The K-12 Learn-to-Code Movement is Leaving Current Graduates Behind: Status and a Case Study

Theodor Wyeld[1(✉)] and Minoru Nakayama[2]

[1] Flinders University, Adelaide, Australia
theodor.wyeld@flinders.edu.au
[2] Tokyo Institute of Technology, Tokyo, Japan

Abstract. There is a movement towards teaching children how to code. This is not a new idea. It has been promoted since the 1980s. It is only very recently, however, that the need to know how to code has become crucial. University-age students need to know how to code to achieve employment after graduation. Post-graduation jobs increasingly require some understanding of how programs work and are developed. Since the early 2000s there has been an increasing demand for graduating students with coding skills. They have attempted to meet this demand by teaching themselves some coding skills (such as for web apps). But, many report they struggle to get past the basics. The tool used in this study immersed students in code in a way that was both fun and leverage existing interests. An online 2D games editor was used for students to learn how to build an app for their mobile device. They interacted directly with the code base and created solutions to problems. Before and after questionnaires showed a shift in sentiment from a fear of coding to a better understanding of code.

Keywords: Learning to code · Learning to program · Code anxiety

1 Introduction

Coding is being promoted as a way to encourage children to think about computing as a creative outlet. It is beginning to be included in the K-12 curriculum in schools globally. Children's socialising on digital networks demands some fundamental understanding of the technology, how it works, and even how to produce for it. Their enthusiasm for creating digital media to share is only matched by the communities that support the activity. Children increasingly modify, create, design, illustrate, animate and tell stories using digital media. This is extended into programmable toys, robots and other devices (Kafai and Baurke 2016). But, this explosion in programming activity at an early age has found their older siblings – between 18 and 25 – playing catch-up. It is this later group that are now at university age and know that the sort of jobs available after graduation increasingly require some understanding of programming; either coding or project managing. Their anxiety about gaining the requisite skills is real, yet largely ignored in curricular outside the traditional university computer science courses (Brooks 1986, 1995). This paper discusses the implications of this and, in brief, a project which attempted to address the deficit by introducing students to coding in a non-threatening albeit realistic development environment. It used a before and after questionnaire to capture any shift in sentiment towards learning to code over the term of the project. It found that, it is possible to relieve some of the anxiety experienced by exposing students to coding from IT and non-traditional IT backgrounds alike. While this reports on only one method of introducing students to coding, the need to provide for an authentic environment whereby students could hold an expectation of succeeding was paramount to its success.

1.1 The Need to Learn to Code

The idea that children should be learning how to code from an early age is not new. Since the time of accessible computing in the late 1970s and early 80s Papert (1980) was advocating its role in the curriculum. Papert (1980) saw computer systems as a new medium for learning. He saw it as a shift away from passive knowledge acquisition to actively promoting individual creativity. The computer provided a logical environment for exploring creativity at a new level. Papert's work with Piaget (see Ackermann 2001) exposed him to different ways of approaching how children learn. Papert (1980) felt a need to shift education from instruction to natural-inquiry; children should be free to construct their own knowledge. His Logo programming language was designed to compliment a child's egocentric position and movement by developing their nascent theories about geometry. He was not simply suggesting that children should learn coding as another language. Rather, he saw computer programming as a way to restructure maths and grammar such that it accommodates the child's natural tendencies to learn as a creative pursuit. Learning to code is in this sense not an end in itself but a means to explore the role of computing in modern society. He subscribed to the constructionist philosophy that computers are merely an epistemological tool to inspire and express creative ideas. He claimed, like Minsky (1988) proposed, the computer's ability to simulate reality, and processes outside of reality, mirrors human thought processes – it provides insights into our own ways of thinking and changes them in the process. Although many schools attempted to incorporate at least some of Papert's ideas in the 1980s, few persisted. It is only recently that there has been a renewed push for coding to be added to the curriculum in early education. However, there is also a burgeoning cohort of university-aged students who need to know how to code as a pre-condition of employment post-graduation.

1.2 The Rise of the Need to Code

The rise of high-tech innovation hubs in places like Silicon Valley, Bangalore, Shenzen (Startup Warehouse), Dublin (Silicon Docks), Tokyo, Taipei, and Seoul, led to a shortage of coders after the early 2000s. The growth in demand for coders was more than 30% between 2007 and 2012, and is predicted to grow at 22% between 2012–2022. Most recently, a group has formed called the 'learn to code' movement (see code.org; codeconquest; codecademy; meetup.com; codelikeagirl.org; Silicon Valley Coding School; Disrupting Engineering Education). They are pushing for coding to be taught in schools. They have support from a wide variety of high-profile advocates such as the mayor of NY, Michael Bloomberg, Microsoft founder Bill Gates, Facebook founder Mark Zuckerberg, and President Obama, pushing for legislation to include computer science in every public-school curriculum.

The rise in the perceived need to learn how to code is based on the success of entrepreneurs such as Elon Musk and Mark Zuckerberg. They are often cited as extraordinary success stories that others should try to emulate. But, some argue coding is not the new literacy – coding is simply the extension of existing skills in a new framework (Farag 2016). The framework is one usually reserved for those who work in it directly. But, increasingly, nearly everything we interact with has some code base. And, knowing how to code helps us understand how it works (see Petzold 2000). But, even if you don't need to know how a car works to drive it, it does help if you have any chance of understanding how to fix it when it breaks down. In this sense, knowing how code works also helps you know what questions to ask even if only to direct others to do the coding for you.

1.3 How Graduating Students are Teaching Themselves Code

Many contemporary graduates are coming to the realisation that in order to secure employment they need to know how to code (see Althoff 2017). Some learn bootstrap (a front end responsive framework for building websites – WordPress was built on it). CSS is another place they start learning how front-end code affects websites. Websites in general are a good place to start to learn coding as it is more about layout and structure than functionality *per se*. Other packages include: SQL, JavaScript, Ruby on Rails, HTML, Sass, Stylus, Meteor (a full stack JavaScript framework). For example, many of the blogs, news sites and eCommerce sites that the current graduates access are web applications. Content Management Systems (CMS) like Wordpress and Magento (which does eCommerce) make it easy to edit and build these sorts of web apps. However, the limits of a CMS are quickly reached. More advanced design or technical features require a more advanced understanding of the underlying code. JavaScript is a common method for adding the functionality.

1.4 Getting Past the Basics

The most common difficulty for beginners is getting past the basics. They may have learned some code but don't really know how to go about building their own programs from scratch. They might understand the theory but can't put it into code. They understand some core concepts but are not sure about where and when to use them. They don't know how to combine the various core concepts such as loops, arrays and variables. They feel lost after completing the basics. This can be caused by a number of factors. Often the learning environment is not the same as the actual developing environment. Hence, getting started in a real environment helps facilitate some self-guidance. The learning environment is often supported with lots of hints and technical advice. But, when it comes time to develop the code from scratch, none of that is there. Knowing the correct syntax does not guarantee they understand the underlying concepts. Too much help makes the programming look artificially easy. When students are left to do their own coding they may find they don't have a deep enough understanding of how to construct it.

Whether working from an IDE (Integrated Development Environment - compiler) or the command line, a real coding environment is more conducive to deep learning than simply writing pseudo code. It is actually from the active debugging and error fixing that learning begins (see Bonwell and Eison 1991; Prince 2004 on active learning). Copying and pasting other peoples' code is useful, but only if the learner understands what it does and how to fix it when it doesn't work.

1.5 What to Focus on

It is not enough to simply know how to code. Many students that have completed a course in coding do not feel they can actually write a program from scratch. They may understand these concepts and syntax but not how to put them together to solve a problem by building an actual program. Perhaps the learning environment had too much support. When this support is taken away, they feel lost. What matters is knowing what the problem is and how to implement a solution for it using code. The focus here should be on solving the problem rather than learning to code *per se* (Shermer 2005).

1.6 Coding Changes a Student's Way of Thinking

Coding should be fun. It should let students make things and build things. But it requires a lot of patience and hours of practice. Coding trains students to think in a certain way – but it is very narrow. Coding should not be seen as a goal in itself. It should instead be seen as a tool for solving problems (see comments by Peter Norvig, director of research at Google, norvig.com). Students are more likely to commit to learning to code if they have a problem they want to solve than learning to code for its own sake. But, while students can learn to code quickly, this doesn't make them a coder or engineer – it generally takes about 10 years to ever become expert in anything. However, even knowing the rudimentary basics of coding gives students a headstart in a world that relies on coding. In some ways, it's more important to know how coders think than actually being a coder.

1.7 Which Languages are in Greatest Demand?

The highest salaries paid to coders by language (usually linked to demand for knowledge of a particular language or shortage of those with that language) can be ranked as: 'Ruby on Rails', Python, C++, iOS, JavaScript, Java, C, PHP and SQL. But, one needs to be able to switch languages as the demand changes. And, one needs to know several languages. For example, even though Ruby and Python might appear to be the highest paying positions, most job ads list C, SQL and Java as also necessary. In fact, Ruby is not listed as often, meaning that, while Ruby may be highly paid, there are not many positions when compared to raw C. JavaScript and Python are often also listed (see Fig. 1). This is the case for established firms. For start-ups, the story is different (see Kelleher and Pausch 2005).

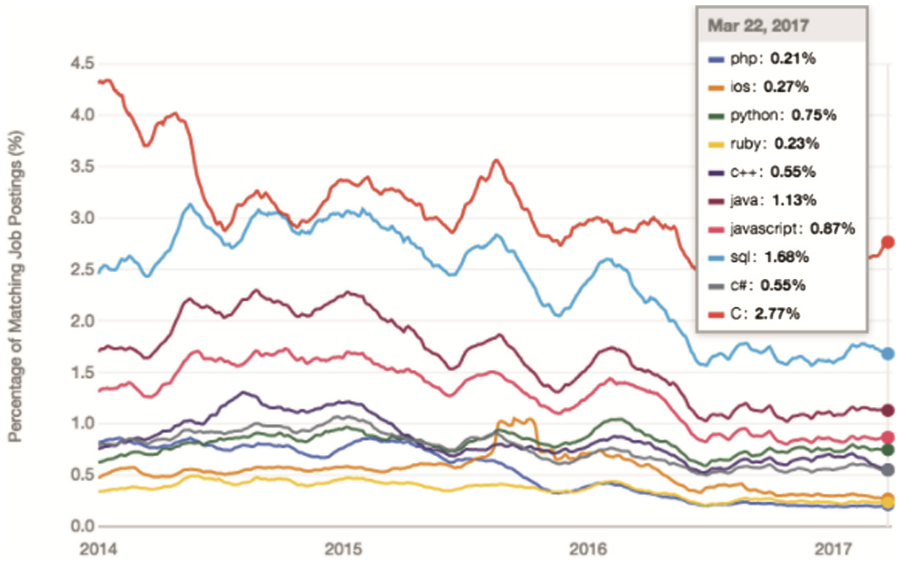

Fig. 1. Distribution of code languages as advertised between 2014 and 2017 (codementor.io, accessed 2017).

While ‘Ruby on Rails’ was the favoured language in 2015, in 2017 it tends to be JavaScript, Python and Java. These are easy to learn languages that can be used to produce prototypes quickly (see Fig. 2).

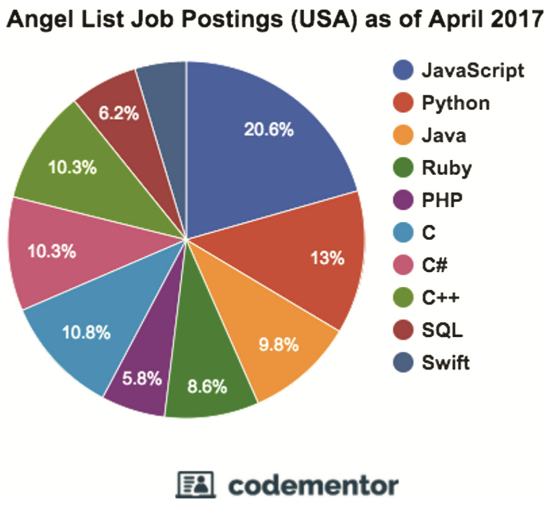

Fig. 2. Distribution of coder language preferences in startups for 2017 (codementor.io, accessed 2017).

In order to get ready for industry post-graduation, beginner learners need to know how to use repository or version control resources like GitHub. There they can post their projects and see what others are doing and share code. Employers often look at a potential employees history of programming activity. GitHub is one of the places they expect to find this history. Between StackOverflow and GitHub there are many resources for beginning learners.

1.8 The Future of Languages

The question of what language to learn now presents a dilemma. Languages fall into and out of favour depending on support and the sorts of apps needing to be developed. The shift from workstations in the 80s, to internet in the 90s, and mobile devices since the 2000s, has seen a lot of change in which languages are used. For example, now that JavaScript can be used for back-end development, and because it is relatively easy to learn, it is becoming increasingly popular. And, because some of the biggest sites were built in 'Ruby on Rails' (which is also easy to learn), they continue to need developers to maintain them (such as Airbnb, Twitch, Hulu and so on). However, Node.js is starting to take over from Rails, just as Rails took over from Python's Django, and so more growth in JavaScript is expected. By contrast, Python is the preferred language for educators and scientists. Hence, there will be continued growth in this language. It is an easy to learn and flexible language so it has wide appeal. As Android has captured such a large proportion of the mobile market, the demand for Java programmers has increased. Java is scalable, stable and has such an enormous collection of feature libraries that almost anything can be developed in it. It is also relatively easy to learn compared to C++. Therefore, Java will continue to grow over the years (see Krill 2014). On the other hand, where Objective-C was once the preferred developer environment for iOS, Swift has now largely taken over. Nonetheless, Objective-C and Swift are very similar, therefore, knowing how to code in both is an advantage. C is a low-level (machine code) language. As such, it is vital for programming at the operating system level. This is quite difficult to understand programming, conceptually and practically. Hence, there are few developers in this domain. But, it also means there are high-paid jobs to be had. However, demand can vary across the years, so it is important to be proficient in more than one language. An example of a language which fell out of favour only to come back is SQL. Other RDMS languages took over from SQL for a while on large database platforms, such as Hadoop, Spark and Cassandra. But, as the databases grew even larger, the non-SQL query languages used started to underperform. As such, SQL has made a come-back. Google's BigQuery now uses SQL. Spark also uses an SQL module for some of its products (ClustrixDB, DeepSQL, MemSQL, and VoltDB). SQL is important for managing and analysing big data. It is also an easy scripting language to learn. It is important to know how to leverage the access to big databases that SQL affords for front-end apps such as data-driven websites. Although C++ is much harder to learn than some of the high-level languages, its power and functionality make it a very important language in high demand. Along with C++, C# is another popular language. In particular, it is used in the gaming industry (Unity 3D). Once learned, it seems to generate its own following; that is, developers are loyal to developing everything they can in it. This means it is well supported by developer communities, but one has to be careful that they do not loose sight of the need to know other languages also.

2 The Case Study Project

From the literature on how best to facilitate learning to program, clearly it is important to set up an authentic developer environment with a 'real' project which both challenges students but also sets achievable goals. As such, our project was focused on supporting teams of students. They begin by accessing some preformed, working, code which they can customise to achieve their own goals. This gives them some autonomy to explore, create, edit, make mistakes and produce, all within a risk-free environment.

From the literature it seems text-based programs are a good place to start. But, they are often not very interesting. By introducing students to something familiar and fun they can push themselves and it holds their interest. Working with a GUI (Graphical User Interface – buttons, panels, sliders and so on) leverages the fact that many common programs implement GUIs (see Benyon et al. 2005; Preece et al. 2007). However, GUIs are not always implemented in the same ways. Some use object-oriented programming, others calls to the native graphics of the operating system. As JavaScript and HTML are often promoted as easy to learn languages in high demand in the industry it makes sense to develop in them.

Based on the apparent preference for learning JavaScript and HTML in a fun but realistic developer environment with teams, achievable goals, some autonomy, and an interface that employs graphics elements, we chose to use an online HTML games editor (see www.mogaed.com, use any 4 letters followed by any 4 numbers to login). The core of the project is for students to develop a 2D JavaScript game which launches in a browser, based on a working example which is evaluated by their class cohort.

The online games editor uses some standard buttons and graphic elements as buttons to demonstrate at least 2 different approaches to the same problem. By integrating HTML into the JavaScript this also showed how they can be integrated. JavaScript is used to generate HTML code. But, because JavaScript and HTML syntax is often similar, it is easy to get them confused – especially for where to put quotes as text wrappers etc. Hence, this project included both challenging elements and common solutions. Students started with some fully developed and functional code. Simply changing the variables and seeing what effect they had helped them understand how they function in the overall code.

2.1 Justification for Using a 2D Online Games Editor

From the literature we can identify some core competencies that the current cohort of students need and how to achieve them. The learning activity needs to be:

1. a fun activity;
2. an authentic experience – in a realistic developer environment;
3. able to help them overcome their anxiety about coding;
4. about working with pre-formed working code;
5. integrated with their existing everyday activities, such as game play;

6. about focusing on problem solving, not coding *per se*; and,
7. in a language that is in high demand yet easy to learn.

Structuring learning activities as enjoyable exercises leads to deeper learning outcomes (Bonwell and Eison 1991). By making the learning task enjoyable, students are more likely to engage and complete all the requirements (Prince 2004). Building a small 2D game on a mobile device is the sort of activity most students enjoy doing.

Working in teams (4–5 students per team, 4–5 teams per class), students can leverage each others' strengths whilst making up for deficits in knowledge. The diverse range of skills and backgrounds in a normally distributed student population fosters collaboration and sharing of skills and knowledge. This also parallels the sort of developer environments they are likely to encounter in the industry post-graduation.

Students express their anxiety about code by avoiding it. Presenting them with the raw code up front forces them to confront their fears. Often they simply haven't had the opportunity to engage in a coding exercise that isn't intimidating because they find the code largely incomprehensible and it is not clear how it works or what each part does. Starting with some pre-formed code that works straight up means they can experiment with some simple, clearly labeled, variables and see the effect immediately. This is intended to break down their barriers to learning to code.

The pre-formed code is in a rudimentary form that encourages exploration and experimentation. It is graphically primitive. The first task is to substitute the graphics elements with their own images. This allows them to personalise the game and take ownership of it.

Most students spend at least some of their day playing games on their mobile devices. It makes sense then that they should want to know how to build their own games. The simple editor used in this project lets them choose from a number of pre-formed games which they can customise and combine to create a version, or entirely different game, from.

Because the code is already pre-formed and the various variables and functions are clearly labelled, they can start to be creative immediately. In the creative pursuit of a personalised game, they need to think through what problems they need to overcome to achieve their goals. In this sense, they don't need to learn the code in detail – just enough to solve the problems they encounter.

The interface is fully functional and largely self-explanatory. They do not have to learn a new piece of software just to learn how to code. JavaScript and HTML were used. These are in high demand, encountered on a daily basis and easy to learn. The students do no need to complete the project knowing how to code, rather, they simply need to understand the role coding plays in problem solving. Along the way they pick up some coding knowledge.

The apps they develop are runnable on multiple platforms: PC, mobile device, tablet and so on. The development environment uses a PC with a mouse and keyboard, but the apps they develop can be run on touch control devices. The code is written in such a way as to accommodate the way different browsers treat code differently. As the variables are clearly labelled they can adjust them to see the effect immediately. The immediacy of the feedback promotes confidence in what they are doing. The editor includes error messaging. It shows the user what the problem is and where in the code it has arisen. This facilities easier problem solving at the code level. The interface is a simple, browser-based, text editor. As such, it avoids the added abstraction of the hidden functions in an IDE. It also means the environment variables, links, dependencies and so on do not need to be set up in advance. This removes much of the potential confusion around programming. The apps they develop include the core features of most graphics type applications – a GUI with buttons, graphics, changes and animation, substitution and methods or functions to activate elements such as moving enemies or scrolling player handlers. For a detailed overview of the online editor see: Wyeld and Barbuto (2014).

This project was first formed in 2009 using Flash. In 2014 it was recast as HTML4. Although HTML5 was available, at that time not all mobile devices implemented HTML5 fully. In 2018 it will shift to HTML5 and leverage the power of the 'canvas' function to open up other possibilities. This shift in codebase mirrors the developer world where languages are constantly evolving.

3 Discussion

From the results of the before and after questionnaires used in this case study we found a shift in sentiment from a fear of coding to a more relaxed approach to coding. However, for this group of students, they still reported being largely dependent on external help. This was expressed in their answers to the open-ended questions in the questionnaires about where to find help, the need for tutorials, and example code. But, this contradicts the abundance of these sorts of resources on the internet, such as Stackoverflow, Code.org, Code Mentor and so on. Hence, while it is possible to learn the basics of coding from these sources, having access to help, tutorials and example code appears to instead make them dependent on these resources rather than giving them the confidence to embark on their own coding projects unassisted. The project was designed to address this.

While the project provided a lot of learning scaffolding in the form of help, tutorials, and example code it also fostered inter-student interaction. In other words, it did not rely only on the resources available to the students to facilitate their code learning. Rather, it promoted active problem solving with the tools to implement solutions. It did this by forcing groups of students to work together towards a common goal and for other groups to peer-review their game productions as they were being developed. In this manner, they were focused on solving specific problems rather than learning code alone. In the process, they overcame some of their anxiety about working with code and how to implement changes, even though they may not have fully understood what the structure of the code was.

4 Conclusion

The after questionnaire was used to capture any shift in sentiment across this project. It required students to reflect on their experience in the project and comment on whether their attitudes to the same sorts of questions asked in the before questionnaire had changed. Students expressed a change in attitude to coding. They had changed their way of thinking about coding – from a mysterious, scary, magic ‘code’, to empowerment, and recognition that success is possible. Hopefully, some of the students will be inspired to practice more coding after this project.

From the before questionnaire we could see that many of the participants who took part in this project had used a game editor before. Yet, there was a clear shift in sentiment that indicated coding was less intimidating after completing the project. They felt more confident about coding and understanding what code does. However, they still demanded more in-code comments, coding examples, tutorials, and external help. This is symptomatic of the lack of confidence with the underlying principles of coding. For this group of students, their lack of fundamental skills prior to this project prevented them from taking charge of their own learning. They still felt reliant on external help after completing the project. This is despite a clear shift in sentiment in other ways. It remains to be seen whether their younger siblings, who are learning to code now, will feel the same in the next few decades. By learning the underlying principles of coding at an earlier age they should be less intimated by the complexities of coding later in their education. In the mean time, the current cohort of students needs to overcome their lack of confidence. This project went some way towards that goal. It provided a format whereby students could experiment with code in a non-threatening environment. However, clearly, their demand for more help is troubling. Hence, while this project alleviated some of this cohort’s anxiety about coding, more investigation is warranted. Future projects may investigate whether text-based coding is the most useful format for learning to build apps. Other approaches might include node-based programming. Either way, clearly this is a pressing issue for which a solution is critical if this particular demographic is to achieve its goals of employment post graduation in a world that increasingly demands programming skills.

References

Ackermann, E.: Piaget’s constructivism, papert’s constructionism: what’s the difference? In: Proceedings Constructivism: Uses and Perspectives in Education, Cahier 8, pp. 85–94. Research Center in Education, Geneva (2001)

Althoff, C.: The Self-Taught Programmer: The Definitive Guide to Programming Professionally. Triangle Connection LLC, Mount Dora (2017)

Benyon, D., Turner, P., Turner, S.: Designing Interactive Systems: People, Activities, Contexts, Technologies. Pearson Education, London (2005)

Bonwell, C., Eison, J.: Active Learning: Creating Excitement in the Classroom, ASHE-ERIC Higher Education Report No. 1. George Washington University, School of Education and Human Development, Washington, D.C., USA (1991)

Brooks, F.P.: No silver bullet - essence and accidents of software engineering. In: Kugler, H.-J. (ed.) Proceedings of the IFIP Tenth World Computing Conference, pp. 1069–1076. Elsevier (1986)

Brooks, F.P.: The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, Boston (1995). First published 1975

Code Conquest - Free Coding Guide for Beginners. www.codeconquest.com. Accessed 22 Aug 2017

Code like a Girl - Providing girls with the tools, knowledge and support. https://codelikeagirl.org/. Accessed 22 Aug 2017

Codeacademy. www.codecademy.com/. Accessed 22 Aug 2017

Code.org: Leaders and trend-setters all agree on one thing. https://code.org/quotes. Accessed 22 Aug 2017

Code Mentor - What Programming Language Should a Beginner Learn in 2017? https://www.codementor.io/codementorteam/beginner-programming-language-job-salary-community -7s26wmbm6. Accessed 22 Aug 2017

Disrupting Engineering Education. www.42.us.org. Accessed 22 Aug 2017

Farag, B.: Please don't learn to code, TechCrunch (2016). https://techcrunch.com/2016/05/10/please-dont-learn-to-code/. Accessed 22 Aug 2013

Gamasutra - Better Games Through Usability Evaluation and Testing. http://www.gamasutra.com/view/feature/130745/better_games_through_usability_.php?print=1. Accessed 22 Aug 2017

Kafai, Y.B., Burke, Q.: Connected Code: Why Children Need to Learn Programming. The MIT Press, Cambridge (2016)

Kelleher, C., Pausch, R.: Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. **37**(2), 83–137 (2005)

Krill, P.: Four reasons to stick with Java, and four reasons to dump it, Java World (2014). https://www.javaworld.com/article/2689406/java-platform/four-reasons-to-stick-with-java-and-four-reasons-to-dump-it.html. Accessed 22 Aug 2017

Learn, Share, Build - Stack Overflow. https://stackoverflow.com/. Accessed 22 Aug 2017

Meetup - Learn to Code - For Complete Beginners in Silicon Valley. www.meetup.com. Accessed 22 Aug 2017

Minsky, M.: The Society of Mind. Simon & Schuster, New York City (1988)

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

Norvig, P.: Director of research at Google. http://norvig.com

Petzold, C.: Code: The Hidden Language of Computer Hardware and Software. Microsoft Press, Redmond (2000)

Preece, J., Rogers, Y., Sharp, H.: Interaction Design: Beyond Human-Computer Interaction, 2nd edn. Wiley, Milton (2007)

Prince, M.: Does active learning work? A review of the research. J. Eng. Educ. **93**(3), 223–232 (2004)

Shermer, M.: The Feynman-Tufte Principle: a visual display of data should be simple enough to fit on the side of a van, in Scientific American (2005). https://www.scientificamerican.com/article/the-feynman-tufte-princip

Silicon Valley Coding School - Learn Coding. www.svcodingschool.com.au. Accessed 22 Aug 2017

Wyeld, T., Barbuto, Z.: Don't hide the code!: empowering novice and beginner programmers using a HTML game editor. In: Proceedings 18th International Conference on Information Visualisation (IV), Paris, France, July 2014, pp. 125–131 (2014)