Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 2, No. 4, February : 2024

A THOROUGH LOOK AT SMART TEACHING PROGRAMS FOR LEARNING PROGRAMMING

Mr. Achinta Kumar Palit, Assistant Professor, Dept.Of Computer Science Engineering, Gandhi Institute of Technology and Management, Bhubaneswar.
Mr. Sunanda Kumar Sahoo, Miss Lipsa Pattnaik, Miss Somya Sucharita Swain, Miss Nilam Simran, Mr. Prasanjit Das, Assistant Professor, Dept.Of Computer Science Engineering, Gandhi Institute of Technology and Management, Bhubaneswar

Abstract
A wide range of intelligent tutoring solutions have been developed with computer programming instruction in mind. The majority of published research focuses on systems designed for use in postsecondary education to teach programming. While some systems instruct more specialized topics like scope or recursion, most systems have been designed to teach basic programming ideas. According to published research, these systems tackle a lot of the challenges that come with teaching beginners to program; that being said; each system is very different and has a lot of extra features that have been included. Interactive programming exercises are a common component of most intelligent programming tutors; nevertheless, different systems use plans, tests, and worked solutions in very different ways. Important details regarding current systems and the frequency of various aspects within them are reported in this systematic review. An outline of the integration of supplemental features into these systems is provided, along with suggestions for how a greater variety of extra features could be supported to enhance intelligent programming teachers.
Keywords: Intelligent Tutoring, Programming Education, Novice Programming

I. Introduction

Intelligent tutoring systems are made of digital technologies that aim to mimic the efficiency of one-on-one human tutoring. Experiments with human tutors have clearly demonstrated the superiority of one-on-one tutoring over larger group instruction [3]. The challenge in the field of intelligent tutoring is to duplicate these advantages in computer-assisted learning systems, which involves incorporating user-adaptive features to customize education. Research in the intelligent tutoring domain has demonstrated benefits with effect sizes comparable to those of human tutoring [11]. Many intelligent tutoring systems have been developed for teaching programming. They shall be referred to as Intelligent Programming Tutors (IPTs) for the duration of this review. Because learning programming presents unique challenges not encountered in other domains, intelligent tutoring systems can be distinguished by their subfield, known as IPTs. These technical difficulties to programming education are specific to IPTs, which are typically designed to do complicated tasks like providing clues on the syntax and semantics of student-produced programs. The majority of IPTs require environments in which to develop and run code. Many intelligent tutoring systems have been developed for teaching programming. They shall be referred to as Intelligent Programming Tutors (IPTs) for the duration of this review. Because learning programming presents unique challenges not encountered in other domains, intelligent tutoring systems can be distinguished by their subfield, known as IPTs. These technical difficulties to programming education are specific to IPTs, which are typically designed to do complicated tasks like providing clues on the syntax and semantics of student-produced programs. The majority of IPTs require environments in which to develop and run code.

II. Methodology

The Guidelines for Performing Systematic Literature Reviews in Software Engineering [10] served as the foundation for the systematic review's methodology. These guidelines served as the foundation

for the research questions, search tactics, inclusion and exclusion criteria, and data extraction procedures that are described below.

Search String

There are several terms used in the field that has to do with intelligent tutors and programming education. We employed a boolean search string with frequently used synonyms for intelligent tutoring and programming education: ("computer science education" OR "software engineering education" OR "introductory computer" OR "introductory programming" OR "teach* programming" OR "learn* programming" OR "novice programming" OR "coding education" OR "cs1" OR "introductory computer science") AND ("intelligent tutor*" OR "adaptive tutor*" OR "cognitive tutor*" OR "smart tutor")

False negatives were checked for by repeating searches with new terms that could be considered synonymous that were found in the literature. In all cases the more obscure synonyms did not result in any new search results being returned that met the inclusion criteria. The search string was also tested with a known sample of articles that matched the inclusion criteria to check that they were included in the search.

2.1 Article Database

The systematic sample was retrieved from the ACM digital library and IEEE. All other papers were obtained from backward referencing. ACM was identified as the most relevant database in the field and the best place to obtain a systematic sample from.

2.2 Data Extraction

The full text of articles was read after it was determined that they satisfied the requirements for inclusion. The name of the IPT, adaptive features, and basic information about the programming language that was taught were extracted. Additionally, boolean variables pertaining to the additional features found in the tools were extracted. To establish some common findings across the field, boolean variables relating to reasonably broad categories were chosen. The outcomes are shown in Table.2.

Explanations of the variables are displayed below:

- Name of IPT.
- Programming language taught - what programming lan- guage is taught by the tutor.
- Primary adaptive features - is feedback and/or navigational support adapted.
- Questions - does the IPT include the use of questions related to programming that do not involve typing code.
- Plan creation - does the IPT include tasks related to user generated program planning or visualization.
- Supplied plans or visualizations - do the IPT use pre-made plans or visualizations of programs as instructional resources.
- Lessons - does the literature report that there is lesson con- tent beyond what is included in programming task descriptions.
- Reference material - does the literature report organized reference material that can be referred to by students.
- Worked solutions - does the literature report worked so- lutions or examples of programs used as an instructional feature within the IPT.

III. Results

78 articles that matched the search string were found when searching ACM. 46 of these articles satisfied the general requirements of being literature about ITSs related to computer science education or programming, based on the abstracts of these articles. A more thorough analysis of the entire text of these articles was conducted to determine the number of individual ITSs reported in the literature and the proportion of these that tutored a particular aspect of scripted programming. Many

of the articles that were retrieved were false positives and had nothing to do with teaching programming. Numerous articles were disqualified for failing to report specific systems, reporting an unproven prototype for a single tutoring function such as feedback generation, or failing to address scripted programming. A few specific IPTs were mentioned in several different works of literature. In these instances, reading every relevant article was done before extracting the variables. 14 distinct IPTs were found after forward referencing important literature found in the articles that were methodically retrieved.

With the search string, IEEE returned only 20 results, none of which met the inclusion criteria. The majority of the results were articles that mentioned some aspect of intelligent tutoring or more general intelligent tutoring in other areas. The first 50 results of a second search that was conducted without quotation marks or Boolean expressions and included the term "intelligent tutoring system programming education" were examined; this too did not meet the inclusion criteria.

3.1 Programming Languages

All of the IPTs that were obtained from ACM and backward referencing that met the inclusion and exclusion standards for university-created tools meant for particular courses. It was reported that only non-scripted programming languages like Alice were utilized in the creation of the only tools intended for pre-tertiary students [5]. Within the 14 IPTs, a variety of languages were taught. According to every study, the IPT was designed to teach a language that was already decided to be necessary for the course. The IPTs represent a wide range of languages; four of the IPTs teach Java, which is the most common language. Table 1 displays the languages and main adaptive characteristics.

3.2 Primary Part Of Tutoring That Is Adapted

The portion of the tutoring process that is modified differs amongst various IPTs. The two most comprehensive categories that were amenable to comparison were adaptive navigational support and adaptive feedback. They're shown in Table 1. Feedback for student-produced programs includes debugging assistance and suggestions for the next step. Syntax, style, and semantics can all be addressed in feedback. Navigational support encompasses methods of visualizing progress and the next steps in learning as well as task or resource assignment. It is significant to note that, because some studies use definitions that narrowly relate the specific types of adaptive components, some of the adaptive features reported in the literature would not meet some more specific working definitions of being intelligent features. These are being reported in order to provide a summary of the IPT sections that are being made adaptive.

Intelligent or adaptive feedback regarding programs. Adaptive or intelligent feedback on student-produced programs is a prevalent characteristic of many IPTs. The general goal of more intelligent or adaptive feedback is to mimic the kind of guidance that a human tutor would provide to inexperienced programmers. Numerous techniques were employed to customize the feedback provided in the IPTs sampled, and a wide range of feedback formats were offered. The difference between IPTs that provide step-by-step instructions and IPTs that solely provide summative feedback on submitted code is among the most obvious. It must offer program feedback in a way that goes beyond what simple unit testing or IDE/console errors could in order to be categorized as adaptive or intelligent feedback.

Out of the 14 IPTs reviewed, 11 of them had more advanced feed- back detailed in the literature. Only two of the IPTs were identified as having no adaptive feedback. A to L was reported to allow students to complete programming tasks but there were no advanced methods of feedback mentioned [29]; the literature was focused on advanced navigational support. It is assumed that in this tutor the primary method of feedback is based on the pass/fail unit testing of programs. One IPT in particular reported assigning programming tasks but did not report the inclusion of a programming environment in the IPT; within this IPT students were given intelligent navigational support through a range of learning materials and assigned programming tasks that they could complete outside of the tutoring system [25].

The majority of techniques for more sophisticated adaptive feedback were published with reference to programming tasks rather than the extra features that are the primary subject of this review. Since adaptive feedback techniques have already been covered in a previous literature review [14], more specific information is not provided here.

Navigation that adapts. In certain IPTs, adaptive navigation is a crucial component. It was found that six of the fourteen IPTs that were examined had adaptive navigation of some kind. More information regarding adaptive navigation is provided in Section 3.3.4. In the majority of these situations, the adaptive navigation is connected to educational and reference resources.

3.3 Supplementary Features Of The Ipt

Table 2 lists specific elements of the tutoring procedure within each IPT. Le et al.'s review of AI-supported tutoring methods for learning programming [15] offered a classification of the main methods across a variety of IPTs Intelligent Tutoring Systems for Programming Education ACE 2018, January 30-February 2, 2018, Brisbane, QLD, Australia

Table 1: IPT Languages Taught

Tool Name	Language	Adaptive Feature
PROUST [9]	Pascal	Feedback
LISP Tutor [2]	Lisp	Feedback
ITEM/IP [4]	Turingal	Feedback, Navigation
C-Tutor [26]	C++	Feedback
ELM-ART [27]	Lisp	Feedback, Navigation
Scope Tutor [12]	Pascal	Feedback
ILMDA [25]	Java	Navigation
AtoL [29]	Java	Navigation
CIMEL ITS [17]	Java	Feedback, Navigation
CPP-Tutor [18]	C++	Feedback, Navigation
J-LATTE [7]	Java	Feedback
ChiQat [1]	Un-specified	Feedback
Ask-Elle [6]	Haskell	Feedback
ITAP [24]	Python	Feedback

Table 2: IPT Features

IPT Name	QU	CP	PL	LE	RE	SO
PROUST [9]	-	-	-	-	-	-
LISP Tutor-[2]	-	-	x	x	-	-
ITEM/IP [4]	x	-	x	x	-	x
C-Tutor [26]	-	-	-	x	-	-
ELM-ART [27]	x	-	-	x	x	x
Scope Tutor-[12]	-	-	x	-	-	-
ILMDA [25]	-	-	-	x	x	x
AtoL [29]	x	-	-	x	-	x
CIMEL ITS[17]	x	x	x	x	x	x
CPP-Tutor [18]	-	-	-	-	-	x
J-LATTE [7]	-	x	-	-	-	-
ChiQat [1]	-	x	x	x	-	x
Ask-Elle [6]	-	-	-	-	-	-
ITAP [24]	-	-	-	-	-	-

This section includes a list of six features that are easily distinguishable in certain systems but not in others. These features, which are important components of the tutoring process that the IPT has automated, are typically included in a system's structured tutoring strategy. The only problem with this is that there's a slim chance that some literature overlooked certain aspects that weren't the publication's primary focus. Since the majority of the studies and features are fairly easily recognized and play a crucial role in the tutoring strategies described, this is not thought to be likely.

- QU (3.3.1) — Questions students answer within the IPT; for example, multi-choice, true and false quiz or short-answer questions; these are distinct from programming problems which students answer by producing code.
- CP (3.3.2) — Creation of some form of program plan by students within the IPT.

- PL (3.3.3)— Pre-made or computer generated plans or visualizations of programs used as teaching resources.
- LE (3.3.4)— Lesson materials on programming concepts.
- RE (3.3.4)— Reference materials.
- SO (3.4) — Worked solutions supplied as an instructional resource.

Questions (QU): Five of the fourteen IPTs that were examined used questions such as multiple-choice, true-false, and short-answer questions that do not require the production of code.

In order to respond to highly targeted questions in ITEM/IP [4], students had to mentally run programs and input their results. No further kinds of questions were reported by ITEM/IP. While conceptual knowledge is frequently the subject of questions, procedural knowledge is related to short-answer questions in ITEM/IP because the user must mentally execute the program by tracing the behavior of the procedures in the given code examples.

AtoL [29] includes a course administration component where teachers can create lessons, as well as program and question tutors fresh workouts. There are three different kinds of questions: multiple choices, short answer, and true/false. AtoL was created especially for laboratory students. Within AtoL, students can choose between programming and question modes. At the conclusion of the lab, a summary of the students' performance on each of these task types is delivered to the lab facilitator. In order to modify the kinds of resources and activities that students are exposed to, AtoL also includes a different kind of question concerning their learning preferences. This question type differs from all other IPT questions because it is not connected to any conceptual or procedural knowledge; instead, it is just used to modify the tutoring tool. This is a noteworthy feature of the IPT since it has been noted that some users have particular preferences for the kinds of supplemental features they would like to use while using interactive learning tools.

Simple questions and programming exercises are said to be included in ELM-ART [27]. The questions are said to be placed inside of a "electronic textbook" and have the option of being automatically marked. There are five distinct categories of exam items in ELM-ART: gap-filling, forced-choice, multiple-choice, free-form, and yes-no questions. It is stated that the way in which students perform on questions and programming exercises is connected to "navigational hints," which encourage or discourage a user from continuing with a certain page of the electronic textbook. Testing is said to be the most trustworthy way to determine whether a user has understood a concept and has the biggest impact on modifying instructions to point them to the next most beneficial page to learn.

Another IPT with integrated reference material is CIMEL ITS [17]. The tool is said to include interactive exercises like drag and drop activities in addition to quiz questions. The reference material is laced with interactive material that is intended to test students' comprehension and reinforce ideas. To ascertain the likelihood that a student would comprehend related concepts, the outcomes of exercises are connected to a student model, which is a Bayesian network based on the domain model. The interactive activities are essential for developing the model of student understanding because the structure of the reference material closely aligns with the domain model. The significance of this lies in the fact that in all three of these IPTs, questions play a crucial role in both the software's adaptive nature and the student modeling that enables the IPT to predict users' likelihood of completing tasks before assigning them.

Diagrams created by users were also used by ChiQat [1]. ChiQat focuses on both programming specific algorithms and data-structures. The two primary teaching modules that are mentioned in the literature are linked lists and recursion. Recursion graph creation is related to user-generated visualization in ChiQat. ChiQat's dual focus on data-structures and graph creation are more closely related to each other than with scaffolding the use of planning tools during the design of traditional sequential programs. Out of the fourteen IPTs, only three actually implement plan creation. This is noteworthy since having a plan creation module has many clear advantages. The J-LATTE [7] approach, which lets users create programs by alternating between concept and program mode, has

the advantage of enabling students to create programs at the concept and block level, where there are fewer potential stylistic variations and incorrect syntax is avoided. Plan creation activities in IPTs for introductory programming are severely lacking, despite the potential advantages, as CIMEL ITS [29] and ChiQat [1] teach more specialized programming concepts of object-oriented programming and recursion, respectively. Improving program planning abilities may be one strategy to lessen the frequency of semantic errors and enhance the student programs' logical structure. Pre-conceived Concepts or Visualizations (PL). Use of pre-made plans or interface program visualizations was reported by five of the IPTs. In the planning mode of LISP Tutor [2], users can see a text-based program plan. Users can see an outline of the program they are creating in a planning mode, which can be activated if it is determined that they are having trouble. Based on what has been published in the literature, the planning mode is more of a concise summary of the task than a comprehensive pseudo code plan.

Within ITEM IP [4], users have the ability to submit support requests, examine a preliminary program blueprint, and receive guidance on how the program operates through the display of program actions. The most important thing to keep in mind is that plans aren't used at the beginning of exercises; instead, they are part of student hints. Plans are not discussed in great detail in ITEM IP, but it is assumed that they are probably straightforward text-based plans intended to enumerate the essential specifications of the task. Additionally, users can view basic code tracing in ITEM IP's basic visualization mode. Based on a computer-generated static tree, the Scope tutor [12] shows an outline and visualization of the program that needs to be created. According to reports, giving students an outline instead of a full program acts as a type of scaffolding to help them focus their attention when solving problems and to identify the areas of a typical program that they should pay particular attention to when analyzing the scope of a completed program.

UML diagrams are used in CIMEL ITS [17]; as stated in Section 3.3.3, users also generate UML diagrams, which serve as the primary means of plan integration. Because CIMEL ITS emphasizes object-first programming, UML plans are essential. The Eclipse IDE and CIMEL multimedia are integrated by CIMEL ITS; in CIMEL multimedia, pre-made UML diagrams are used to scaffold the creation of them in the Eclipse plug-in.

In addition to user-generated graphs and visualizations, ChiQat [1] incorporates pre-made visualizations of data structures and algorithms into its educational materials. The section on user-generated plans and visualizations has already addressed this.

The literature does not go into great detail about LISP Tutor [2]'s basic text-based planning mode, but it is said that users can switch to it to view important details about the program they are currently working on.

Similar to the process of creating plans outlined in Section 3.3.2, plans and visualizations are not frequently employed in tools for beginning programming. Just three of the programs that use pre-made plans teach general programming skills. Recursion and scope are taught by ChiQat [1] and Scope Tutor [12].

Additionally, some IPTs include arranged reference material that is available for browsing and request at any time. In certain IPTs, this reference material is arranged like a digital textbook and includes interactive questions. Reference material is more comprehensive content that can be viewed by the user in a non-linear manner and is not only organized within lesson sequences. Table 2 provides a summary of the IPTs that contain lessons and reference material.

IV. Discussion

IPTs contain a wide range of distinct features. It is clear from methodically examining how questions, lessons, planning exercises, reference materials, and worked solutions are used within IPTs that a wide range of features make up the tutoring processes within IPTs. Examining the features and resources covered in this review is worthwhile because, aside from IPTs, all of them have historically played some role in programming education. Plans, reading lists, practice questions,

and other extracurricular activities are not exclusive to IPTs. Although many IPTs only incorporate a few of these elements, asking learners about important ideas, teaching them how to write code plans, having them work through solved problems, and utilizing reference materials are all potentially beneficial methods to enhance learning. While it would not be feasible or advantageous to incorporate every feature into every IPT, additional features might be able to address a lot of the issues that arise in the field of IPTs. Plans, which are comparatively underutilized in IPTs, could aid in addressing the inherent challenges of helping participants comprehend program structure and related problems.

Tools such as ELM-ART [27] and CIMEL ITS [17] combine auto-marking with reference content to improve the intelligence and interactivity of student models and navigational support. Although reference materials may be thought of as conceptually static, in certain IPTs they serve as the foundation for student modelling and models of domain knowledge. Since the reference material and the student models share the same domain models, adaptive navigational support can be implemented with relative ease. It may be more difficult for students to model, navigate adaptively, or complete tasks if programming tasks are not contextualized within larger, more thorough reference materials. Relatively few IPTs are linked with reference materials, given the abundance of pre-existing materials for programming education.

V. Conclusion

The features of IPTs that are related to aspects of programming education that are not found in tutoring systems are presented in this review. The application of these features in a methodical sample of IPTs is presented in this review. It is clear that no set of features has been applied consistently in the field of intelligent tutoring when it comes to programming education. This paper's main contribution is its description of the various ways in which features have been implemented and the frequency with which particular features appear in various systems. Certain IPTs incorporate extra features into the intelligent part of the system that aren't specifically related to typed programming. This has bearing on the IPT field. When creating systems It would be beneficial to think about connecting additional resources to the system's intelligent and adaptive component when developing the intelligent-gently tutor programming. Further resources beyond programming assignments may be added to improve the efficacy of the intelligent component. Creating intelligent feedback and hints about syntactic and semantic problems in programming tasks is a laborious task. Many distinct features can be used to aid in these processes. The difficulty of providing feedback and suggestions regarding the organization of students' programs in tasks where there are several approaches to solving an issue is one particular illustration of this. Although thoughtful criticism on program design is beneficial, incorporating user-generated program plans into One ought to think about an IPT. A user is less likely to introduce structural issues into the code they create if they can plan a program algorithmically. Additionally, planning supports sound programming techniques. Additionally, planning tasks can be automatically marked and incorporated into the clever part of Intelligent Tutoring Systems for Programming Education ACE 2018, Brisbane, QLD, Australia, January 30-February 2, 2018. auto-marking plans are typically more effective because they are typically created at the statement or block level and have a lower degree of syntactic variance. The lack of extensive reference material integrated into the intelligent component of most IPTs leaves a gap in the field. Of the IPTs reviewed, only two—ELM-ART [27] and CIMEL ITS [17]—reported comprehensive The lack of extensive reference material integrated into the intelligent component of most IPTs leaves a gap in the field. Merely two of the IPTs under review—ELM-ART [27] and CIMEL ITS [17]—provided comprehensive reference materials paired with programming tasks and inquiries. Not only do reference materials provide students with an additional resource, but they are also inherently a thorough mapping of do-main knowledge. A fundamental component of the majority of intelligent tutoring systems is student modeling. A common method of modeling what interconnected concepts a student understands and subsequent steps, such as what they should learn

next or repeat, is to map a student model onto a domain model. Student role-playing can be successful There are numerous other ways that planning exercises and reference materials could be incorporated into intelligent tutoring, in addition to a host of other supplemental features. Here are a few specific conclusions regarding the integration of these features with IPTs. The dearth of assessments of the efficacy of individual IPT features represents a gap in the literature. Although the primary focus of this review was not on the quantitative evaluation of the tools, each tool under evaluation was assessed holistically, with no investigation into the efficacy of individual features. It is also challenging to compare the efficacy of various tools in the field because they were assessed under a variety of unique experimental conditions and designs.

References

[1] Omar AlZoubi, Davide Fossati, Barbara Di Eugenio, and Nick Green. [n. d.]. ChiQat-Tutor: An integrated environment for learning recursion. In Proc. of the Second Workshop on AI-supported Education for Computer Science (AIEDCS)(at ITS 2014). Honolulu, HI (2014).
[2] J.R Anderson and E.Skwarecki. 1986. The Automated Tutoring of Introductory Computer Programming. Commun. ACM 29, 9 (Sept. 1986), 842–849. https://doi.org /10.1145/6592.6593
[3] Benjamin S Bloom.1984.The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. Educational researcher 13, 6 (1984), 4–16.
[4] PL Brusilovsky.1992. Intelligent tutor, environment and manual for introductory programming. Educational and Training Technology International 29, 1 (1992), 26–34.
[5] Stephen Cooper, Yoon Jae Nam, and Luo Si. 2012. Initial results of using an intelligent tutoring system with Alice. In Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. ACM, 138–143.
[6] Alex Gerdes, Bastiaan Heeren, Johan Jeuring, and L Thomas van Binsbergen. 2016. Ask-Elle: an adaptable programming tutor for Haskell giving automated feedback. International Journal of Artificial Intelligence in Education (2016), 1–36.
[7] Jay Holland, Antonija Mitrovic, and Brent Martin. 2009. J-LATTE: a Constraint- based Tutor for Java. (2009).
[8] Danial Hooshyar, Rodina Binti Ahmad, Moslem Yousefi, FD Yusop, and S-J Horng. 2015. A flowchart-based intelligent tutoring system for improving problem- solving skills of novice programmers. Journal of Computer Assisted Learning 31, 4 (2015), 345–361.
[9] W Lewis Johnson and Elliot Soloway. 1985. PROUST: Knowledge-based program understanding. IEEE Transactions on Software Engineering 3 (1985), 267–275.
[10] Staffs Keele et al. 2007. Guidelines for performing systematic literature reviews in software engineering. In Technical report, Ver. 2.3 EBSE Technical Report. EBSE. sn.
[11] James A Kulik and JD Fletcher. 2016. Effectiveness of intelligent tutoring systems: a meta-analytic review. Review of Educational Research 86, 1 (2016), 42–78.
[12] Amruth N Kumar. 2005. Generation of problems, answers, grade, and feedback- âĂŤcase study of a fully automated tutor. Journal on Educational Resources in Computing (JERIC) 5, 3 (2005), 3.
[13] H. Chad Lane and Kurt VanLehn. 2003. Coached Program Planning: Dialogue- based Support for Novice Program Design. In Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '03). ACM, New York, NY, USA, 148–152. https://doi.org/10.1145/611892.611955
[14] Nguyen-Thinh Le. 2016. A classification of adaptive feedback in educational systems for programming. Systems 4, 2 (2016), 22.
[15] Nguyen-Thinh Le, Sven Strickroth, Sebastian Gross, and Niels Pinkwart. 2013. A Review of AI-Supported Tutoring Approaches for Learning Programming. In Advanced Computational Methods for Knowledge Engineering, Ngoc Thanh Nguyen, Tien van Do, and Hoai An le Thi (Eds.). Springer International Publishing, Heidelberg, 267–279. http://dx.doi.org/10.1007/978-3-319-00293-4_20

[16] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn East- mond. 2010. The scratch programming language and environment. ACM Trans- actions on Computing Education (TOCE) 10, 4 (2010), 16.
[17] Sally H. Moritz, Glenn D. Blank, Shahida Parvez, and Fang Wei. 2007. A "Design- first" Curriculum and Eclipse&Trade; Tools. J. Comput. Sci. Coll. 22, 3 (Jan. 2007), 51–52. http://dl.acm.org/citation.cfm?id=1181849.1181856
[18] Samy S Abu Naser. 2008. Developing an intelligent tutoring system for students learning to program in C++. Information technology journal 7, 7 (2008), 1055– 1060.
[19] John C Nesbit, Olusola O Adesope, Qing Liu, and Wenting Ma. 2014. How Effective are Intelligent Tutoring Systems in Computer Science Education?. In Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on. IEEE, 99–103.
[20] John C Nesbit, Qing Liu Arita Liu, and Olusola O Adesope. 2015. Work in Progress: Intelligent Tutoring Systems in Computer Science and Software Engineering Education. In
[21] Nelishia Pillay. 2003. Developing intelligent programming tutors for novice programmers. ACM SIGCSE Bulletin 35, 2 (2003), 78–82.
[22] Thomas W Price. 2015. Integrating Intelligent Feedback into Block Programming Environments. In Proceedings of the eleventh annual International Conference on International Computing Education Research. ACM, 275–276.
[23] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. Commun. ACM 52, 11 (2009), 60–67.
[24] Kelly Rivers and Kenneth R Koedinger. 2017. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. International Journal of Artificial Intelligence in Education 27, 1 (2017), 37–64.
[25] Leen-Kiat Soh. 2006. Incorporating an Intelligent Tutoring System into CS1. In Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '06). ACM, New York, NY, USA, 486–490. https://doi.org/10. 1145/1121341.1121494
[26] JS Song, SH Hahn, KY Tak, and JH Kim. 1997. An intelligent tutoring system for introductory C language course. Computers & Education 28, 2 (1997), 93–102.
[27] Gerhard Weber and Peter Brusilovsky. 2001. ELM-ART: An adaptive versatile system for Web-based instruction. International Journal of Artificial Intelligence in Education (IJAIED) 12 (2001), 351–384.
[28] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In Proceedings of the 18th international conference on evaluation and assessment in software engineering. ACM, 38.
[29] Jungsoon Yoo, Chrisila Pettey, Sung Yoo, Judy Hankins, Cen Li, and Suk Seo. [n. d.]. Intelligent Tutoring System for CS-I and II Laboratory. In Proceedings of the 44th Annual Southeast Regional Conference (2006) (ACM-SE 44). ACM, 146–151. https://doi.org/10.1145/1185448.1185482