Scientific Research Publishing

Creative Education, 2024, 15, 1223-1236
https://www.scirp.org/journal/ce
ISSN Online: 2151-4771
ISSN Print: 2151-4755

Evolution of the Software Toolchain for Computer Science Students

Yusuf Pisan

Computing and Software Systems, University of Washington Bothell, Bothell, USA
Email: pisan@uw.edu

How to cite this paper: Pisan, Y. (2024). Evolution of the Software Toolchain for Computer Science Students. *Creative Education, 15,* 1223-1236.
https://doi.org/10.4236/ce.2024.156074

Received: May 15, 2024
Accepted: June 25, 2024
Published: June 28, 2024

Abstract

Introductory programming classes continue to be challenging for students. Early attempts in teaching introductory courses focused on teaching language syntax and expanding to procedures for organizing large programs. Later, the object-first approach gained popularity focusing on the object-oriented nature of modern programming languages. We advocate a toolchain-first approach to teaching programming where the tools used in writing, debugging, compiling, formatting, versioning and testing the code are given as much importance as the language being taught. We have successfully integrated industry standard tools into our introductory programming courses. The tools provide the scaffolding to students as they develop their code. By prioritizing tools and tool based help as an integral part of programming, we prepare students for more complex tools, such as LLM based AI companions, that will be used to produce software.

Keywords

Computer Science Education, Introductory Programming, Automated Feedback Systems

1. Introduction

Introductory programming classes are challenging for students (Guzdial et al., 1998; Pears et al., 2007; Porter et al., 2013; Xie et al., 2019). First, there is the conceptual difficulty involved in learning a new language. The first language to teach students has been the subject of a long debate. Java as the first programming language is often chosen as it is popular, easy to learn, enforces good programming practices, isolates the programmer from memory management issues and has a large community and resources available on the web. Pascal as the first programming language has been preferred as it is simple, strongly typed, good

for structured programming and widely used in education. Scheme as a first language is often chosen for its simplicity, minimalism, ability to introduce complex programming concepts, such as closures early on, has minimal syntax and is widely supported by the research community in many universities. Python as a first language has the advantage of simplicity, extensive set of libraries, allows users to write short programs that can perform complex operations. Python's wide use in the data science area also makes it a good candidate for real-world programming. C++ as a first language is argued as it allows the user to learn programming from the ground up, is the most powerful language, can be used to introduce procedural as well as object oriented programming at an early stage.

Regardless of the programming language chosen, the difficulty of learning a new language remains. In fact, even more than the difficulty of learning a new language is learning the software environment that must be used to input this new language to be able to run the programs. There are often specialized integrated development environments (IDEs) that are designed to support a particular language. For example, BlueJ (Kölling et al., 2003) is typically used when learning Java. BlueJ uses boxes to represent classes, and allows the user to double-click on boxes to open text-editors where Java code can be entered. BlueJ has explicit steps for compiling and executing code, highlighting the multistage process that is needed for running a program. In addition to BlueJ, there are other java IDEs, such as Greenfoot (Kölling, 2010), NetBeans (Boudreau et al., 2002), IntelliJ (IntelliJ, 2011), Eclipse (des Rivieres & Wiegand, 2004), JCreater (Ashbacher, 2002) and others.

When we examine IDEs for other programming languages, we encounter a similar landscape, an abundance of choices. As expected, each IDE has their own set of advantages as well as quirks that differentiate it from others. A programmer that is used to working with a particular IDE often has difficulty switching to another. The common reason given is that buttons are not where they should be or the shortcuts that they have been used to no longer work hinder the programming process.

The conceptual difficulty of learning a new language combined with the challenge of mastering a complex piece of software make introductory programming courses difficult. Educators often focus on the programming language that is being studied and dismiss the IDE as just a tool. From students' perspective these two are often inseparable as they have to learn them at the same time. Students often cannot differentiate between errors in programming with configuration or setup issues with the IDE. This problem is further exacerbated with advances in AI programming assistants, such as CoPilot, which help complete code, write new functions, provide just-in-time documentation, making code generation faster, but at the same time requiring more informed guidance from the programmer.

We advocate for a "toolchain-first" approach where understanding and using the software development toolchain is primary (Berry et al., 2006; Pisan, 1994; Pisan et al., 2003; Roy et al., 2002). Students are expected to have a deep under-

standing of the tools they are using, be able to configure and adjust them as needed. We have used this approach since Autumn 2018 and outline how our approach has evolved over time. We use C++ as the programming language. It is especially suited for our purposes since there are many professional quality tools available that can be incorporated into the toolchain.

Using the toolchain-first approach allows getting the maximum help from the supporting tools. The programmer approaches the tools not as something that limits them, but as supporting their primary activity and fading into the background as familiarity with the tools increases. We have found that programs produced by students under this approach follow style-guidelines much more closely, have fewer errors, make use of well-understood programming idioms, incorporate error-checking techniques, and are easier to read and understand.

2. Software Toolchain

The toolchain is closely linked to the programming language being used. For our purposes, we have adopted a C++ toolchain (Wojtczyk & Knoll, 2008). We have adopted C++ as the programming languages as it was already used in our Data Structure and Algorithms courses. C++ is a well-established programming language, widely used in industry and as a result has a mature set of tools that are available for it. Some of the tools that make up our toolchain, such as Visual Studio Code, can be used for multiple languages whereas other tools, such as clang-tidy, is specific to C++. The toolchain-approach emphasizes learning programming in the presence of tools that are assisting the student. Depending on the language chosen, the help available from these tools will differ.

We discuss the various components of the software toolchain below and discuss alternatives.

2.1. IDE—Visual Studio Code

The IDE, an integrated development environment, is central to the student experience (Rahman & Morgan, 2021). Students use the IDE to enter their programs and in many cases trigger additional tools through the IDE. Among software developers, the choice of the IDE is seen as a personal choice. When properly configured, programs written under different IDEs cannot be differentiated. For C++, there is a wide range of IDEs available. Some of the most popular ones are Visual Studio Code (Tan et al., 2023), Eclipse, Clion, Visual Studio, XCode, Atom, CodeLite, Vim, Emacs and Code Blocks. Exploring each IDE in-depth is a very time consuming activity. As a result, programmers choose IDEs based on recommendations and their own limited research.

Introductory programming classes, and even to more advanced data structures classes that follow, often provide limited guidance to students to choose an IDE. One of the reasons for this is that university professors themselves are not professional software programmers, so they may do very limited programming as part of their daily work. Furthermore, their choice of an IDE is often influenced by many years of having worked with their IDE. It is not uncommon for

instructors to be using the same IDE that they have been using since graduate school or to pick an IDE based on limited research and experience.

We have adopted Visual Studio Code, one of the most popular IDEs, as the central part of our toolchain. VSC is commonly used by software developers, so as students mature and become more experienced programmers, they do not need to change the IDE they are using. As expected, students come into the classroom with their own knowledge and preferences. This has in some cases resulted in resistance to use the chosen IDE. Our approach in these cases have been to demonstrate how VSC is part of a much larger software toolchain. While it is possible to substitute VSC with another IDE, it would require significant amount of configuration to interface with other tools. For some IDEs, this level of configuration is not possible. For others, it might be beyond the student's ability.

The default view for VSC is a main editor window for writing code, a left sidebar showing the list of files in the project and a right minimap showing a condensed view of the code. The editor uses multiple colors to highlight different programming features. As the user enters code, IntelliSense automatically suggest completions based on the language. The menubar is typical of many common applications, providing additional operations that can be performed such as opening files, copy-paste operations, finding a specific text, and so on.

Visual Studio Code's strength as an IDE comes from its flexibility. VSC is created to be primarily an editor, whether the user is programming in C++ or Java. The functionality, such as highlighting keywords, matching parenthesis, providing autocompletion, are all provided by language specific extensions that are part of VSC. As a result, VSC can be seen as a language neutral editor. It can be used as easily for Java as it is for C++. See Figure 1 for an example of how VSC uses highlighting to help programmers.

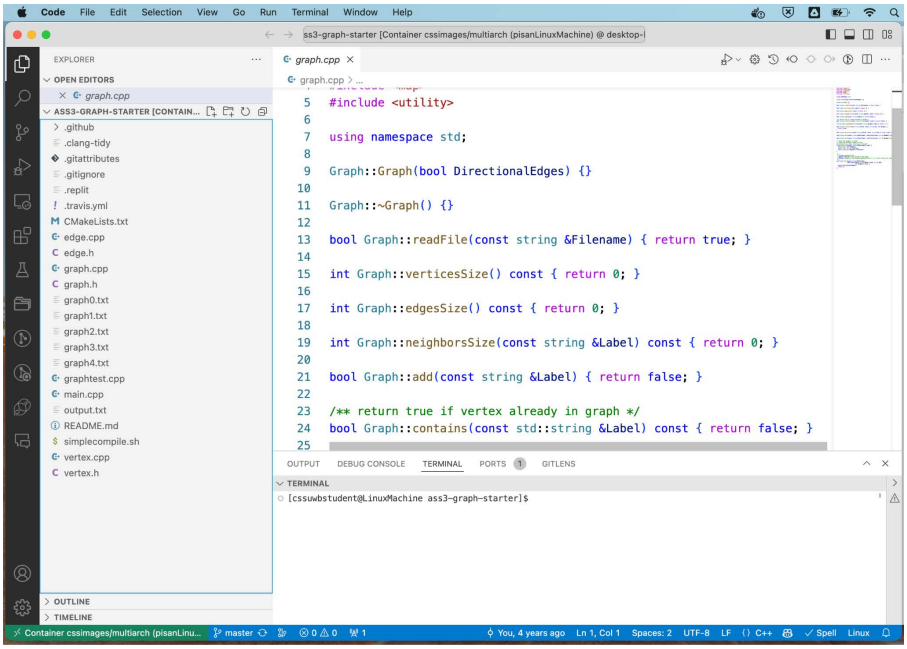

Figure 1. Visual studio code.

While programming classes often use a single programming language, students who are taking multiple classes may need to switch between different programming languages based on the course. As a language neutral editing environment, VSC offers the flexibility to be used in multiple courses for different languages.

A common concern among programmers is whether their tool is free or paid, and along with that what company or user group is supporting the tool. VSC is freely available and is supported by Microsoft. One of the main advantages of being supported by a commercial company is the well-organized web pages, regular updates, extensive set of help pages. In addition, VSC is available for multiple operating systems. As a free tool, VSC enjoys a large community of users that can provide assistance when needed as well as reporting bugs to improve the software.

2.2. VSC Extensions

Visual Studio Code comes with several builtin extensions. First, IntelliSense provides context aware suggestions for code completion. Second, debug features allows developers to step through their code and examine variable values which reduces the need to use print statements for debugging. Third, VSC has integrated Git commands that are used for software versioning and sharing. Fourth, VSC has a large marketplace of extensions that can be easily added. Some of these extensions, such as the C++ extension from Microsoft, has over sixty million downloads. Fifth, VSC allows integration with Microsoft Azure to deploy server based programs, such as Node, React, Angular, etc.

One particular extension, Remote SSH, has proven to be particularly useful for teaching purposes. In a typical classroom, we have students who use Windows, Mac and Linux. It is important to provide a uniform experience for all these students. Having VSC as the central editor is the first step, but it is not sufficient since C++ programs require a compiler before code can be executed. Once again there are many options for compilers, some of them builtin, for different platforms. Unfortunately, there are often small differences in these compilers that lead to differences in student experience. In addition, installing a compiler may not be trivial depending on the platform. This can lead to conflicts where a piece of code successfully runs on the student's computer, but fails to run on grader's computer.

Remote SSH extension allows users to login to a remote machine, a linux based departmental server to seamlessly to create files, compile and execute programs. This way all student programs are tested on the same system. Any changes to the tools, such as the compiler, take effect immediately for all students.

Most computer science departments already have several linux based servers available for students to use. Unfortunately, these servers don't have any graphical user interface, so they cannot be used directly by students to develop programs. By using students' own computers for the graphical user interface part

and having the backend as the departmental server, we use best parts of both systems. Students are editing programs in an environment that they are familiar with while the compiling and execution happens on the departmental servers for consistency of results. In addition, not having to support hundreds of students using an editor on the departmental computers reduces the load for these servers.

2.3. Docker

Remote SSH is a very useful extension to Visual Studio Code. Unfortunately, there are some drawbacks to software development when using this extension. The most significant drawback is since the students are logging into the departmental servers, they are at the mercy of the network connection which can be interrupted. The network connection also can result in lagging where the result of an operation may not be immediately available to the user. This also creates an equity issue as not all students have high-speed internet connection available to them.

After using Remote SSH for over two quarters and trying to fix network issues which ranged anything from servers not responding fast enough to university firewall blocking connections, we started searching for a different solution for students' software development. Docker provides the ability to package and run an application in an isolated environment. We created a Docker image that was identical to the departmental server, making sure that all the necessary tools were installed on the image. Students downloaded Docker, which is freely available, and followed the instructions we provided on the Wiki to download an image and run Visual Studio Code with a Docker connection. Having VSC run with Docker connection makes it possible to access local files that are on the student's computer while at the same time having the same compiler and development tools

Using Docker meant that the departmental servers were not used for programming classes and could be repurposed for other activities. More importantly, since Docker was installed and running on the students' local computer, there were no more any network issues to deal with. Since Docker accesses the files on the student's own computer, students could create directories, zip or move files as needed from their own operating systems interface rather than having to do it from inside a Docker command line.

2.4. ClangFormat

While spaces, tabs, new lines and indentation are not significant for C++ compilers, having a consistent style is essential for code readability. Even among C++ programmers, there is heated debates on how programs need to be formatted. Different companies often have their own style guides, dictating how many spaces need to be used for indentation or whether K&R, Allman, Linux, Whitesmiths or another style needs to be used for matching curly braces. We do not expect this debate to ever conclude, but at the same time it is important to sup-

port students and help them in creating readable code.

Visual Studio Code does not impose a specific style on programs, but leaves it to the user to choose a style. ClangFormat is a set of tools developed by the open source community to specifically address code style. ClangFormat achieves this by having an extensive set of configuration options. These options dictate how many spaces, if any, are to be inserted after an open parenthesis or how many spaces a new line is indented inside a conditional. ClangFormat provides six common styles: LLVM, GNU, Google, Chromium, Microsoft, Mozilla and WebKit. A user can adopt one of these styles fully or use the given style as a starting point and make their own modifications.

We have chosen LLVM as the coding style, mainly because it is the default for the ClangFormat tool. Figure 2 shows a partial list of the styling options that are the default for LLVM code style. The code style is generated using a command line tool and stored in a file called .clang-format. Visual Studio Code automatically examines this file and indents the code based on the configuration options specified. If a student is using Visual Studio Code and ClangFormat as advised, there is nothing else they need to do in terms of style. Cognitive load with writing a program is reduced since the IDE takes care of the trivial issues such as spacing allowing the student to focus much more on content. As instructors, we can run the ClangFormat tool on a given program to check that it conforms to the standards. If it does not, a report is automatically generated. This report can be provided to the students as feedback. This process significantly reduces the instructors' work of providing style feedback, allowing instructors to focus on more higher-level feedback. As an instructor who has had to tell students over and over how to indent their code and who had to give the same feedback to dozens of students each assignment cycle, I can confirm from personal experience the difference a tool like ClangFormat makes in reducing the workload.

2.5. Clang-Tidy

Clang-tidy is a C++ "linter" tool designed to fix typical programming errors, like style violations, interface misuse, or bugs that can be deduced via static analysis. For example, a typical mistake by novice programmers is creating infinite loops by accident, as seen in Figure 3.

This unintended infinite loop can be detected through static analysis. clang-tidy can be integrated into Visual Studio Code to provide context aware feedback as the program is being written or can be run as an independent program by the instructor when students have submitted their assignments. Similar to clang-format, if a student is using clang-tidy with VSC integration, they can notice mistakes in real-time while working on the code. For students who fail to utilize the full power of the software toolchain, the report from running clang-tidy can be provided as feedback to the students.

The clang-tidy checks are organized in twenty-five categories. For novice programmers, the most important category is "bugprone" which indicates that a

piece of code is likely to be the result of a bug. There are eighty-three different checks in this category. The ones we have found most frequently occurring are:

Figure 2. ClangFormat styling options.

```
int i = 0, j = 0;
while (i < 10) {
   ++j;
}
```

Figure 3. An infinite loop.

1) assert-side-effect
2) assignment-in-if-condition
3) bool-pointer-implicit-conversion
4) empty-catch
5) inc-dec-in-conditions
6) incorrect-enable-if
7) incorrect-roundings
8) infinite-loop

9) integer-division
10) switch-missing-default-case
11) too-small-loop-variable
12) unused-return-value

By introducing clang-tidy as part of the student development environment, we are able to catch mistakes at the earliest possible point. This saves, as much as possible, from students making silly mistakes, reduces cognitive load and allows students to work on the main task rather than spending precious time on debugging code.

2.6. Valgrind

One of the reasons we use C++ programming language is for students to understand how memory management works in software. While some programming languages automatically take care of memory management, we believe that a deep understanding of memory usage is essential for computer science students. Unfortunately, as programs get larger, memory management gets trickier. valgrind is a tool that runs the executable in a protected memory environment to detect any memory access issues or leaks with the program. Since it is a run-time analysis, it is only capable of detecting issues with the branches of code that gets executed. Despite this limitation, valgrind remains one of the best tools available for detecting memory leaks.

We had tried integrating valgrind earlier, but found that the executable was not available on all the desired platforms. Once we have made the switch to use Docker, we were able to add valgrind as part of our standard toolset. The integration of valgrind with Visual Studio Code still remains somewhat problematic. Unlike other extensions, valgrind has to be run from the command-line interface explicitly as a separate step by the programmer. The output from valgrind can also be somewhat cryptic, but by providing examples to students on what indicates acceptable output and what indicates a problem with the code, we are able to make the best use of this tool.

2.7. Test Based Development—Assert

How do we decide when a program is working correctly? In the simplest case, we examine the output and compare it to the expected output to check correctness (Sterner, 2021). This is a very rough measure since a program has to work correctly for many different inputs and checking the correctness of the output for each input at a time can be both time consuming and prone to error. Test-based development is a specific software engineering practice that advocates starting program by writing test cases and incrementally developing the program to satisfy the necessary test cases. While test-based development is a desirable goal, in early stages of learning programming, it introduces additional complexity for novice users. Instead we use "assert" statement embedded in the starter code to check the correctness of values as the program is running.

The advantage of using "assert" statements is that if the value in the statement is false, the program terminates at that point making it clear to the student that something is wrong. This is infinitely better than print statements that are used for debugging where finding first print statement that does not match the user expectation can often be tedious.

2.8. Test Based Development—Google Catch

We used "assert" statements successfully for multiple years, but there are some clear drawbacks of this approach, so we searched for more robust solutions. Google Catch frameworks provide a more complex testing structure, but it also has several advantages. First, the tests are run in parallel which makes testing large programs much faster. Second, an actual report on test results is produced allowing the programmer to examine the report rather than relying on program termination to determine if there are any errors. Third, Google Catch framework allows for more complex tests to be created using explicit setup and tear-down procedures.

Introducing Google Catch into the software development toolchain does make the toolchain slightly more complex; special compilation instructions are needed when compiling programs using the Google Catch library. The advantage is the introduction of professional level software development tools which eases students' transition from the educational environment into industry.

2.9. Software Versioning—GitHub and Continuous Integration

Complex software is developed by teams over multiple years. This requires implementing version control systems that can keep track of changes made as well as who has made changes to the program. There are many different software versioning systems, but GitHub (Puryear & Sprint, 2022) is especially popular with open source software projects as it is freely available and has extensive sets of features.

Student assignments are often completed individually or in small teams. Unlike industry projects, student projects are mostly written from scratch rather than being built on top of a large codebase. Nevertheless, keeping track of software versions is an important skill for students. In addition, using GitHub for student project automatically enables a backup of the software to be kept on a remote site.

There are several fundamental concepts that students need to master as part of keeping software versions. First, any changes are *pull*ed from the remote repository to the local computer. Once the student has made the necessary changes, programs are *commit*ted to the local database. The final step is, *push*ing the code to the remote repository to make it available to others. As part of the *push* action, students also have to write a short description of the change that has been made for recordkeeping purposes.

One of the features that GitHub, as well as other remote repositories, offer is

to have automated actions that can be triggered every time new changes are *push*ed onto the codebase. This is typically referred to as *continuous integration* and can be configured. For example, for programming courses we have the actions configured to compile the updated software and run it though the series of tools that make up the toolchain to generate a report. This report highlights any problems with the student code and make it easier to debug programs.

2.10. Starter Code—Google Classroom

To help students learn good programming habits, we need to provide them with good examples. One way we achieve this is through providing starter code for different assignments. We use Google Classroom for these purposes as it has some significant advantages to just making the code available on a web page to be downloaded. Google Classroom integrates with GitHub where a specific public repository can be designated as the starter code. Once a student accepts the assignment by clicking on the invitation link, a private GitHub repository is setup for the specific student and the starter code is copied into the private repository.

Setting up a private repository for each student provides them with their own software development tools and space. The private repository is still accessible by the instructor, so that instructors can check on student progress as they are working on the assignment.

3. Results

We have been using the toolchain-first approach for the last five years. Despite its obvious advantages, switching to it has required some additional training for instructors as well as fine-tuning instructions for students and instructors on how to setup the different tools. We currently have 2200 private repositories at https://github.com/orgs/uwbclass/repositories corresponding to different student assignments and actively use this toolchain in courses. The change of focus from language teaching to toolchain mastery has been in multiple stages as each piece of the toolchain was introduced. We have noticed several improvements as a result of this approach.

Efficiency. Visual Studio Code improves students speed in developing programs. For example, students are able to refactor code easily by changing a function name in all the places that occur throughout multiple files which makes it easier to improve the code for readability. The color coding provided by VSC makes any typos immediately visible. Automatic indentation provided means that students are no longer chasing after a misplaced semicolon and can focus on higher level tasks.

Testability. The introduction of assert statements initially and then the more complex Google Catch framework means student code incorporates unit test from the very start. In addition, instructors can add their own test files to further test student code. This has lead to more robust software that is tested at every

stage.

Maintainability. Student code is standardized using tools such as clang-format and clang-tidy which makes it easier to read this code and make improvements as necessary.

Ease of Managing Large Classes. The toolchain that we have adopted for our classroom is similar to the toolchain used by thousands of software developers in large companies. As a result, tools for configuring, distributing and standardizing this toolchain is readily available which makes it easy for instructors to manage large classes. In addition, the use of the standard chain makes it easy for students to adapt to professional company environments easily.

4. Limitations

There are several limitations of this study that we plan to remedy with further study. First, it is difficult to compare the efficacy of the toolchain-first approach quantitatively as it is difficult to establish a control group. We plan to compare students' performance in a toolchain-first classroom with a traditional classroom. However, when comparing students, in addition to the pedagogical approach, there are other factors, such as the instructor, in-class exercises and homeworks conducted and even what time the class is held that can impact student performance. Similar to the debate on object-first approach versus procedural approach in teaching programming languages, we expect the toolchain-first approach will require multiple studies to establish its advantages.

Second, we have used C++ as the programming language and have made specific choices on the tools that compromise the toolchain. We need to explore whether the improvements we have seen hold for other languages. For each language, there are different set of tools, with varying levels of usefulness, that can be used which will impact student learning. We have adopted a toolset based on their use in professional software development environments, but there are often alternatives for each tool, and sometimes company specific tools that programmers can use that we do not have access to. The toolchain approach emphasizes getting maximal use from tools which separates it from traditional approaches where the tools may or may not be used.

Third, our work has been conducted at a single institution, on classes with forty to forty-five students who are all intending to study computer science. The toolchain-first approach needs to be tested at multiple institutions with different class sizes and with a wider range of students to determine its efficacy. The feedback we have received is similarly limited to a specific cohort, a longer study to collect feedback from students to refine the approach is necessary.

5. Conclusion

We have successfully integrated the professional programming toolchain into the classroom resulting in much easier management of large classes, improved student code, better tested student programs and improved student experience.

Leveraging the support that these tools provide has made the programming a more collaborative experience for the students. As additional tools, such as LLM based AI tools, become commonplace, the support for the toolchain will continue to improve. Our approach of using industry standard tools better prepares students for work in industry making programming much more of an exploratory task.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

Ashbacher, C. (2002). JCreator 2.0 LE. *Mathematics and Computer Education, 36,* 90.

Berry, R., Makino, M., Hikawa, N., Naemura, M., Pisan, Y., & Edmonds, E. (2006). Programming in the World. *Digital Creativity, 17,* 36-48. https://doi.org/10.1080/14626260600665728

Boudreau, T., Glick, J., Greene, S., Spurlin, V., & Woehr, J. J. (2002). *NetBeans: The Definitive Guide: Developing, Debugging, and Deploying JAVA Code.* O'Reilly Media.

Des Rivieres, J., & Wiegand, J. (2004). Eclipse: A Platform for Integrating Development Tools. *IBM Systems Journal, 43,* 371-383. https://doi.org/10.1147/sj.432.0371

Guzdial, M., Hohmann, L., Konneman, M., Walton, C., & Soloway, E. (1998). Supporting Programming and Learning-To-Program with an Integrated CAD and Scaffolding Workbench. *Interactive Learning Environments, 6,* 143-179. https://doi.org/10.1076/ilee.6.1.143.3609

IntelliJ, I. (2011). *The Most Intelligent Java IDE.* JetBrains. https://www.jetbrains.com/idea/#chooseYourEdition

Kölling, M. (2010). The Greenfoot Programming Environment. *ACM Transactions on Computing Education, 10,* 1-21. https://doi.org/10.1145/1868358.1868361

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The Bluej System and Its Pedagogy. *Computer Science Education, 13,* 249-268. https://doi.org/10.1076/csed.13.4.249.17496

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J. et al. (2007). A Survey of Literature on the Teaching of Introductory Programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education* (pp. 204-223). Association for Computing Machinery. https://doi.org/10.1145/1345443.1345441

Pisan, Y. (1994). Visual Reasoning with Graphs. In *Eighth International Workshop on Qualitative Reasoning about Physical Systems* (pp. 205-211). Nara.

Pisan, Y., Richards, D., Sloane, A., Koncek, H., & Mitchell, S. (2003). Submit! A Web-Based System for Automatic Program Critiquing. In *Proceedings of the Fifth Australasian Conference on Computing Education* (pp. 59-68). Australian Computer Society.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in Introductory Programming. *Communications of the ACM, 56,* 34-36. https://doi.org/10.1145/2492007.2492020

Puryear, B., & Sprint, G. (2022). Github Copilot in the Classroom: Learning to Code with AI Assistance. *Journal of Computing Sciences in Colleges, 38,* 37-47.

Rahman, M. M., & Morgan, R. P. (2021). A Remote Instructional Approach with Interac-

tive and Collaborative Learning to Teach an Introductory Programming Course during COVID-19 Pandemic. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)* (pp. 940-946). IEEE. https://doi.org/10.1109/csci54926.2021.00210

Roy, J., Richards, D., & Pisan, Y. (2002). Helping Teachers Implement Experience Based Learning. In *International Conference on Computers in Education, 2002. Proceedings.* (pp. 1396-1397). IEEE. https://doi.org/10.1109/cie.2002.1186265

Sterner, K. (2021). *Automated Checking of Programming Assignments Using Static Analysis.* Malardalen University.

Tan, J., Chen, Y., & Jiao, S. (2023). *Visual Studio Code in Introductory Computer Science Course: An Experience Report.* arXiv: 2303.10174 .

Wojtczyk, M., & Knoll, A. (2008). A Cross Platform Development Workflow for C/C++ Applications. In *2008 the Third International Conference on Software Engineering Advances* (pp. 224-229). IEEE. https://doi.org/10.1109/icsea.2008.41

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H. et al. (2019). A Theory of Instruction for Introductory Programming Skills. *Computer Science Education, 29,* 205-253. https://doi.org/10.1080/08993408.2019.1565235