1145/3372782.3406256

Gilsing, M., Pelay, J., & Hermans, F. (2022, December). Design, implementation and evaluation of the Hedy programming language. *Journal of Computer Languages*, *73*(101158), 1–17. Retrieved 2023-02-14, from http://www.scopus.com/inward/record.url?scp=85140296845&partnerID=8YFLogxK doi: 10.1016/j.cola.2022.101158

Goldberg, A., & Kay, A. (1977). Methods for teaching the programming language Smalltalk. *Report No. SSL*, 77–2.

Hermans, F., & Schlesinger, A. (2024, October). A Case for Feminism in Programming Language Design. In *Proceedings of the Onward! '24*. Pasadena. (To appear) doi: 10.1145/3689492.3689809

Juul, J. (2024). *Too much fun: the five lives of the Commodore 64 computer*. Cambridge, Massachusetts: The MIT Press.

Kirkpatrick, G. (2017, May). How gaming became sexist: a study of UK gaming magazines 1981–1995. *Media, Culture & Society*, *39*(4), 453–468. Retrieved 2025-04-08, from https://doi.org/10.1177/0163443716646177 (Publisher: SAGE Publications Ltd) doi: 10.1177/0163443716646177

Lewis, C. M., Anderson, R. E., & Yasuhara, K. (2016, August). "I Don't Code All Day": Fitting in Computer Science When the Stereotypes Don't Fit. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 23–32). New York, NY, USA: Association for Computing Machinery. Retrieved 2025-05-20, from https://dl.acm.org/doi/10.1145/2960310.2960332 doi: 10.1145/2960310.2960332

Lockwood, J., & Mooney, A. (2018, February). Computational Thinking in Secondary Education: Where does it fit? A systematic literary review. *International Journal of Computer Science Education in Schools*, *2*(1), 41–60. Retrieved 2025-05-27, from https://www.ijcses.org/index.php/ijcses/article/view/26 (Number: 1) doi: 10.21585/ijcses.v2i1.26

Margolis, J., & Fisher, A. (2001). *Unlocking the Clubhouse: Women in Computing* (First Edition ed.). Cambridge, Mass: Mit Pr.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic Books, Inc.

Resnick, M., Myers, B., Nakakoji, K., Shneiderman, B., Pausch, R., & Eisenberg, M. (2005, January). Design Principles for Tools to Support Creative Thinking. *Report of Workshop on Creativity Support Tools*, *20*.

Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M. L., Minsky, M., ... Silverman, B. (2020, June). History of Logo. *Proc. ACM Program. Lang.*, *4*(HOPL), 79:1–79:66. Retrieved 2024-09-22, from https://dl.acm.org/doi/10.1145/3386329 doi: 10.1145/3386329

Turkle, S. (2005). *The Second Self: Computers and the Human Spirit*. The MIT Press. Retrieved 2024-04-22, from https://direct.mit.edu/books/book/2327/The-Second-SelfComputers-and-the-Human-Spirit doi: 10.7551/mitpress/6115.001.0001