Challenges Faced by Teenage Programmers in Software Development

May 2, 2022

Word Count: 5,499

Abstract

As software has become an integral part of society, many teenage programmers have taken their hand at creating software applications, such as websites or video games. However, since programming is not such an easy concept to master, many developers have been left struggling in their coding journeys

As indicated in the current body of literature, many professionals have struggled with marketing, creating applications for multiple platforms, and garnering enough project sponsorship. Although these studies were great in that they described the common issues, they did not cover the teenage population, hence leading me to investigate the most common obstacles of software development amongst the teenage population and how they can resolve these challenges.

To investigate this topic, a 2 part study was conducted which surveyed teenagers on their most challenging programming obstacles and which interviewed industry level professionals for solutions/advice for these challenges.

It was then revealed that teenagers faced some of the same issues as industry level professionals, such as cross platform integration and brainstorming unique application purposes, however, they were unique in that they had issues managing their school work with their development time.

With the completion of this study, the obstacles of the teenage population were finally uncovered and teenage programmers are now able to refer back to the advice given within the study to further their software journeys.

Introduction

In today's world, people are constantly surrounded by different types of softwares within their everyday life. For example, the apps on their phones, the websites on their browser, and the applications on their desktops are all just a few types of software applications that people use on a daily basis. Especially since society is constantly drifting towards becoming more virtual, as seen by the recent increases in online storefronts, virtual chatting applications, and social media apps throughout the past decade, software development (which refers to the process in which computer programmers create mobile applications, websites, and more) is a field that has become more and more crucial for the advancement of today's tech generation. Due to this rising demand, the number of software developers throughout the workforce have risen dramatically over the past few decades.

Especially within the United States, the demand for computer scientists has risen by 1000% since the 1970s, creating over 2 million jobs available for Americans today (Mithas). This growing interest doesn't only stop at industry level computer science jobs, but it has also translated into the interest of teenagers at the University or high school level. For instance, from the year 2022 to the year 2023, there was an 8.13% projected increase of computer science majors throughout American colleges, showing that this particular field is one that many individuals consider fruitful or rewarding. Although there is a clear increase of interest within the realm of software development, this skill of computer programming isn't such an easy concept to master and has left many software developers struggling with the basic fundamentals.

For example, software developers today are having trouble learning how to code, learning how to attract their target audience, and learning how to launch their app on multiple platforms such as iOS and Android, hindering their app's functionality and usability. Unfortunately, the

issues listed above are only a few of the struggles computer programmers face on a daily basis, with other obstacles having an even larger impact on the success or failure of a software developer. Especially since this particular field represents a large chunk of the American workforce, it is vital to understand the common pitfalls these developers face today and what they can do to make their software products more successful in the future.

Literature Review

To find out where computer scientists struggled the most in terms of learning a specific programming language (the set of vocabulary and commands that allow humans to give instructions to a computer) or creating their own software application (such as an app or game) a literature review was conducted to evaluate the current research standing of this particular topic of inquiry. To start, current research suggests that some challenging aspects faced by novice programmers has been the familiarization of the abstract concepts associated with computer programming and the limited amount of resources available to programmers for learning how to code. According to Matthew Butler, a department chair from the faculty of Information Technology at Monash University, a survey conducted on 500 Computer Science University students enrolled in the course, FIT1002 Computer Programming at Monash University, had revealed that most novice programmers fail to progress in learning how to code due to "the conceptual difficulty of various elements of the curriculum, the [low] level of feedback that is available to students," and different types of learning styles implemented in computer science courses (Butler). To delineate which concepts made programmers struggle the most, Butler would then ask respondents to evaluate their understanding of a specific programming concept from 1-7 to find out which concepts were the most jarring.

Table 1: *Average Difficulty Rating for Understanding Curriculum (n=167)*

Topic	Conceptual Difficulty	Average Rating (out of 7)
Algorithms	High	1.98
Syntax	Low	2.56
Variables	Mid	2.14
Decisions and Loops	Mid	2.53
Arrays	Mid	4.17
Methods	High	3.92
OO Concepts	High	4.92
OO Design	High	4.52
Testing	Mid	3.60

As seen in the table above, Butler came to the conclusion that the main concepts that made learning how to code difficult for university students were the topic of Arrays, methods, and OO concepts. Furthermore, to understand which studying strategies would benefit these computer science students the most, this research would also dive deeper into the respondents' study habits. After some survey iterations, the study had concluded that for respondents who had a high level of understanding of their programming language, students had spent an average of 9.75 hours a week studying each unit from the textbook while only spending 4 hours per week practicing coding exercises. In terms of which environments aided university level programmers in development, the survey had revealed that 31% of students had preferred laboratory classes/assignments to develop their skills in programming while only 3% of participants preferred learning programming in peer groups, illustrating which learning methods were most desired by young coders.

In addition to research done on the problems of learning the fundamentals of computer science, there has also been research done on the problems in software development that have challenged corporate level programmers the most within the past few decades and a prediction of

obstacles that may rise in the future. In a study conducted by Manish Anand, a computer science practitioner with over 10 years of experience in Fortune 500 companies such as Nokia, Adobe, and Microsoft, Anand was able to conduct a meta analysis of various research articles evaluating the business aspect of this field and was eventually able to deduce which factors caused software applications to be the most inefficient or nonfunctional. For instance, Anand articulated that in the 1970s to 2000s, the biggest issues with software development had to be "budgeting," since many companies within the meta analysis didn't have enough resources to buy server rooms to hold data or to invest in marketing strategies (Anand). From the current time period, 2000 to 2020, the most critical problems faced by computer programmers today are code design (how the code is structured within the application) and how developers approach to solve a specific coding issue. Finally, Anand was also able to predict what critical issues would impede programmers in the future by evaluating the trends in which the software industry is lacking today. According to Anand, in the future, developers will face issues with "developing large-scale systems," collaborating effectively with their programming peers on different components of the apps, managing crowded workplaces, and thriving under high maintenance costs (Anand). To ensure that the readers weren't left wondering how to solve these obstacles, Anand also provided some of the solutions to the obstacles found above. For instance, Anand emphasizes how individuals can improve their ability to code by attending seminars and coding workshops in their spare time, by conducting fundraisers and product demos to acquire funding, and by participating in team bonding activities to ensure that every team member is comfortable with working together during development (Anand).

Apart from the business/corporate aspect of computer science, research has also been conducted in the specific fields of software development, such as mobile application

development. According to a survey conducted by Mona Joorabchi, Ph. D. holder in software engineering from the University of British Columbia, 188 industry professionals had revealed that a major challenge associated with mobile application development consisted mainly of cross platform integration, or the process in which an application is available on multiple platforms (for example, creating an application that is available on both iOS and Android posed as a challenge). For instance, "76% of survey participants see the existence of multiple mobile platforms as a challenge" since it fragments their product and creates double the work (Joorabchi). Furthermore, "each mobile platform is different with regard to the user interface, user experience, and human computer interaction," implying that developers had to start from scratch whenever they migrated to different platforms/devices (Joorabchi).

Research Gap

Although these articles were great in that it identified which obstacles in software development hindered industry level programmers and university students the most, whether that had to do with issues with financing, working with one's peers on programming projects, or marketing their software product, these articles neglected to address the teenage population, or the ages from 13 - 17. Especially since this age group represents the future of computer scientists throughout the nation and contains a large number of programmers who have currently released products to the public today, it is vital that this gap should be addressed, hence leading me to the question: which aspects of software development are the most difficult for teenage programmers, and how can they overcome these challenges. With the completion of this study, data will be revealed on whether the same problems that hinder industry level programmers persist for the teenage age range or if there are unique problems due to the subjects' age group. The overarching goals of this research would be to allow teenagers to prepare for the common pitfalls

a majority of software developers don't account for when starting their developing process, hopefully allowing them to create even more functional and successful apps in the future.

Methodology

The study design was broken up into 2 parts: a quantitative survey, which would reveal what the majority of teenagers thought were obstacles that limited the functionality and usage of their applications, and a qualitative interview conducted on industry level professionals, which would reveal proposed solutions for teenagers to implement if they ever got stuck during their programming projects. This particular design was chosen due to 2 main reasons. In terms of the survey, this method of measurement best aligns with the research purpose since it collects a wide variety of responses, allowing a generalizable compilation of the common pitfalls faced by teenage programmers to be clearly articulated . In terms of the interview aspect, this method aligned with the research's purpose since the advice/proposed solutions given by the interviewees could then be used by teenage programmers to get unstuck from any complications within their software creation. Now that the basic design of the research is clearly articulated, a deeper investigation of these research methods can be conducted.

Survey Design

Throughout the electronic survey, respondents were asked a series of free response questions which asked whether the respondent had faced a specific obstacle with software development in the past and why it was so difficult to overcome. For instance, respondents were asked whether they had issues with brainstorming unique purposes for their applications (additional survey questions located in Appendix A). In terms of where these survey questions had originated from, the majority of questions located within this method were adaptations of the same questions asked by researchers in the literature review. For instance, Mona Joorabchi had

asked 188 software professionals whether they had issues with creating software for multiple devices, finding time for development, or brainstorming purposes for their applications. Since this study was focused on finding the common obstacles of teenage developers, it only made sense to readapt some of these same questions to understand whether the issues of teenage programmers differed or were similar to those faced by industry level professionals.

Survey Population

In terms of the survey's population, the school district in which the researcher resided in was the population of interest since the district encapsules a large diversity of students of different races, sexualities, income, and ages, allowing this population in particular to be a great model for the current software development workforce. To find the specific population of teenage software developers within this area, students from computer science courses within the district were inquired since this method ensured that the respondents had prior background knowledge of computer science and application development. However, since this age group fell below the age of 18, it was required that these survey respondents had signed consent forms identifying that the survey respondents identity and responses will remain anonymous and the research itself would be conducted in a harmless fashion, which was approved by the District's Institutional Review Board.

Data Analysis

After the initial data was collected, there needed to be a form of analysis which would interpret these common obstacles and deduce how it affected their success as a developer. To do so, a series of comparative analyses were conducted between whether the developer was able to create a functional software application (or an application that has functional buttons, images, and UI elements) and the most prevalent obstacle for that developer. To analyze these trends,

google sheets was the mode of analysis since it allowed the researcher to create tables and graphs which could be easily compared with the use of correlation coefficients or descriptive statistics.

Interviews

After these thorough analyses were conducted, it was time to move onto the second part of the research design, the interview process. After selecting several industry level software developers, a Vice President from the I.T. department of tailored brands, one of the biggest fashion retailers in America, the C.E.O. of Smart Sites Inc, an organization devoted to helping the community with the creation of tech projects, and the I.T. Manager of Amazon Services (all acquired through the use of family connections), each industry professional would be asked to give their interpretation of the teenager's most common issues that were revealed with the preliminary surveys and what they would do to overcome them as professionals. For instance, if an interviewee was asked "what advice would you give to a teenage programmer having trouble learning how to code," the respondent would articulate resources and tutorials available for the teenage programmer to access. Finally, to actually record this data, a transcript was created that contained each interviewees individual responses to the questions indicated above (Appendix B).

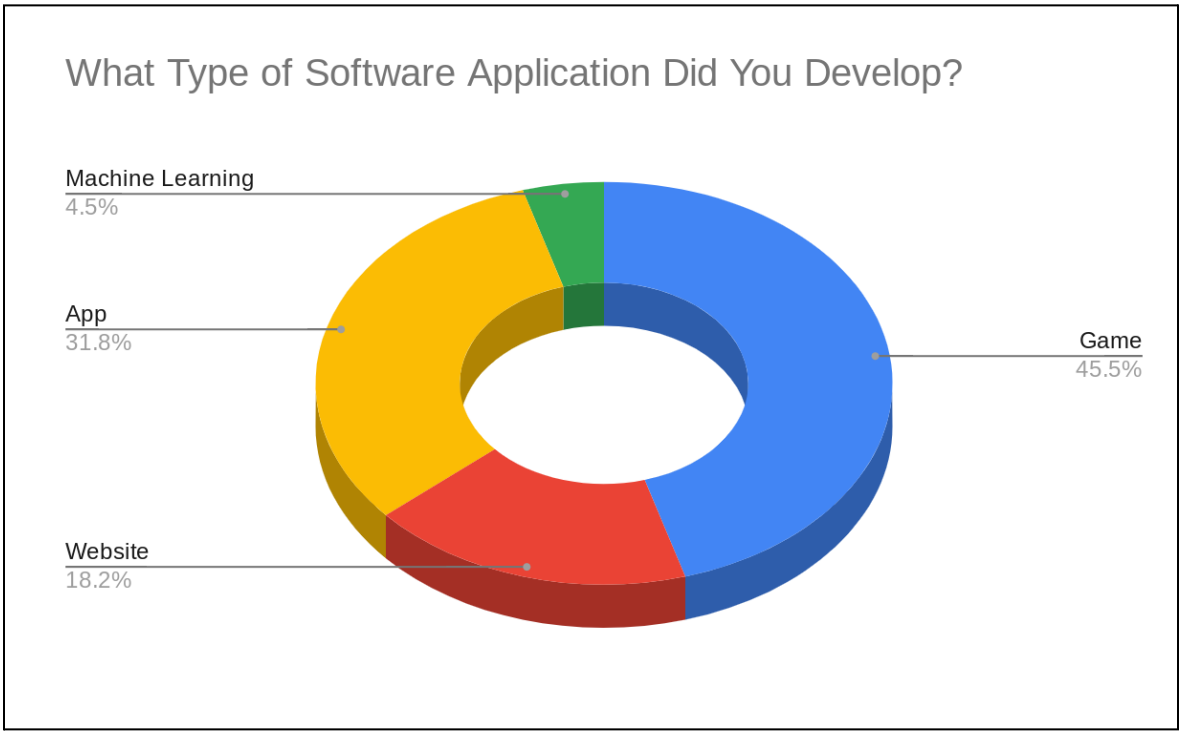
By conducting these forms of analysis, the common issues of teenage programmers will become more clear and the steps to eradicate these issues will be displayed, allowing students to prepare in advance for their coding journeys.

Results/Findings

After conducting surveys on 29 teenage programmers throughout the researcher's school district, it was clear that the level of inexperience faced by these early software developers had greatly affected the success of their software product. To start, this survey had reached a wide variety of teenage programmers who coded a diverse set of programs. As seen in figure 1, 45.5%

of respondents were game developers, 31.8% of respondents were app developers, 18.2% of respondents were web developers, and 4.5% of respondents had developed their own machine learning Algorithms.

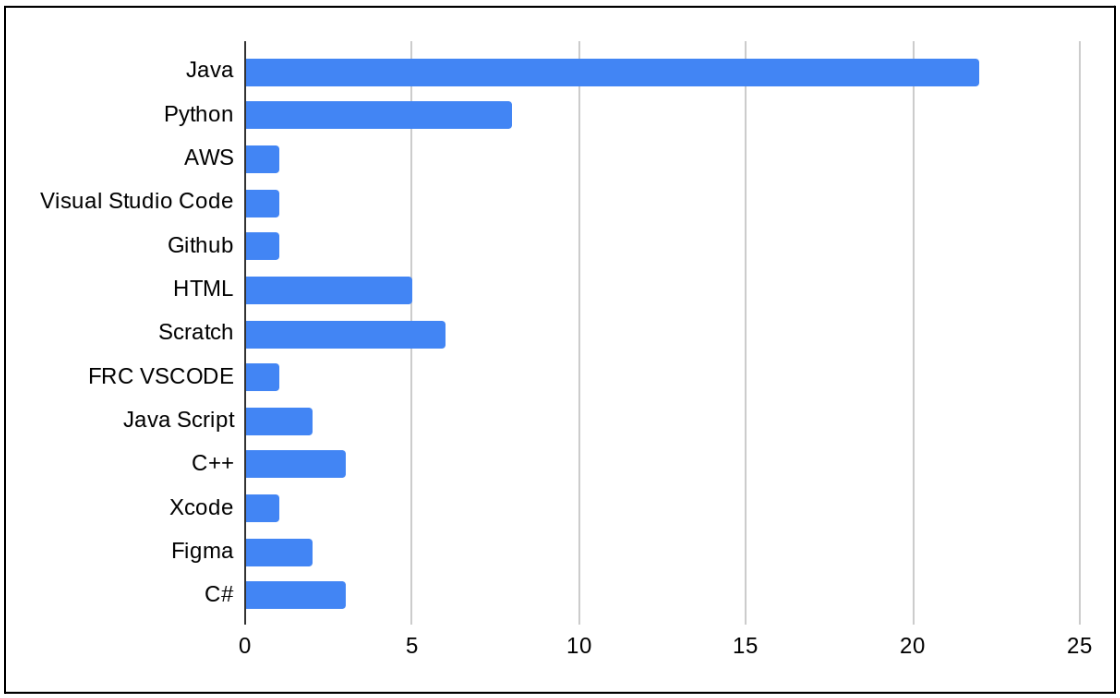
Figure 1: *Percent Distributions of the Types of Software Teenagers Decided to Create*



This data has shown that coding at the teenage level is quite similar to the industry level in some aspects. For instance, there was diversity in the type of software programs that each developer created, and the products that were created were rather complex, such as apps, games, and machine learning. To further evaluate the skill set of each developer, respondents were also asked which coding language they normally used during their software creation (Note: A Coding/Programming language is a language that computers can understand. For example, if someone were to create a website, they would need to code in HTML since this is the specific machine language that will be used to construct these complex webpages). The data represented

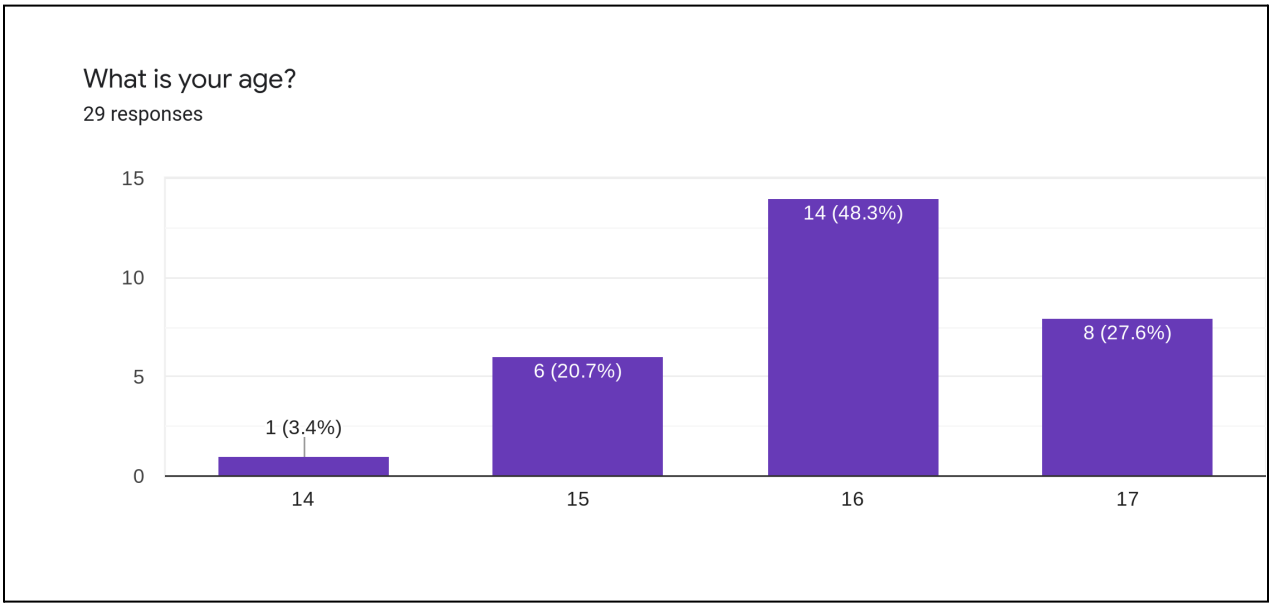
in figure 2 suggests that Java was the most popular language, Python was the second most popular, and Scratch was the third most popular

Figure 2: *What Programming Languages do you Create Software Applications With?*



In terms of the ages of these teenage programmers, the data revealed in figure 3 had found that 3.4% of respondents were 14, 20.7% were 15, 48.3% were 14, and 27.6% were 17, articulating that a wide range of teenagers were accounted for in this study.

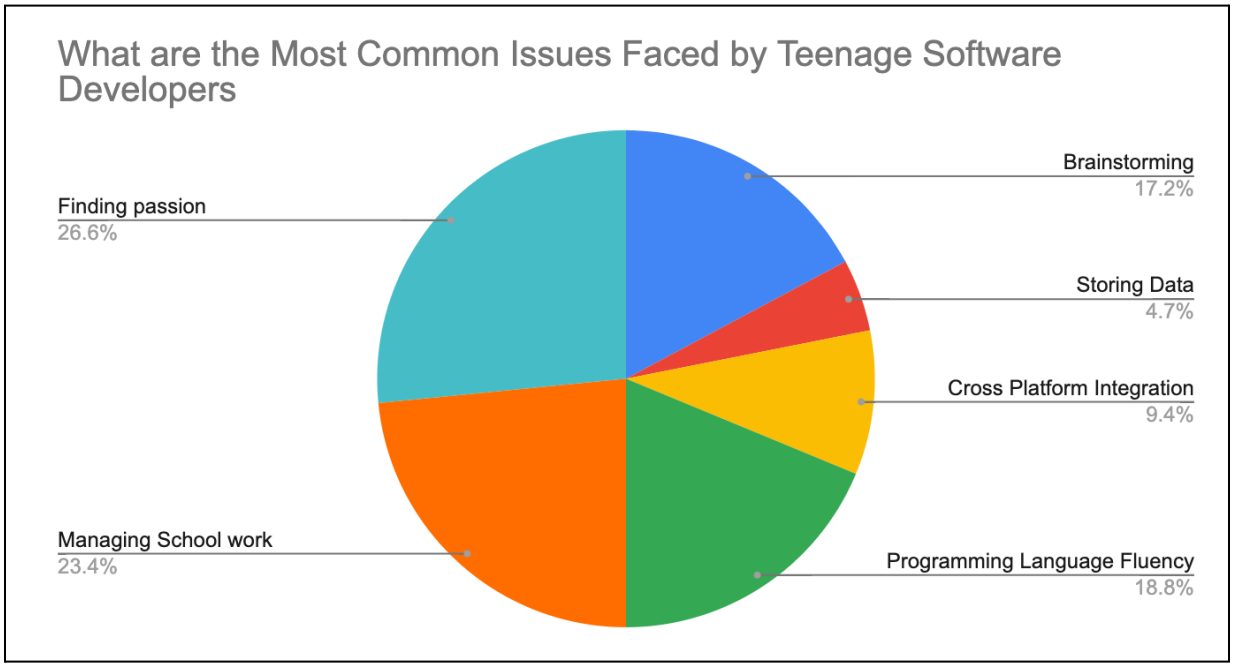
Figure 3: *Ages of Respondents*



Now that the various backgrounds of these survey respondents have been clearly articulated, we can dive deeper into the common issues faced by teenage programmers during software development.

After asking a series of yes, no and free response questions indicating whether a specific obstacle such as brainstorming a purpose for an app, being proficient in the coding language used to develop that app, or finding passion to create that app had impacted the functionality of a programmer's software product (Appendix A), the most influential obstacles faced by this age group were revealed. As seen in figure 4, the most common issues faced by teenage programmers (ranked from highest prevalence to lowest prevalence) were finding passion to program, managing school work, programming language fluency, brainstorming, cross platform integration, and storing data.

Figure 4: *Percentage Distributions of the Common Pitfalls in Teenage Programming*

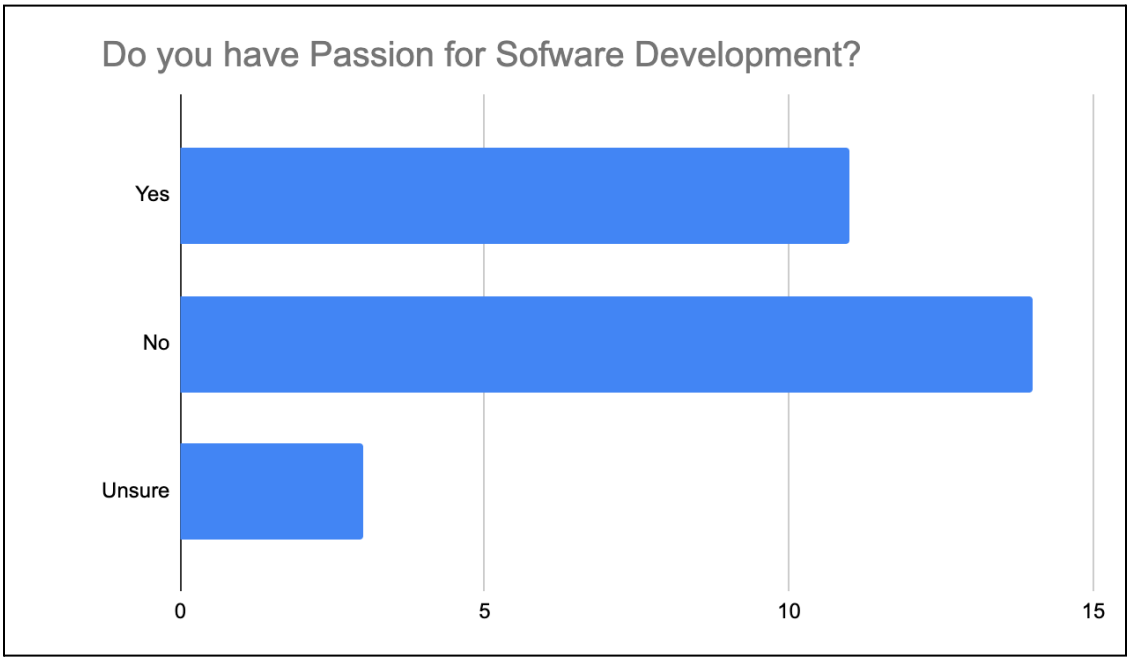


To truly understand which obstacles hindered the functionality and coding processes of teenagers the most, each obstacle will be evaluated in a more focussed lens.

Finding Passion/Interest to Continue Development

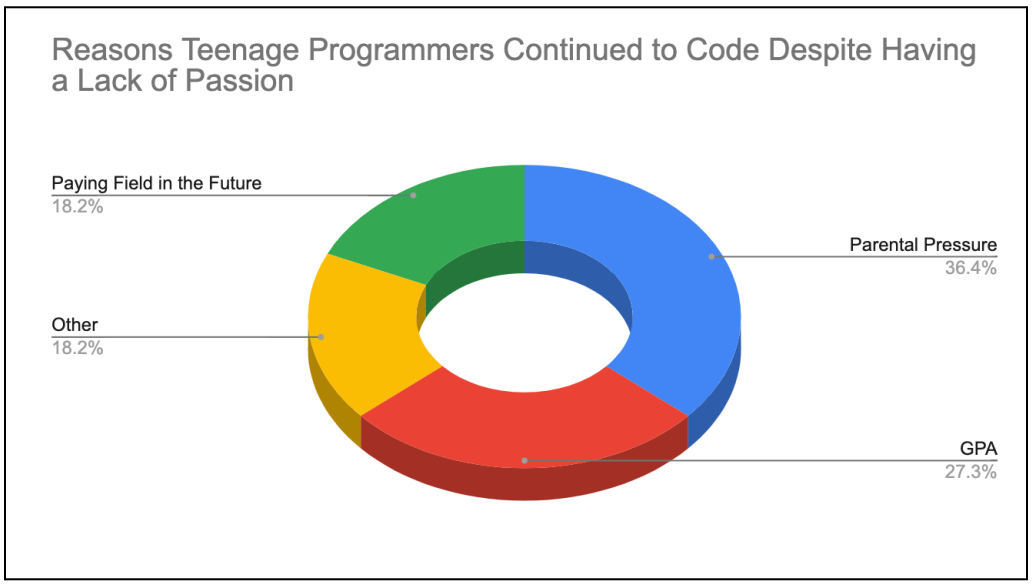
To evaluate whether a developer's passion for computer science impacted their work, respondents were asked to describe why they started their development journey and whether or not they were truly interested in creating apps for the general public. Surprisingly, as seen in figure 5, 50% of respondents had stated that they had no passion for software development, and had a hard time finding the motivation to continue creating their software projects.

Figure 5: *Number of Teenagers Who Had an Interest for Coding*



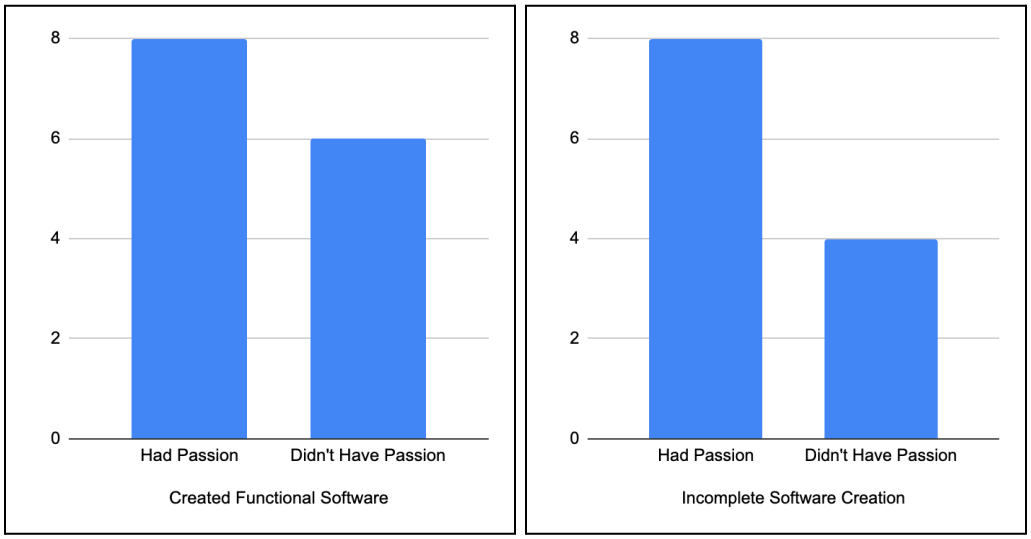
This trend was surprising because if teenagers had no interest in learning how to code, why would they start in the first place? When asked for the reason teenagers decided to take on their software projects, respondents had indicated that they had only done computer science to help with their G.P.A or were forced by family members to pursue computer science.

Figure 6: *Reasons Teenngaers Coded Despite Having no Interest for Computer Science*



Although this lack of passion had definitely existed within this data set, it surprisingly did not affect the programmers ability to create a fully functional software. By definition, a software was considered functional if it contained usable buttons, graphics, and could be used as intended on a specific device such as a phone or a laptop, and as seen in figure 7, for those who created a functional software, there was a close split between those who had passion and those who didn't, and on the side where the developer was not able to create their application, the majority of respondents had interest for programming, proving counterintuitive.

Figure 7: *Comparisons Between Passion and App Functionality*

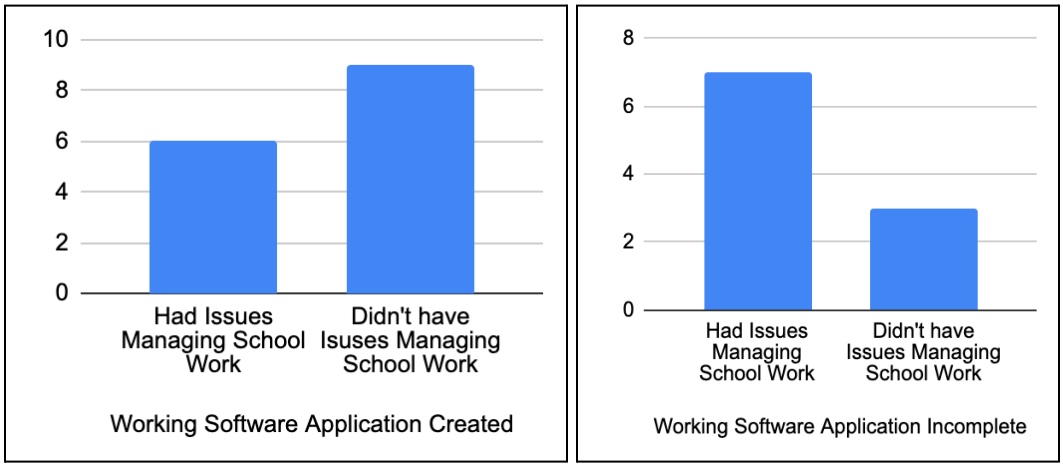


Managing School Work and Development

The second most prominent issue in teenager's software development had been managing their school work, extracurricular activities, and other time commitments with their programming. For example, 52% of respondents had stated that they had difficulties finding time to create their software applications. This issue is especially not surprising considering the amount of school work teenagers are given on a weekly basis. According to the Washington Post, from the years 2018 through 2022, students were given on average 2.7 hours of homework per week night (Strauss). On top of coming back home after long, grueling hours of school work and

extracurricular activities, students would then have to complete their homework, chores, and finally scrap up some time just to go to sleep. As a result, it can be seen why adding software development to the mix may impact teenagers' ability to manage time while effectively managing development precision. This trend in particular is revealed by the graphs shown below which compare students' abilities to create a fully functional software application and students' ability to manage their school work.

Figure 8: *Comparisons Between Time Management and App Functionality*



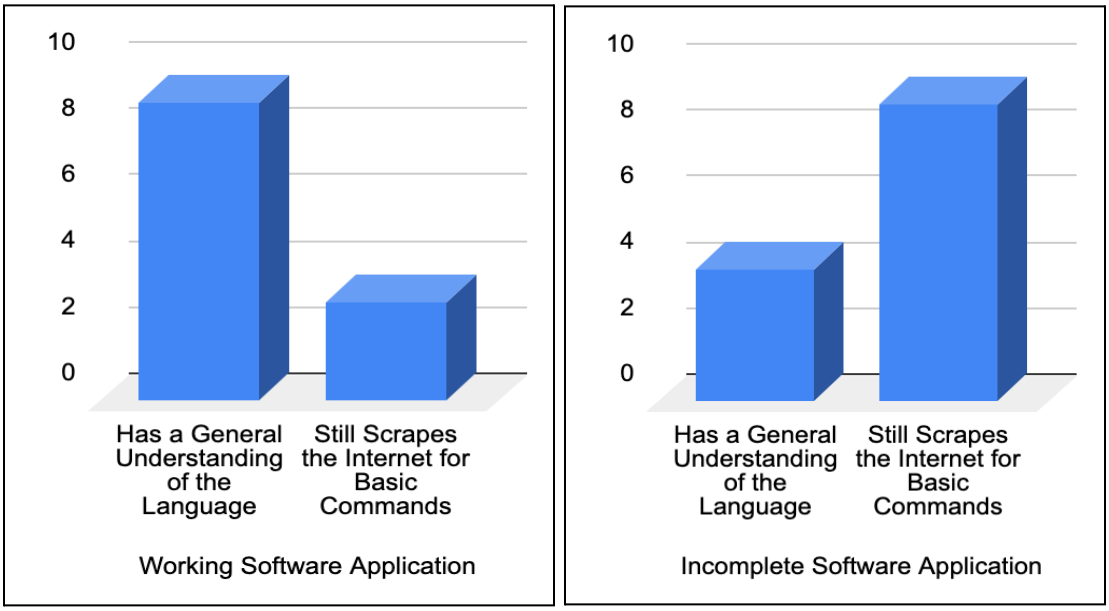
Of the students who had failed to create working software applications, 70% of those respondents also had issues managing their school work with their development. However, of the students that were able to create their apps as they had intended, the majority of respondents were those who could manage their development time effectively. In order for teenage programmers to be successful in their software ventures, it is clear that programmers must be mindful and organized with their time management.

Programming Language Fluency

Another issue throughout teenagers' software development journey had been obtaining good programming language fluency, or the ability to write code effortlessly with minimal use of outside resources. For example, 12 out of the 29 respondents had stated that they were still scraping the internet for basic commands such as how to allow the user to interact with buttons or how to display images on the software's home page. To see whether this issue had impacted app functionality, another comparative analysis was conducted between app functionality and programming language fluency. As seen in Figure 9, for those who had failed to create a functional software application, 69% of programmers were still scraping the internet for basic syntax commands. However, for respondents who did manage to create a fully functional

software application, only 25% had come from programmers who had little experience with the programming language while the other 75% came from respondents who possessed a large understanding of how to code.

Figure 9: *Comparisons Between Programming Language Fluency and App Functionality*

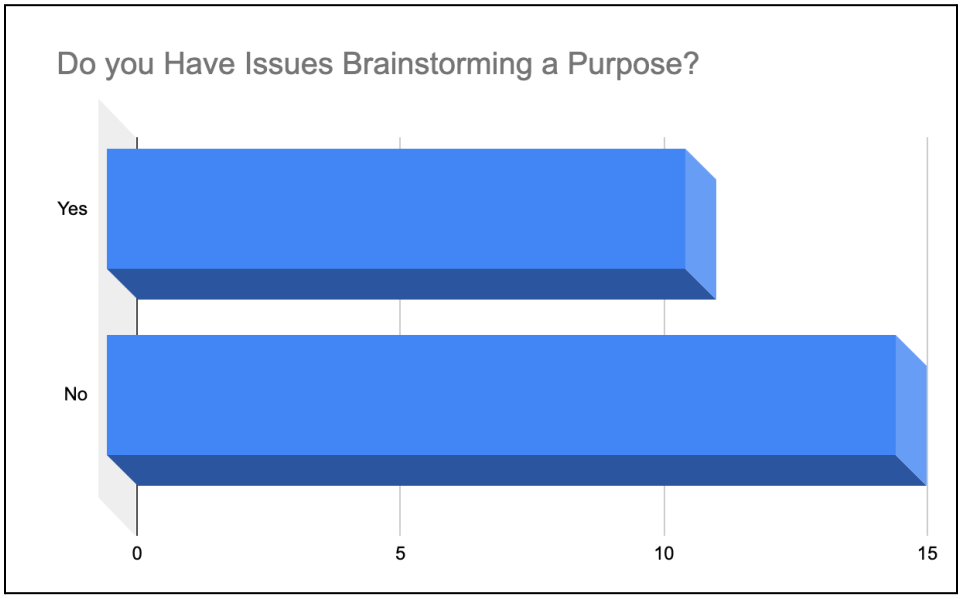


As a result, it was clear that a programmer's fluency with the language was definitely a factor in a programmer's success rate.

Brainstorming

Furthermore, respondents also had a considerably difficult time coming up with unique, out of the box ideas to work on while starting their software development processes. For example, 42% of survey respondents had stated that standing out from the millions of apps readily downloadable on platforms such as the app store or play store was hindered the most by their inability to brainstorm creative app concepts.

Figure 11: *Amount of Students Who Have Trouble Brainstorming a Purpose or Function for their App*



According to the Harvard Business Review, a critical task needed to make a product successful is to understand the strengths and shortcomings of rival products and identify why an innovator's product is unique or adds value to the current market (Steenburgh). For example, if a small company were to create a software application which could allow users to post pictures, message each other, and view each other's stories, they would be crushed by rival competition such as facebook or instagram. In the same way, these respondents had also faced the same issue of coming up with a unique idea that would set them apart in the growing computer science market. As a result, it is necessaryl that future teenage programmers are aware of this phenomenon.

Cross Platform Integration

The last issue faced by teenage programmers was cross platform integration, or creating software that was functional on multiple platforms. For example, if a respondent had created a functional app for iPhones, they tended to have difficulties launching the same app on a platform

like Android and vice versa. As indicated by Mona Joorbachi, the reason this integration is so difficult to implement is due to the fact that programmers will have to account for different screen sizes, different user interfaces, and would have to learn a whole different coding language to create the same app on 2 different devices (Joorabchi). Although it is clear why such an issue would be hard to overcome, this obstacle was surprisingly only endured by 15% of the respondent population, proving not as influential as one may assume.

Advice/Interview Results

After completion of the survey results, it became clear that teenagers had encountered some development issues that were unique to their own age groups. For example, a big issue faced by teenagers was managing to find time to code after a long day of school work, extra curricular activities, and relationships, a factor that is not so influential in the professional field where a programmer's only job is to develop apps in corporate offices. Since solutions to some of the problems listed above do not readily exist in the current research community, interviews were conducted on industry-level professionals to see what strategies teenagers could implement to break free from these obstacles and thrive in their software creation.

After interviewing 3 industry level professionals, it became clear what strategies teenage programmers would need to implement into their own coding process to better the functionality/usage of their apps (Note: some interview responses were repetitive so the least redundant response would be chosen to display within this study). As stated by the C.E.O. of Smart Sites Inc, to efficiently manage development time with teenagers' school work, students should "create time tables and priority lists which will help them stay on top of their assignments and cut out any unnecessary tasks from their work day," hence allowing programmers to find the time to program with minimal stress.

Another task that students should do to become more successful in their programming endeavors is to practice their programming language methodically to improve fluency. As stated by the manager of Amazon, students should utilize online resources, such as YouTube or Stack Overflow, which Contain Sample Coding Exercises and tutorials that will allow teenagers to become more proficient in their coding languages.

In terms of brainstorming an idea/purpose for their software application, the advice given to these teenagers would be to "find the processes of everyday life which are done manually and to try [automating] them." This process of brainstorming is especially powerful since it is seen all throughout software today. Take doordash as an example, they took the manual process of ordering food in store and converted it to a way where consumers can order straight from their app, giving them a stellar purpose.

Furthermore, for teenagers struggling with launching their applications on multiple platforms, such as an application on both iOS and Android, the biggest advice given to teenagers by the VP of Tailored Brands would be to "find team members that can code in different coding platforms," hence reducing the development time and workload by half. For example, in the case of iOS apps, if one person was to code in Xcode (a coding service for creating iPhone apps), the other team member should be able to code in android studio (a coding service for creating Android apps) to allow the application to be distributed to multiple platforms in an efficient manner.

With the conclusion of these interviews, it became clear what strategies teenage programmers should implement to break free from any halts in development they may face in the future.

Conclusion/Evaluation

After the completion of this study, it became apparent that some of the same obstacles faced by corporate level computer programmers carried on to this particular age group. For example, brainstorming a concrete purpose for an application, creating a software that runs smoothly on different devices, and finding motivation/passion to create code have all been issues that were both apparent within the literature review and the results of the surveys given to teenagers. However, there were also some unique obstacles that had only impacted this particular age group. For instance, managing development time with school work, extracurricular activities, and relationships was especially the challenge due to the many burdens of being a high school student.

Furthermore, by conducting comparative analyses on which specific obstacle the teenager had faced and whether this had impacted the functionality of the application, it became more clear the extent to which the developers' inexperience hindered their success, such as how programmers who had a narrow understanding of programming were more likely to be unsuccessful in creating a functional app within the study. Finally, it was also uncovered that some issues in the corporate world were not as applicable to teenagers. For instance, although industry level programmers struggled with "crowded workplaces," low project sponsorship, and miscommunication amongst peers, these issues were not as relatable on the teenage level since teenagers rather created their applications in the comforts of their homes rather than a full scale office building (Anand).

Limitations

Although this research had successfully found the common issues faced by teenage developers and the solution to these obstacles, the population size would've been the biggest

limiting factor. For example, the 29 respondent samples that were collected were all taken from the researcher's school district, one of the most educated districts in the United States. As a result, this sample may not be as representative or universally applicable to all teenage programmers throughout the United States, who may not have the same access to computer science courses or financial resources as students within this district.

Another limitation of this research was that most of the respondents had coded in the java programming language rather than an even split between multiple languages. In the real programming world, different languages have different issues, whether that relates to developing a strong UI (the graphical elements on the screen), UX (how intuitive the application is to use), or syntax. For example, the issues of an app developer, such as cross platform integration, may not be the same as a website developer that worries more about graphic design. As a result, the common problems faced by this subset of people may not relate to all programming languages, but the problems found above may relate more to programmers who coded in Java.

Future Studies

To mitigate the limitations listed above, researchers should first try to get more representative samples from different cities and states to make the results more generalizable. Furthermore, future studies should also try to get an equal number of respondents that were fluent in a variety programming language to get a more varied/applicable result to represent the diverse coding applications situated in the programming workforce (mobile apps, websites, etc). Finally, future researchers could also demo the respondents' applications for themselves to see which part of the software was the most nonfunctional and the reason behind these lapses in functionality.

Implications

Regardless of any limitations found within this study, the research certainly has numerous implications. In the research aspect, this study adds to the current body of literature since it evaluated a different age group/population (teenagers aged 13-17) not previously studied in the field of software development. Furthermore, the study also allows researchers to realize that the struggles of programming may not be the same for every person in different age groups, but rather complex in their own fashion. For example, although teenagers struggle with some of the same issues industry level professionals do, such as brainstorming unique application purposes, staying motivated while developing, and creating applications for multiple platforms (Joorabchi), some challenges were unique to their age range, such as maintaining strong programming language fluency. In the developmental aspect, this study has proved beneficial since it gives the option for future teenage programmers to refer back to the advice given by the industry level professionals in case they are ever stuck within their coding endeavors, allowing teenage programmers' to become more successful in their programming journeys.

With the completion of this study, it has become clear that software development, no matter the age range, is a field that poses a lot of obstacles. However, if more efforts are concentrated in resolving these challenges and more attention/aid is given to the diverse age groups within this profession, more developers will be enabled to create stunning applications for the general public, allowing technology to further progress throughout society.

Works Cited

Anand, Manish K., and Vasudeva Varma. *Issues, Challenges and Opportunities for Research in Software Engineering*. 2008, http://dx.doi.org/.

Butler, Matthew, and Michael Morgan. "Learning Challenges Faced by Novice Programming Students Studying High Level and Low Feedback Concepts." *Proceedings of Ascilite Singapore 2007 ICT: Providing Choices for Learners and Learning*, Nanyang Technological University, 2007, pp. 99–107.

Joorabchi, Mona Erfani, et al. "Real Challenges in Mobile App Development." *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2013, pp. 15–24.

Mithas, Sunil, et al. *Artificial Intelligence and IT Professionals*. IEEE, 12 July 2019, https://deliverypdf.ssrn.com/delivery.php?ID=944003031017122110066006092081097099000056068039003000127072110072070120091006091118102007096001105013006095076124127030080091013019005026020127077002082024098089068076017096020109121004026017126007018066016083120093024096096110005117094111074089110&EXT=pdf&INDEX=TRUE.

Steenburgh, Thomas. "You Invented a Great New Product. Now, How Do You Sell It?" *How to Sell New Products*, Harvard Business Review, 23 Oct. 2018, https://hbr.org/2018/11/how-to-sell-new-products.

Strauss, Valerie. *Perspective | Does Homework Work When Kids Are Learning All Day at Home?* The Washington Post, 2 Sept. 2020, https://www.washingtonpost.com/education/2020/09/01/does-homework-work-when-kids-are-learning-all-day-home/.

Appendices

Appendix A: *Survey Questions That Were Distributed to Respondents*

1. What is your age?
2. What kind of software product did you create?
3. Have you taken computer science courses in the past?
4. What programming languages do you use to create software?
5. How many years of experience do you have with that particular language/platform?
6. Do you have a general understanding of your programming language or do you still scrape the internet for basic commands?
7. Do you have difficulties brainstorming a software application idea/purpose or was your task assigned to you?
8. Do you have difficulties storing data? For example, do you store data directly on to the user's device or do you store your data in a remote server?
9. Do you have issues making your software available across multiple devices or platforms?
10. Do you have issues marketing your software product?
11. Do you have passion for computer science or do external factors, such as parental advisory, impact your desire to code?
12. Are computer science courses in school helping you through your journey? If so, why or why not?
13. How many downloads/purchases/plays has your software received
14. Was your software application functional? For example, did the buttons, images, text, and graphics function as they were intended?

Appendix B: *Interview Questions and Responses*

Responses From the Vice President of Tailored Brands I.T.

1. What advice would you give to a teenage programmer who is struggling to manage their school work and extracurricular activities with their software development time?

   "Students should try their best to not procrastinate and squeeze in development time whenever they are not doing something essential."

2. What advice would you give to a teenage programmer who is having problems with learning their specific coding language. How can they improve programming language fluency?

   "Try to enroll in coding courses online to better your understanding. Shadowing an adult or professional may also help with fluency. It may also be useful to read the documentation."

3. What advice would you give to a developer who is having difficulty coming up with a purpose or topic for their app?

   "Teenage programmers should find the processes of everyday life which are done manually and try to automate them. For example, if no one decided that shopping online could be more efficient, we wouldn't see those platforms today."

4. What advice would you give to a developer who is having trouble launching their app on 2 different platforms. For example, an iOS developer trying to launch their app on Android.

   "If you want to create applications that are readily available on different platforms, you should separate the project members into 2 differen teams. Especially, you should be able to find team members that can code in different coding platforms."

Responses From the C.E.O of Smart Sites Inc

1. What advice would you give to a teenage programmer who is struggling to manage their school work and extracurricular activities with their software development time?

   "Students should try to create time tables and priority lists which will help them stay on top of their assignments and cut out any unnecessary tasks from their work day."

2. What advice would you give to a teenage programmer who is having problems with learning their specific coding language. How can they improve programming language fluency?

   "Try to watch YouTube tutorials on how to code since it is free and easily accessible. Also, it never hurts to try some programming projects on your own time. Try to also teach the content to your friends since teaching the content you know to others will aid in memory retention."

3. What advice would you give to a developer who is having difficulty coming up with a purpose or topic for their app?

   "Trace back through your everyday life and find the places that can be done electronically. Pick something you are actually devoted and interested in to make the purpose more meaningful."

4. What advice would you give to a developer who is having trouble launching their app on 2 different platforms. For example, an iOS developer trying to launch their app on Android.

   "Try to use frameworks like React Native that allow programmers to use a single programming language to code in 2 different platforms, rather than stressing out with learning 2 unique programs."

Responses From the Manager of Amazon Services

1. What advice would you give to a teenage programmer who is struggling to manage their school work and extracurricular activities with their software development time?

   "Teenagers should try to submit all of their assignments on time to allow more time in the future to work on more meaningful projects like software."

2. What advice would you give to a teenage programmer who is having problems with learning their specific coding language. How can they improve programming language fluency?

   "In order to learn a programming language to a greater understanding, teenagers can try and check out online programming resources such as W3 Schools, Stack Overflow, and YouTube to allow the programming concepts to settle within their skill set."

3. What advice would you give to a developer who is having difficulty coming up with a purpose or topic for their app?

   "Think of the apps that you use on a daily basis. Are there any ways that app is missing something, anything that could be added to its function that could give it a unique selling point?"

4. What advice would you give to a developer who is having trouble launching their app on 2 different platforms. For example, an iOS developer trying to launch their app on Android

   "The best way to launch multiple apps on different platforms would be to use project management to your likings. The leader of the project should clearly define which team is in charge of what and should clearly articulate the goal and timelines to their entire team."