Attitudes and Self-Efficacy in Young Adults' Computing Autobiographies

Andrew J. Ko
The Information School | DUB Group
University of Washington
ajko@u.washington.edu

Abstract

Little is known about the formation of people's first perceptions about computers and computer code, yet it is likely that these impressions have a lasting effect on peoples' use of technology in their lives and careers. Brief autobiographical essays about these first impressions were solicited from a diverse population of young adults and these essays were analyzed for factors that contributed to positive and negative attitudes about technology, formation of self-efficacy, and authors' relationship with computing later in life. The results suggest that first encounters with code must be accessible, error-tolerant and socially engaging, that mentorship can be a crucial factor in the acquisition of programming skills, and that cultivating positive self-efficacy in programming skills requires repeated positive exposure across the lifespan. These results raise several issues for novice programming languages and tools and suggest a number of new approaches to computing education.

1. Introduction

Society is increasingly reliant on the world's software infrastructure. From younger populations who live their personal lives on social networking sites [16] to growing elderly populations discovering e-mail as a way to connect with family [8], people around the globe are accepting technology into their lives as an essential means of personal contact and commerce.

The people who improve and maintain this software infrastructure, however, are a much smaller group. These are the computer scientists, programmers, IT administrators, web developers, and systems analysts, among us, the technically trained individuals who at some point in their lives, decided that computer technology would not only be part of their personal lives, but also the core of their professional careers.

Unfortunately, while these professionals are in rising demand [7], technical professions such as these are still viewed as the least popular subjects in some countries [13], while computer science enrollment in the United States is dropping or stagnant. Furthermore, people's first experience with computer code and programming are often so negative, it is enough to turn them away from technical professions [2]. As our dependence on information technology increases, these perceptions and first impressions about computing are perhaps the greatest barrier to growing and maintaining a worldwide workforce of computing professionals.

What then makes some first exposures to computing successful and others not? Is it the design of the tools? The parental support? The feedback of teachers? What other factors might be involved? There has been considerable research on *teaching* programming (e.g., 11) and also work about academic enrollment in computer science (e.g, [10]), but little work that analyzes people's encounters with computing across the lifespan, and how these encounters interact with career choices. This study begins to address this gap.

To explore people's encounters with code, we solicited autobiographical essays from 58 students pursuing a degree in *informatics*. Their essays revealed a number of common paths leading to and away from computing careers, indicating several crucial factors in the decisions to pursue technology related careers. One of the findings was that respondents' early encounters with code were often *simple*, *social* and *rewarding*, whereas their later encounters with code were *painful*, *disheartening* and *forced*. Furthermore, many respondents' impressions of programming, despite positive early experiences, were conflated with impressions of academic computer science, which was described as *cold*, *rigid*, *proud*, and *divorced from any relevance to people and society*. These results give researchers and educators a more nuanced idea of how to address this present and future challenge of encouraging the pursuit of technology careers.

In the rest of this paper we review prior work that has investigated people's perceptions of computing careers and then discuss our methodology and results in detail. We end with a discussion of the implications of our results on education, software development tools and academic and industrial recruitment.

2. Related Work

At the heart of any person's career trajectory is one's perceived *self-efficacy* at a set of skills [1]. This notion of self-efficacy refers to people's beliefs about their abilities to exercise influence over events that affect their lives (at work or elsewhere). Research on children's career trajectories shows that perceived self-efficacy influences aspirations, the strength of commitments, the quality of strategic and analytical thinking, motivation and perseverance, and even causal attributions of success and failure [1]. In general, evidence shows that children's formation of self-efficacy is crystallized at a young age.

We know a great deal about what influences self-efficacy. Parental engagement is one factor, but gender biases imposed by parents and society also play a role. This is related to Erikson's theory of psychosocial development [9], which centers around the notion of *identity* (the sense of self we develop through social interaction). The later stages of identify formation center around the formation of competence and belief about skills. People who receive encouragement from parents, teachers and peers develop confidence in their ability to be successful at particular tasks. Those who do not remain unsure about their abilities. This theory would predict that individuals who learn about programming with the help and support of mentors or with the positive reinforcement of peers, would form an identity around their newly developed skills.

Some prior work relates self-efficacy to computing careers. For example, one of the most visible efforts in past years has been efforts by computer science educators to understand the gender gap in computer science student populations. In a series of studies during the 90's, Fisher and Margolis investigated gender issues in an undergraduate computer science program [10] finding that women had significantly less coding experience, that they came to computing later in life than men, and that women were conflicted about adopting "geek identity" so pervasive in the computer science program. However, these studies involved students who already had developed some perceived self-efficacy in computing; they do not explore the origins of this self-efficacy.

Other efforts have focused on the debate around the notion of *digital natives* [4], who grow up with technology and the expectation that one continually learns and cultivates skills around new technologies. The argument is that digital natives show few aversions to learning technologies, because from an early age, they were expected to learn new technologies. The hypothesis is that this early engagement leads to improved *learning skills* (hence, self-efficacy). Detractors point out that not all children and young adults have the same comfort with digital technology, even when surrounded by it from an early age.

Other work focuses on designing new programming tools to foster self-efficacy. The Alice 2 programming environment [6] was designed to create interactive 3D worlds, and later augmented to facilitate storytelling by middle school girls. In studies of these efforts, the type of content and ease with which people could create personally relevant content were both crucial factors in affecting motivation [14]. Hundreds of similar systems have similar goals, with varying degrees of success [14]. Beckwith et al. studied the influence of tool design on self-efficacy in these types of programming environment, revealing that the design of computing tools can affect self-efficacy in subtle (and gender-specific) ways [3]. Blackwell's attention investment model, which proposes concepts of risk and reward perception, is also related to self-efficacy, in that both operationalize peoples' beliefs in their ability to acquire new skills [5].

Perhaps the most relevant prior work was a study of German university students, comparing computer science majors to psychology majors [18]. They described the psychology majors as having low technical self-efficacy and viewing software one of many tools, and described the CS majors as having high technical self-efficacy and as seeing no boundaries between software use and software design. The study presented here is different in *sample*: the students in the present study are from the United States, which may reveal some cultural variations and were all majoring in a technical, but non-CS degree. The essays we solicited were of a longer and more consistent length.

3. Method

The goal of the study was to understand the technical and social contexts of peoples' encounters with code. To gather this data, students in an *informatics* class about user-centered design were asked to write 1000-1,500 word biographical essays on first encounters with computing technology, first encounters with computer code, the current role of technology in their lives, and the relationship between these three topics in their lives. As part of their essays, they were asked to provide their age, gender and city of birth. They were also told that the essays would be kept confidential, with the exception of anonymized quotes. All students agreed to write essays instead of completing an alternative course assignment.

One reason for choosing this sample is that informatics degree program from which students were recruited has a reputation for attracting students that either did not get accepted into a computer science degree program, do not enjoy programming (though many do), but *do* enjoy technology. Our analysis of the respondents' essays was an opportunity to explore a potentially diverse set of motivations for choosing the degree program that they did.

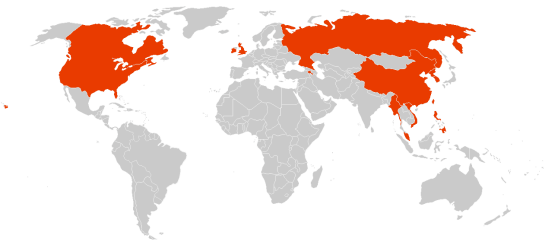

Figure 1. Countries in which respondents were raised.

To encourage some degree of uniformity to the essay topics, respondents were provided with a "lure text," an example essay by the author[1] to direct participants to interesting topics without explicit instruction to do so. There are a number of ways in which this text could have influenced responses, for example, by creating expectations about a certain structure, tone, or type of story. Given these limitations, the essay form gave respondents time to reflect on and refine their responses, which would not have been possible with interviews or surveys.

Among the 58 respondents, there were 46 men and 12 women, with an mean age of 22 (±3), ranging from 21 to 41. As shown in Figure 1, respondents were raised in a variety of countries, including the United States, Korea, Russia, Philippines, Taiwan, Armenia, Myanmar, Malaysia, Vietnam, England, and China. Most respondents were from the United States (44), and of these, most were from the state of Washington (25). Of course, many of the respondents had moved multiple times, whether within the US or to it. All students had taken three introductory computer science courses, including advanced data structures.

The essays were analyzed by first reading the essays in detail, recording possible trends in the essay texts related to factors that respondents described as influencing their confidence in technical skills. A second pass through the essays was performed to assess the prevalence of each of the trends identified in the collection of essays. Trends that appeared in roughly more than a quarter of the essays were selected read in more detail. The results presented in this paper, and the quotes selected to illustrate them, represent the result of this process.

4. Results

While there was considerable variation in the respondents' essays, there were also a number of trends in respondents' encounters with computers and code and the role of mentorship in supporting self-efficacy. For example, no single encounter with technology or code was seen as fundamental to pursuing technology careers. Instead, encounters were *cumulative* and a mix of positive and negative. Respondents identified several specific technologies as crucial in determining their attitudes, including "The Oregon Trail," an educational software game of the mid-80's, programmable Texas Instruments graphing calculators (such as the "TI-82") in the 90's, and the *view source* command of early web browsers. Finally, positive encounters with code involved *friendship*, *mentorship* and *play*; negative encounters involved a distinct sense of *failure*. This raises issues of social context, motivation, and self-efficacy. We discuss each of these trends in turn, with supporting quotes from the essays. When quotes are used, they are intended to capture a larger pattern across all of the essays.

4.1. Encounters with computers

Although the focus of this study was respondents' encounters with code, the respondents' encounters with digital technology are important in making sense of the their programming experiences. One way of characterizing these experiences is to think about the *roles* that technology played in respondents' lives. One respondent from Vietnam described these roles well, in describing the first time he saw a computer:

> It was a place where a lot people were seating [sic], sticking their eyes to square boxes, some people were smiling, some people were laughing, also some people looked serious but they were doing same thing that their two hands always and always on a thin rectangle box looked like type writer.

Across the essays, respondents used computers to *learn*, *play*, *socialize* and *work*. Based on the data, the primary role was *play* in the form of video games. They played games on video game consoles, as well non-educational games on personal computers. Many expressed high positive affect for video games both in their early youth and present life:

> From a very early time in my life technology, and particularly video games, have had a large influential role in my life...This shaped my free time, activities, the people I got along with, and my class choices as I grew.

Others appropriated tools intended for work to play:

> I remember opening the calculator, paint, and notepad programs. The calculator was pretty nifty at the time, but I was not as thrilled by it as much as paint. I had a blast with that program.

Not all respondents mentioned playing video games, and if they did, not all mentioned enjoying them. Some suggested that games merely passed the time.

> I never really understood the minesweeper game mostly because I kept hitting the mines and losing right away. So I mostly just played around with solitaire. After a while of just playing games on it, I became a bit bored. That's when I decided to explore around and look into the other folders.

[1] The lure text used is available at `http://faculty.washington.edu/ajko/appendices/lure.html`.

Many respondents' first encounters with computers as entertainment occurred in *learning contexts* at school. Most of the respondents were part of an era in North America in which computers were just beginning to play a role in formal education. Respondents mentioned playing educational games, especially *The Oregon Trail* and *Math Blaster*, and learning how to use word processors and spreadsheets. While in many cases respondents were required to participate, most enjoyed them:

> The more I struggle to remember that day, the easier it all comes back; the sharp click of the 1-button mouse, the confusing array of letters not arranged in any discernible order on the keyboard, and of course the sheer exhilaration that was The Oregon Trail (and dying from dysentery)...From that day forward, my mind married the two concepts of computing and fun.

Respondents also used technology to *socialize*, sometimes in the form of collaborative online games, or just to communicate:

> I remember making my mom sign up for Prodigy, Compuserve, and finally staying with AOL. Compuserve had a chat system and I remember being so amazed at how cool it was to talk to people all over the country and world in an instant. I made a couple online friends.

Some respondents mentioned being required to use computers as *work* tools (for example, as a word processor or spreadsheet), most of the respondents' earlier encounters were not particularly goal driven.

These four roles of computers were not always independent. The Oregon Trail was entertaining, but it was also intended to be educational. Some respondents mentioned *playing* with Excel, by typing in numbers and making graphs. Most of the games that respondents played for fun were played with others.

4.2. Encounters with code

For most respondents, computer code played a later role in their gravitation toward technology careers, though some first encounters with code were also first encounter with computers. The most common first encounters with code were the *view source* command of most browsers and the DOS command prompt. Respondents indicated that these were the first times that they had ever seen computer code:

> [Dad] wasn't the guru with the answers to my endless list of questions and I'm glad he wasn't because that forced me to find out the answers for myself. ...he gave me one last piece of advice: I was pointed in the direction of View > Page Source and found the holy grail, the motherload, the Fort Knox of confusing gibberish.

Other common encounters included the BASIC programming language included with many common platforms in the 80's and 90's:

> My intentions were to create a game whose title in this manual had attracted my attention. It wasn't until years later I realized I had been copying BASIC code that would have programmed a game for me.

Texas Instruments graphing calculators were also a common for respondents first code encounters:

> I spent quite a bit of time with a third good friend (whom I also live with) making games on the calculator. The culmination of our efforts was in a text based role playing game we called Arena. There were duels and experience points and items and weapons and spells! What beautiful days those were.

Less common encounters were through productivity tools, for example, one student learned Excel formulas:

> We needed to derive a set of data from a given set and then present it in a graph. At first I thought it wasn't too useful to know spreadsheet codes because I can always do the computation using my calculator. However, it became painful and time consuming when I dealt with more than one hundred entries... I learned to appreciate the use of computer codes the hard way.

Some respondents learned to code in video games:

> Starcraft had a map editor, which essentially allowed you to design a stage. The editor had options to cause events or certain things to trigger events. The coding was very simple... It was difficult to learn at first, but it got easier when I looked at other examples of stages to help me with the editor.

The crucial factor behind all of these encounters, however, were the contexts and motives. One of the most common reasons to learn the DOS command syntax was to launch games with the help of a parent, sibling, or friend. In other cases, parents had shown their children the *view source* command or they had stumbled upon the command when trying to learn how to create their own web pages for social reasons, including publicizing a group for a multiplayer online game or creating web pages to share with friends (pre-MySpace). In the case of the graphing calculators, the impetus for programming was often to create and play games during math class, or to create simple tools to help automate computations for homework or exams.

Of course, many first exposures to code were more structured, such as after school programs, high school courses, or introductory college programming courses. There were a number of distinctions between these different types. First, the after school, summer camp and high school courses were often described as elective and voluntary (or by the suggestion of the parent). Respondents attitudes towards these were typically positive:

> Once I reached high school, ... I took the class the first chance I could, and immediately found myself excelling ahead of the class due to my prior experience with web development growing up. The teacher of the class also happened to be my new cross country coach... Through these two activities I became very close with my teacher and coach, and continued to take many programming classes.

In contrast, college programming courses provoked strong negative feelings from most respondents:

> This carried me through high school into college where my love of programming has been brutally murdered by out of control CS Monsters. I said earlier that my love of the subject matter was inspired through socialization. Well, many of the people I have met in the CS major have grated on my nerves like a cheese grater. They are possibly the most proud people I have ever met.
>
> After 143 [a data structures course], I felt drained and worn out from programming. I started to think I wasn't meant to do any coding because of all the stress I went through...
>
> I realized that I not only wanted to gain the technical background of how computers work and how to develop applications, but I also wanted to see how that affected people who use computers. ... what good it was to build a super computer that did anything imaginable, if there was no demand for it.
>
> ...the next class got more into serious programming, and more relevant to what professional programmers do for a living. I absolutely loathed this class and its content. I found it boring and stale, like a math class.

These and other opinions generally centered around the formality of the course content and its and irrelevance to humanity and society.

Not all respondents disliked the introductory computer science courses.

> Finally, everything fell in place and I felt like I just climbed Everest. The thirst for this feeling of accomplishment is what kept me going in between each coding project.
>
> My scattered understanding of code became more organized when I took CSE 142, the introductory Java class. Again I was fascinated by the intricacies and patterns that emerged.
>
> ...made that quarter a programming hell. Yet, I persevered, and actually really enjoyed the long hours I put into coding and optimizing algorithms.

One last kind of first encounter with code was as an employee. Several respondents' first encounter was in testing code, or in creating web sites for a company. Of course, the motivational subtext to these, as compared to classes, is quite different: respondents talked about their skills being used for something explicitly useful as well as being paid for their work:

> Six months working at Boeing and continue working while in school as intern (part time work) I have successfully made one software engineer got mad on me because of 15 software bugs I found on one of his software client (part of the whole application).

There were other unique circumstances, including one respondent whose boyfriend taught her how to program in order to help her pursue a new career. Others learned on the job, using code as a means to accomplish their work:

> ...my supervisor asked me if I could work on a Visual Basic script that would automate weekly processing of billing reports. He told me I could either take that project, or continue do boring billing reports in Excel. I took the more challenging route and ended up producing a very useful and powerful script that saved the company many man-hours and dollars.

4.3. Mentorship

Mentorship was often a crucial component of respondents' encounters with code. In some cases, parents were a guiding force in respondents' technology experiences:

> When my step-dad ran out of things to take away, he finally decided to give me things to do. One day he purchased a big book of Borland C++ and told me that once I finish every single exercise in the book and pass all the tests, I could get all my other freedoms back. I toiled over it for four long months but came out of it a better man.
>
> The first one was my dad teaching me basic DOS commands because the computer games that we had at the time were all DOS based games.
>
> I would be willing to say though that my first exposure to code set me on the path to becoming a developer "just like my dad"
>
> ...my father never ceased to bring home the newest and coolest thing that technology could offer.
>
> Another fond memory of going to work with my father was the dumpster diving... Before each move, there would be dumpsters lined up in the aisles for people to discard unwanted computers and components... I used these computers to build my understanding of how to install and configure an operating system.

Clearly, some of these mentoring experiences were quite heavy handed, while others were more subtle. This was also true for mentoring friendships and for respondents' own mentoring experiences:

> I met my best friend — through a hacking competition at college. Remembered we were assigned at the same team... We eventually won the contest and he was the first guy I ever met really know about Linux, a world that I never know before.
>
> ...many of my close friends on my street had gone through similar experiences and were intrigued with computers as well. My first exposure to computer code came in 6th grade when one of my older friends showed me the website he had written in HTML.

Underlying these experiences and others were several forms of coercion, such as the expectations of friends (to create a website to support a social group), or the expectations of parents and teachers.

4.4. Programming and Self-efficacy

It was common for respondents to describe their encounters with code as *life-long*, *indirect,* and *cumulative*. There was no single event in each respondents' story that was the determining factor in their (dis)interests in pursuing technology careers:

> I don't think any one of those experiences set my life on a path of how and why I use computers or even why I want to be at the forefront of computer technology development.

To respondents, none of these encounters explicitly changed their attitudes toward code; rather, technology played a slowly increasing role in their lives until it was recognizably important. These experiences over each

respondents' lifespan were often disjoint, and separated by many years. However, respondents were definitive in their descriptions of their career decisions. Most focused on the negative perceptions of programming as an activity:

> So I believe my faculties to work with a computer were more than capable at this early age. Unfortunately none of the programming I attempted ever succeeded, I still remember the bitter taste of failure. And now that I mention it, that might have been my first conscience realization of failure.

> I know that programmers will make pretty good money, but I'd rather work at a place that I enjoy.

Others focused more on perceptions of themselves that made programming an unsuitable career choice:

> I know I'll never be good enough to make decent games and I don't think I want to do that as a career because of how time consuming it can be and not to mention it's a pain in the butt to do as a job.

> When college came I fought against this notion of not liking programming and eventually gave way to not being good enough at programming...It wasn't that I hated programming it was that I didn't have the patience or the trust for it.

Those that enjoyed programming tended to also focus on the characteristics of programming activity, rather than characteristics of themselves.

5. Discussion

The intent of this study was to document peoples' encounters with code across the lifespan, in order to learn about how future generations can be enticed to pursue technology careers. Like Schulte and Knobelsdorf's study [18], we found a continuum of attitudes, with some respondents viewing software as a tool to use, and others blurring the line between software *use* and software *design*.

In addition to replicating these results, our data also revealed several insights about how students arrived at these various perspectives on technology:

- There were several specific technologies that respondents remember influencing their interest in code, including *BASIC*, *Texas Instruments graphing calculators*, and the *View Source* command of web browsers.
- Mentorship, from parents, friends and teachers was a common but not universal way that respondents engaged with computer code. From the essays, it does not sound that mentorship is either necessary or sufficient, but it can be instrumental in influencing computing interests.
- For some people, even a lifetime of interest in technology and computing is not enough to entice them to pursue careers in programming, though respondents' perceptions about programming jobs are inaccurate and seem heavily influenced by the culture of college computer science courses.

These insights have several implications. We will discuss these starting from the scope of programming tools, moving outwards to the whole of society.

5.1. Implications for Programming Tools

Out of all of the technologies that could have successfully introduced respondents to programming, what made *Texas Instruments graphing calculators*, *BASIC*, and browsers' *view source* command so successful? We can speculate about key traits that these technologies have in common:

- *Accessibility*. The graphing calculators had a dedicated "PRGM" button; *view source* showed code in a single click; and in the days of DOS, the BASIC IDE was six keystrokes away from ones' first program.
- *Error-tolerance*. These three technologies were all either forgiving of errors, or had such simple syntax that errors were unlikely in simple programs. This meant that respondents were likely able to successfully create program output without immediately facing the challenge of debugging.
- *Socially-engaging output*. Respondents shared the results of the programs they created, whether through the utility of a solver on a calculator, the broadcasting of group identity on the web, or text adventures or audible output in a simple BASIC program. They created programs for the output that the programs produced, not for the program itself.

The motives behind most of these encounters with code were about supporting social relationships or reinforcing an interest in video games (for example, the desire to create the things that one enjoys using).

There are problems with most of these technologies today, first being that they are less accessible. "View source" features of web browsers are less viable ways to learn HTML because most web pages are too complex to see a simple example (or worse yet, it is automatically generated and no longer human readable). The textual and numerical output of graphing calculators may no longer be as enticing. Windows Vista and Mac OS X no longer come with simple programming tools like BASIC.

One of the central efforts to replace these experiences with something more generationally appropriate is Alice 2 [6], spearheaded by the late Randy Pausch. The design philosophy behind Alice is to create an accessible, error-preventing, and social programming experience. Although access is still a challenge (downloading over 100 MB and following an installation procedure is not nearly as simple as the examples from the 80's), it has more than succeeded in

motivating learners and supporting their social goals [14]. These same goals have been successfully introduced into computer science courses [11].

A central problem for future efforts in programming tools will be to solve these access problems, but also to provide people with a variety of content with which to be social. Are children still enticed by animation? What about social applications, such as on Facebook? How can programming environments be deployed on the web, to simplify access and support the social aspects of learning to program?

5.2. Implications for Early Education

As discussed early in the related work, self-efficacy and identity are formed earlier in life, thus any efforts to instill a curiosity about code must likely come early to be sustained later in life. The results of this study suggest that mentorship can and should play a central role in ensuring future generations of technology professionals. K-12 teachers, if given the opportunity, could seek opportunities for children to learn and create with code, allowing children to work together and help each other. To do this, researchers may need to identify and design technologies that support these endeavors. Educators then need to recognize that this type of learning benefits greatly from *collaboration*, and avoid worrying about cheating and credit, in favor of more marketplace-driven skill acquisition, as in some research-inspired workshops [17].

Several of the respondents mentioned childhood friends who essentially acted as a gateway to programming skills, enticing them with what the friend had created and then leading them through the process of acquiring the skills. In much the same way, several respondents mentioned parents playing the same role. There are a number of things that parents can do to increase the likelihood of these experiences. For example, parents could create and run summer workshops that introduce children to each other and to code. Researchers could be involved in designing curriculum, tasks and tools that increase the chances of establishing mentoring relationships between children and even other parents.

5.3. Implications for Higher Education

Almost universally, the respondents disliked the introductory computer science courses. This may simply be a sampling bias, since some of the students reported not doing well in these courses. On the other hand, respondents identified several specific critiques that have been reported in other studies of computer science education [10]:

- *CS department culture and pride*. Respondents felt that other more successful peers were elitist, exclusive and anti-social, rather than collaborative and inclusive. They also tended to generalize these perceptions to the software industry at large, rather than just the computer science department.
- *Mathematical instruction.* Respondents felt that computer science was too focused on the discrete math foundation of programming and less on what programmers create and enable with software.

These findings do not necessarily suggest changing computer science departments themselves. Many faculty in computer science departments may think of the above characteristics as desirable parts of computer science department identity. Whether or not this is an effective perspective, it is clear that this perception seems to force an artificial decision upon people with interests in programming: to be a "programmer," one must also adopt "programming culture." The unfortunate thing about this choice is that so many of the respondents in this study had lifelong positive experiences with programming, only to abandon them in college because of misperceptions about the similarity between academic computer science culture and the software industry. Most studies of software development practice show that far from being elitist and anti-social, it involves a great deal of social interaction, inclusion and creativity [15]. Thus, at the very least, computer science departments and other technology related academic schools need to ensure that their academic cultures more closely mimic the cultures of practice for which they prepare students.

5.4. Implications for Society

Although the focus of this paper was in cultivating technology professionals, there are countless other professions that have similar goals. There are several efforts to encourage more women to participate in science (e.g., *www.awis.org*), for example. There are countless efforts in middle schools and high schools to expose children and young adults to a variety of careers and professions, sometimes through invited speakers, but also through job-shadowing. Where do efforts to cultivate a generation of technology professionals fit into this larger societal context?

Perhaps the most important finding here was that no one positive experience with code was enough to keep a person engaged with coding throughout their lifetime; instead, it required persistent, cumulative positive exposure (as is suggested by general research on self-efficacy [1]). This suggests that not only will children need positive first encounters with code at a young age, but they will need additional, and different experiences throughout middle school, high school, and college. A single programming environment and tool will not be enough, nor will an engaging workshop. Instead, we need to design a collection of diverse experiences with code to cultivate future generations of technology professionals.

5.5. Limitations

As with any study, there are several limitations to the conclusions we can draw from this data. First, while the respondents' varied in age and backgrounds, they were all students at the same university and the majority were from the northwest region of the United States, which is one of a few regions with a predominance of professional programmers (some respondents mentioned that their parents were in fact professional developers). Furthermore, the sample did *not* include students avoiding technology related careers altogether. Therefore, there is some degree of self-selecting bias here, since most of the students in the course want technology to be part of their lives and careers. There may be other populations of people with similar experiences with code across their lifespan, but who did *not* pursue technology careers for a variety of reasons. This makes it difficult to know if the factors identified by respondents play a causal role in the pursuit of technology careers. Furthermore, the sample only included a few students also pursuing computer science degrees; computer science students may have substantially different experiences across their lifespan. Lastly, this sample comes from a particular historical era and earlier or later generations may differ greatly.

Besides sampling limitations, the insights discussed in this paper are a narrative constructed by the subjective and unavoidably biased views of the author. Great effort was made to avoid taking quotes out of context, but the generalizations and interpretations of the essays are inherently limited to a single perspective. There are several facts about the author that may help in interpreting the validity of the results in this paper: he shared many generational experiences with the respondents, programming and video games were a primary source of entertainment and social experiences in his life, and like many of the respondents in the sample, he was raised in the northwestern United States (but not by parents who worked in the software industry).

6. Conclusions

As society increases its reliance on software, it is increasingly important to maintain a skilled workforce of technology professionals. The results reported here suggest that improving education is an important but insufficient part of this effort. To truly cultivate a population of skilled software developers, parents of young children and K-12 educators should help children find socially engaging and mentorship-driven ways of *playing* with code across their lifespan. To enable this, researchers should focus on accessible, error-tolerant, and socially engaging programming languages to facilitate these efforts.

7. References

1. Bandura A., Barbaranelli C., Vittorio Caprara G., Pastorelli C. (2001). Self-efficacy beliefs as shapers of children's aspirations and career trajectories. *Child Development, 72(1),* 187-206.
2. Beaubouef T. & Mason J. (2005). Why the high attrition rate for computer science students: some thoughts and observations. *SIGCSE Bulletin 37*(2)*,* 103-106.
3. Beckwith L., Kissinger C., Burnett M., Widenbeck S., Lawrance J., Blackwell A., & Cook C. (2005). Tinkering and gender in end-user programmers' debugging. *ACM Conf. on Human Factors in Computing Systems*, 231-240.
4. Bennett S., Maton K. & Kervin L. (2008). The 'digital natives' debate: A critical review of the evidence. *British Journal of Educational Technology, 39*(5), 775-786.
5. Blackwell A.F. (2002). First steps in programming: A rationale for Attention Investment models. *IEEE Human-Centric Computing Languages and Environments*, 2-10.
6. Cooper S., Dann W., Pausch R. (2003). Teaching objects-first in introductory computer science. *SIGCSE*, 191-195.
7. U.S. Department of Labor, Occupational Outlook Handbook (OOH), 2008-09 Edition.
8. Day J., Janus A. & Davis J. (2005). Home computers and internet use in the united states. U.S. census bureau, Washington, DC. Retrieved Sept. 24th, 2008 from *http://www.census.gov/prod/2005pubs/p23-208.pdf.*
9. Eagle M (1997), Contributions of Erik Erikson, *Psychoanalytic review 84 (3)*, 337–47.
10. Fisher A. & Margolis J. (2002). Unlocking the clubhouse: the Carnegie Mellon experience. *SIGCSE 34(2)*, 79-83.
11. Forte A. & Guzdial M. (2005) Motivation and non-majors in CS1: Identifying discrete audiences for introductory computer science. *IEEE Trans. on Education, 48(2)*, 248-253.
12. Glaser, B. (1992). Basics of grounded theory analysis. Mill Valley, CA: Sociology Press.
13. Hendley D., Stables A., Parkinson J., & Tanner H. (1996). Pupil's attitudes to technology in key stage 3 of the national curriculum: A study of pupils in South Wales. *Int'l J. of Tech. and Design Education,* 6, 15-29.
14. Kelleher C., R. Pausch, & S. Kiesler (2007). Storytelling Alice motivates middle school girls to learn computer programming. *ACM Conf. on Human Factors in Computing Systems*, 1455-1464.
15. Ko A.J. DeLine R., & Venolia G. (2007). Information needs in collocated software development teams. *Intl. Conference on Software Engineering*, 344-353.
16. Lampe C.A., Ellison N., & Steinfield C. (2007). A familiar face(book): profile elements as signals in an online social network. *ACM Conf. on Human Factors in Computing Systems*, 435-444.
17. Rosson M.B., Carroll J.M., Seals C., & Lewis T. (2002). Community design of community simulations. *ACM Conf. on Designing Interactive Systems*, 74-83.
18. Schulte C. and Knobelsdorf M. (2007). Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. *Intl. Computing Education Research Workshop*, 27-38.