Comparison of Feedback Strategies for Supporting Programming Learning in Integrated Development Environments (IDEs)

Jarno Coenen(✉), Sebastian Gross, and Niels Pinkwart

Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
{jarno.coenen, sebastian.gross, niels.pinkwart}@hu-berlin.de

Abstract. In this paper we investigate whether providing feedback to learners within an Integrated Development Environment (IDE) helps them write correct programs. Here, we use two approaches: feedback based on stack trace analysis, and feedback based on structural comparisons of a learner program and appropriate sample programs. In order to investigate both approaches, we developed two prototypical extensions for the Eclipse IDE. In a laboratory study, we empirically evaluated the impact of the extensions on learners' performance while they solved programming tasks. The statistical analyses did not reveal any statistically significant effects of the prototype extensions on the performance of the learners, however, the results of a qualitative analysis imply that the provided feedback had at least a marginal impact on the performance of some learners. Also, feedback from the participants confirmed the benefit of providing feedback directly within IDEs.

Keywords: Adaptive feedback · Integrated Development Environment · Java programming

1 Introduction

PROGRAMMING skills are important in both professional and educational contexts. Computer science education below the university level is encouraged by a number of governments in Europe [1, 2], as well as in the USA [3]. University level enrolment in computer science studies is also steadily increasing each year [4]. Learning a programming language can, among other means, be effectively aided by solving smaller programming tasks. This learning method can be assisted by teachers (tutors) and/or tools. Here, the application of computer-supported learning systems can be effective in time and financially sensitive situations. There are several applications of computer-supported learning systems. For instance, Intelligent Tutoring Systems (ITSs) simulate the role of human tutors using models of formalized domain knowledge and student knowledge to provide feedback to learners when solving a given programming problem [5].

Learners often struggle with identifying and fixing errors in their programs. When a stack trace is printed in the console, indicating one or more errors, it is often difficult for novice programmers to understand [6–9]. Integrated Development Environments (IDEs) are a common approach for support, not only for experienced programmers, but also for learners to find and fix errors in their programs by implementing a wide range of features such as syntax highlighting, syntax checks, debugging, and minor semantic and logical checks. However, even though IDEs provide different features to address syntactical and semantical issues, not all of them are appropriate for novices. For instance, debugging a program requires a programmer to have knowledge about the mistake(s) she made. We therefore propose to extend IDEs by implementing features which address both logical and semantic mistakes, as well as syntactical errors in learners' programs, and thus help them develop correct programs.

The outline of this paper is as follows: In Sect. 2, we review related work in the domain of learning programming. We introduce our approach and summarize its design and implementation in Sect. 3. We conducted a lab study where we evaluated the proposed approach. A description of this study and its results are summarized in Sect. 4. Section serves as a conclusion and an outlook on future work.

2 Related Work

There are many approaches which aim to support learners of programming in using computer-supported learning systems. Ample research articles exist which focus on how to support novice learners in understanding stack traces. HelpMeOut [10] is a social recommender system for the Java-based IDE called Processing[1]. This system offers assistance to learners for one specific stack trace at a time, and recommends a bug fix based on a database of bug fixes from other peers. DrScheme [7] is an IDE for the Lisp-based dialect Scheme. Using this system, a learner can select a language level (e.g., "Beginner Student Language") and receive support messages for each of these errors, which serve as reminders for concepts that the learner is expected to know depending on the selected level. Gauntlet [11], on the other hand, is a pre-processing program that is only accessible online through a specific website. Learners can enter their Java programs through a web form after which they can view Gauntlet's outcome. In this outcome, possible syntax errors are then explained in pedagogically suitable phrasing, and some novice level semantic errors are also identified and sufficiently explained. From the given examples, it seems that Gauntlet simply alters compiler messages rather than offer additional error description text with examples.

Aside from stack trace based approaches, there are other systems that aim to help the learner learn programming using AI and adaptive techniques. J-LATTE [12] is an Intelligent Tutoring System (ITS) that provides two modes for teaching a subset of Java. The first mode is purely conceptual, involving the abstract algorithmic solving of problems without writing Java code. The second mode is a coding mode that introduces the use of Java code for the first time within the program. This mode provides on-demand feedback for constraint violation in the learner's program. This feedback is

[1] Processing is both a Java based IDE, as well as a kind of simplified Java programming language for educational purposes. The project website is available at http://www.processing.org/.

designed as text-based note for each violated constraint. JITS [13] is another ITS where a learner can extend fragments of a given program for a given exercise on her own. Here, a learner can request feedback on demand, for which the system provides syntactical help with quick fixes. To some extent, feedback for logical errors based on the exercise specification is also provided. JO-Tutor [14] is an ITS program which focus on how to handle Java object by providing digital learning materials, such as quizzes and sample erroneous programs, as well as a Java editor. The learner then receives feedback if their solution is correct.

All of these approaches have one crucial drawback: the user learns a programming language in an environment that is typically different from real world usage scenarios. When a learner moves to a real IDE, she is required to adapt her techniques accordingly. If the program simply alters errors for a learner, she is then also required to relearn the exceptions of the programming language in question. If instead a real IDE is used, with error messages compliant to the programming language standard, no transition to the real world usage is needed. Therefore, it is worth developing features that are seamlessly integrated into IDEs in order to provide a variety of types of feedback to learners. These features would be particularly helpful in providing feedback for solving semantic and logical errors, as even these basic programming skills are often insufficiently understood by learners [6–9]. This problem stems from the fact that erroneous but syntactically correct programs can be compiled and partly executed, but not necessarily produce the desired outcome. Here, feedback on semantic and logical errors could provide meaningful information to assist the learner in overcoming this initial hurdle.

3 Approach: Design and Implementation

As stated, our goal is to extend IDEs to provide meaningful feedback to learners in order to help them develop syntactically and semantically correct programs. We chose Eclipse IDE as the host system above other alternative IDEs (e.g. NetBeans[2], and IntelliJ IDEA[3]), because it is more widely used among programmers [15], [16][4], [17, 18], Due to its popularity in professional software development and computer science education at university level, we focused on supporting Java programmers.

In order to help learners find and fix both syntactical and logical/semantical mistakes, we developed two extensions for the Eclipse IDE: (i) in order to complete the programming cycle, a stack trace analysis provides feedback to a learner, and (ii) based on comparison of learner's solution and appropriate sample solutions, we highlight and contrast both the learner's solution and (steps of) a sample solution. The proposed design tries to be integrated seamlessly into the IDE, and guide the learner on demand through the available feedback.

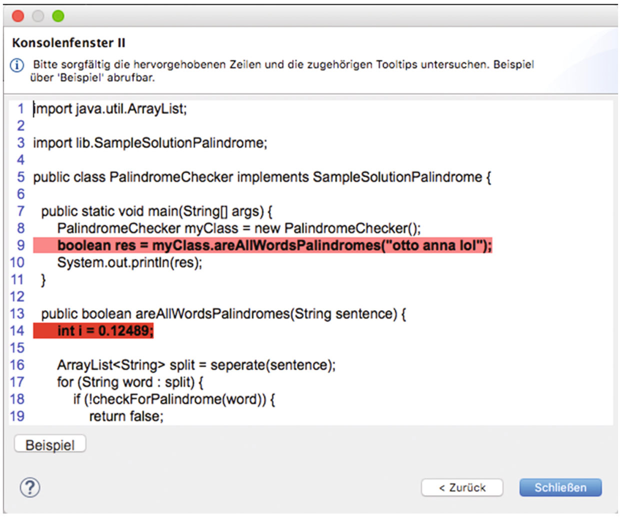
[2] NetBeans is an open source IDE supporting, amongst others, Java (see https://netbeans.org/).

[3] IntelliJ IDEA is an Apache 2 licensed IDE for non-commercial use supporting, amongst others, Java (see https://www.jetbrains.com/idea/).

[4] In this survey in 2016, Eclipse is ranked second to IntelliJ for the first time in their survey, however only by a marginal 5%.

3.1 Feedback Based on Stack Trace Analysis

The first extension (referred to as EXT_1) is based on analyzing stack trace exceptions from console output. When an exception occurs while executing a program, the extension parses the stack trace to retrieve the exception signature and the associated lines of code. The signature is then matched against the exceptions previously defined in a database. If the exception has been defined in the database, then the type of exception is known. In this case, three types of exceptions are possible: a runtime exception, a syntax exception, or a logic error. Using the information from the database, the extension then highlights related lines of the code in a window (as illustrated in Fig. 1). Each type of exception has its own distinctive background color, and unknown exceptions share one default background color. The code line throwing the exception is highlighted in a bright color, and subsequent associated code lines in a lighter shade. Highlighted lines also provide additional tooltips with notes from the database, and a button to request examples. Clicking on this button produces a pop-up dialog consecutively listing all examples defined in the database for this specific exception.



Konsolenfenster II

Bitte sorgfältig die hervorgehobenen Zeilen und die zugehörigen Tooltips untersuchen. Beispiel über 'Beispiel' abrufbar.

```
import java.util.ArrayList;

import lib.SampleSolutionPalindrome;

public class PalindromeChecker implements SampleSolutionPalindrome {

  public static void main(String[] args) {
    PalindromeChecker myClass = new PalindromeChecker();
    boolean res = myClass.areAllWordsPalindromes("otto anna lol");
    System.out.println(res);
  }

  public boolean areAllWordsPalindromes(String sentence) {
    int i = 0.12489;

    ArrayList<String> split = seperate(sentence);
    for (String word : split) {
      if (!checkForPalindrome(word)) {
        return false;
```

Fig. 1. View on stack trace analysis details.

For providing feedback to learners based on stack trace analysis, we composed a database of typical errors in Java. Since a variety of entries were possible, as Java offers a vast amount of possible errors and exceptions, we reviewed several papers to identify those errors which are typically made by novice programmers (see [19–25]). Our research produced five papers (referenced in Table 1) that included enough raw data to compose the following table, which consolidates each paper's top 10 most commonly made errors. Overall, the literature research revealed 9 errors (see Table 1), which we added to the stack trace database. We also added the additional errors ‘NullPointerException’, ‘ArrayIndexOutOfBoundsException’, ‘cannot cast from … to …’, and ‘cannot make a static reference to the non-static’.

Table 1. Java errors typically made by learners.

Error message	Papers
Cannot resolve symbol	[19, 20, 22–24]
; expected	[19, 20, 22–24]
Illegal start of expression	[19, 20, 22–24]
Bracket-expected (i.e. ‘(‘,’)’, ‘{‘,’}’, ‘[‘or’]’)	[19][a], [20, 22–24]
Class or interface expected	[19, 20, 22]
<identifier> expected	[19, 22, 24]
Incompatible types	[19, 20, 22]
Unknown-method	[20, 22, 23]
Else without if	[22, 24]
Not a statement	[19, 24]
Reached EOF while parsing	[23, 24]
Unknown-class	[20, 23]
.class expected	[20]
Missing return statement	[22]
Method cannot be applied to parameter types	[23]

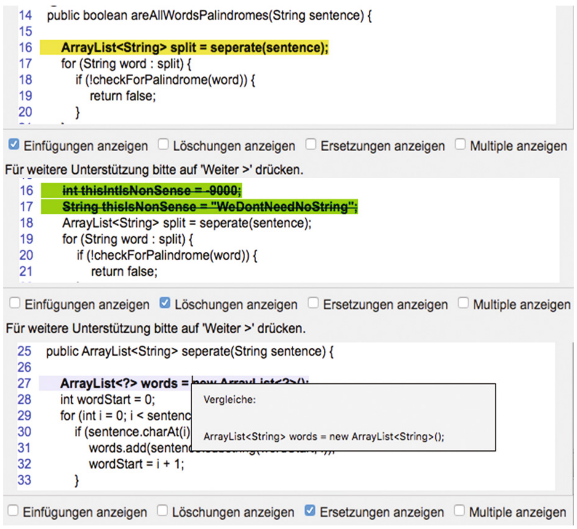
[a]These papers offer more detail, i.e., they list the errors separately, but are listed here only once for better comparability.

3.2 Feedback Based on Comparisons to Sample Programs

The second extension (referred to as EXT_2) implements the example-based learning approach and uses feedback strategies whose benefits were evaluated in previous studies [26, 27]. Technically, this approach is based on the “TCS” toolbox[5] which analyzes Java programs and calculates a dissimilarity score for two Java programs [28]. The toolbox uses advanced machine learning techniques. Although the toolbox identifies the most similar sample solution, the extension can opt to choose the next sequential (more advanced) sample solution step for alignment and eventual presentation. If a more advanced sample solution step exists that is similar to both the originally identified sample solution step and to the learner solution, but not to the final sample solution, then this more advanced sample solution is used for alignment. This procedure is meant facilitate the learner in making further progress.

We implemented a view (illustrated in Fig. 2) which displays an instruction message prompting a learner to carefully examine highlighted code areas, and a text area containing the learner’s current Java editor code (see Fig. 2). Below the text area, there

[5] For more details, see the homepage of the toolbox, available here: https://openresearch.cit-ec.de/projects/tcs.



```
public boolean areAllWordsPalindromes(String sentence) {

    ArrayList<String> split = seperate(sentence);
    for (String word : split) {
        if (!checkForPalindrome(word)) {
            return false;
        }
```

[✓ Einfügungen anzeigen] Löschungen anzeigen Ersetzungen anzeigen Multiple anzeigen

Für weitere Unterstützung bitte auf 'Weiter >' drücken.

```
    ~~int thisIntIsNonSense = -9000;~~
    ~~String thisIsNonSense = "WeDontNeedNoString";~~
    ArrayList<String> split = seperate(sentence);
    for (String word : split) {
        if (!checkForPalindrome(word)) {
            return false;
```

Einfügungen anzeigen [✓ Löschungen anzeigen] Ersetzungen anzeigen Multiple anzeigen

Für weitere Unterstützung bitte auf 'Weiter >' drücken.

```
public ArrayList<String> seperate(String sentence) {

    ArrayList<?> words = new ArrayList<?>();
    int wordStart = 0;
    for (int i = 0; i < sentenc
        if (sentence.charAt(i)
            words.add(sentenc
            wordStart = i + 1;
        }
```

Vergleiche:

`ArrayList<String> words = new ArrayList<String>();`

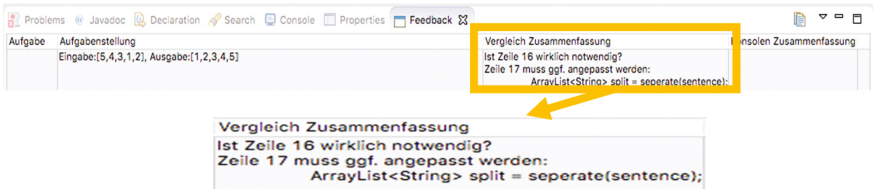
Einfügungen anzeigen Löschungen anzeigen [✓ Ersetzungen anzeigen] Multiple anzeigen

Fig. 2. View on feedback based on structural comparisons of learner's program and an appropriate sample program.

are the following checkboxes: (1) show insertions, (2) show deletions, (3) show replacements, and (4) show multiple. These checkboxes resemble the operations from the toolbox analysis. By selecting a checkbox, the rows in question are highlighted in different colors (depending on the corresponding operation). 'Show insertions' indicates that the highlighted code is missing in the learner solution. 'Show deletions' indicates that the highlighted code in the learner solution is possibly redundant (i.e. the line of learner's program is missing in the sample solution). 'Show replacements' operation indicates that the highlighted code area is marginally different from the sample solution (i.e. a minor alteration would transfer the learner's solution to be exactly the correspondent sample solution code line). 'Show multiple' indicates that two or more operations can be applied to the corresponding code area. Based on the findings through the toolbox, the extension retrieves lines of code from a sample program for all insertion operations. These code lines are then inserted into the learner solution and highlighted when the corresponding checkbox is selected. Additionally, highlighted code areas contain tooltip notes, which show the corresponding code line of the sample program.

3.3 Feedback Summary Panel

In addition to the two standalone views on feedback details, we also implemented a panel (illustrated in Fig. 3) which provides a summary of requested feedbacks and enables users to recover previous views on feedback details.



Aufgabe	Aufgabenstellung	Vergleich Zusammenfassung	nsolen Zusammenfassung
	Eingabe:[5,4,3,1,2], Ausgabe:[1,2,3,4,5]	Ist Zeile 16 wirklich notwendig? Zeile 17 muss ggf. angepasst werden: ArrayList<String> split = seperate(sentence);	

```
Vergleich Zusammenfassung
Ist Zeile 16 wirklich notwendig?
Zeile 17 muss ggf. angepasst werden:
          ArrayList<String> split = seperate(sentence);
```

Fig. 3. Panel for brief summaries of previous feedback requests.

4 Evaluation

In September 2016, we conducted a laboratory study to investigate the impact of the prototypically implemented feedback features (as described in Sect. 3) on users' performances. Overall, 23 participants took part in the study. All participants were required to have some prior Java programming knowledge, the extent of which was assessed by a pretest. The primary focus and consideration of the study was to determine the effectiveness of the two feedback features in helping the learner develop syntactically and semantically correct programs.

4.1 Hypotheses

In order to evaluate the research question of whether providing feedback directly in an IDE helps learners develop correct programs, we stated the following hypotheses:

1. The extensions positively influence users' performance in comparison to those settings where users do not have any kind help beyond Eclipse features. We, thus, expect a statistically significant difference between performances achieved when feedback features (as described in Sect. 3) are available in comparison to those performances learners achieved without using the feedback features.
2. The feedback features based on stack traces (as described in Sect. 3.1), and based on structural comparisons to sample solutions (as described in Sect. 3.2) differ in their impact on learners' performance in solving a programming task. We, thus, expect a statistically significant difference in performance between these learners who use feedback feature based on stack trace analysis in comparison to those learners who use feedback feature based on comparisons to sample programs.

Hypotheses 1 directly addresses the research question. Hypothesis 2 assesses whether EXT_1 or EXT_2 influences the performance of the learners to a similar or different extent.

4.2 Study Design

In order to measure the impact of the feedback features (as described in Sect. 3) independently from each other, we randomly assigned participants to two groups. In both groups, participants were first asked to solve a programming task (referred to as

TASK_1) without any kind of assistance from the Eclipse IDE. Then, participants were asked to solve another programming task (referred to as TASK_2) using the Eclipse IDE with one of the proposed feedback features. Here, we randomized the solving order of the programming tasks in order to lessen the impact of possible inequalities between the tasks. The first group could use feedback features based on stack traces (EXT_1), and the second group could use feedback features based on structural comparisons to sample solutions (EXT_2). As programming problems, we designed two programming tasks (referred to as FibSum80 and MergeSort, respectively). Programming task FibSum80 required participants to calculate the sum of the first 80 Fibonacci numbers. Programming task MergeSort required them to sort an integer array using the recursive merge sort algorithm. In advance, we created sample programs for both programming tasks, collecting sample solutions from multiple websites and amending them to fit the exact exercise description, splitting each sample solution into multiple steps. For each programming task, participants had 40 min for completion.

To set equal grounds among the participants regarding basic Java syntax, all participants were provided with a Java cheat sheet consisting of the basic control flow elements of Java. Participants were also provided with a hard-copy tutorial demonstrating the feedback feature (depending on the group they belonged to) and how to use it. Beyond that, they were not allowed to use any kind of help (e.g., web search engines).

For analysis purposes, we logged selected types of information while a user is using the extension. This information included the amount of extension accesses, the time spent in each extension, the source code at the time of accessing the extension, and the final programs.

4.3 Results

All of the learners' final programs were rated on a scale from 0 to 100%, considering three criteria. The results of a Shapiro-Wilk test of normality for each dataset for hypothesis 1 indicated that the datasets were not normally distributed. We thus used a nonparametric Wilcoxon Signed Ranks Test. This test did not reveal any statistically significant difference between the two extensions ($p = .549$, $Z = -.599$ based on positive ranks). Hence, the hypothesis could not be confirmed by this data.

Table 2. Table of ranks used by the Wilcoxon Signed Ranks Test. Of 13 participants, 6 experienced decreased performance while using EXT_2, 4 did not experience alterations in performance when using EXT_2, 3 experienced increased performance.

		N	Mean rank	Sum of ranks
Performance task 2 with extension – performance task 1 without extension	Negative ranks	6	4.58	27.50
	Positive ranks	3	5.83	17.50
	Ties	4		
	Total	13		

The dataset used for analyzing hypothesis 2 was not normally distributed. Therefore, we performed a nonparametric Mann-Whitney U Test. The results did not reveal any statistically significant difference in performance, regardless of whether participants used the extension or not ($p = .330$, $Z = -1.099$ not corrected for ties). Hence, we could not reject the null hypothesis "both samples are from the same population".

The mean rank reveals a moderate, but statistically non-significant tendency towards decreasing performance with the use of the feedback features (see Table 2).

Table 3. Qualitative analysis results of the participants that have used the extension at least once.

Participant	Extension	Pre-test score	Task 2	Score task 1	Score task 2	Qualitative analysis results
1	EXT_1	4.3	FibSum80	0%	0%	No help
2	EXT_1	6.7	FibSum80	0%	100%	Little help
3	EXT_1	5.7	MergeSort	66%	33%	Little help
4	EXT_1	5.4	MergeSort	66%	33%	Little Help
5	EXT_2	3.2	FibSum80	33%	66%	No help
6	EXT_2	3.7	FibSum80	44%	100%	No help
7	EXT_2	6.3	MergeSort	100%	33%	No help
8	EXT_2	5.8	MergeSort	100%	0%	Little help
9	EXT_2	5	MergeSort	66%	0%	No help
10	EXT_2	4.2	MergeSort	33%	0%	No help
11	EXT_2	3.2	FibSum80	0%	0%	Little help
12	EXT_2	2.4	FibSum80	0%	0%	No help
13	EXT_2	4.4	MergeSort	0%	0%	No help

Measured in qualitative terms, the impact of the extension was low, as the participants' solutions mostly disregarded the provided feedback (see Table 3). The qualitative analysis was based on the following criteria: (1) if there is no indication of transferred feedback, then result to 'no help,' (2) if there is some indication of transferred feedback, then result to 'little help.' Particularly those with a low pre-test score and very primitive learner solutions at the point of requesting feedback had difficulties using the provided feedback effectively in their solutions.

5 Conclusion and Future Work

The data collected from the laboratory study was, in the end, not sufficient enough to confirm the stated hypotheses. This could possibly be attributed to the relatively small number of participants, the potentially overshadowing influence of the difference between the two programming tasks, and, in the case of EXT_2, the additional factor of the choice of sample solution steps. Concerning the choice of sample solution steps, the qualitative analysis showed that inexperienced participants accessed feedback, but did not manage to improve their solution with the provided feedback. The analysis involved the comparison of the learner solution with the presented sample solution step. In cases where the presented sample solution had a different approach than the learner's, no transfer of feedback was noticeable. This implies that the sample solution steps the feedback was based on missed some approaches and/or may have offered too little information to be of value to a novice programmer. Here, fine-grained sample solution steps might be more beneficial for novices.

According to the survey completed by participants at the end of the study, two participants who used EXT_2 liked the idea/concept of the extension in general. A third participant reviewing EXT_2 liked its clear structuring and uncomplicated integration into Eclipse, as well as the fact that his code was directly available in the extension for analysis. This supports the idea that the participants, in their roles as learners, would find an IDE extension such as the one created for this project useful for learning programming. If we consider this feedback in combination with the positive overall usability rating of the extensions, we can determine the idea behind this research is valid, and that this paper makes important contributions to a largely unexplored theme. The main drawback of EXT_2 named by participants was that the feedback was unspecific and misleading, particularly when the difference between the learner program and the sample program was considerable. The claim that the extension is nonspecific is to some extent true. The extension specifically highlights code areas it has identified as divergent in some way, but it does not offer a text-based report of the finding as further feedback. This would require the extension to conduct a far more comprehensive analysis of each finding, and possibly generate a report for a great variety of different scenarios.

For future usage, the stack trace database for EXT_1 should be extended in this way, so that a wider range of exceptions are considered. In order to improve the usability of both extensions, it may be necessary to examine the color scheme applied, as one participant found it not intuitive. A possible new feature, which was also suggested by two participants of the study, would be feedback for the Eclipse debugger. This feedback could include the automated setting of breakpoints, feedback on the usage of the Eclipse debugger, and additional feedback on specific variable contents that could have caused errors.

Acknowledgement. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under the grant "DynaFIT – Learning Dynamic Feedback in Intelligent Tutoring Systems" (PI 764/6-2).

References

1. European Commission: Coding - the 21st century skill 6 July 2016. https://ec.europa.eu/digital-single-market/coding-21st-century-skill. Accessed 4 Nov 2016
2. Balanskat, A., Engelhardt, K.: Computing Our Future - Computer Programming and Coding - Priorities, School Curricula and Initiatives Across Europe. European Schoolnet, Brussels (2015)

3. The White House - Office of the Press Secretary: Remarks of President Barack Obama – State of the Union Address as Delivered, 13 January 2016. https://www.whitehouse.gov/the-press-office/2016/01/12/remarks-president-barack-obama-%E2%80%93-prepared-delivery-state-union-address. Accessed 04 Nov 2016
4. Zweben, S., Bizot, B.: 2015 taulbee survey: continued booming undergraduate CS enrollment; doctoral degree production dips slightly. Comput. Res. News **28**(5), 2–60 (2016)
5. Self, J.: The defining characteristics of intelligent tutoring systems research: ITS care, precisely. Int. J. Artif. Intell. Educ. (IJAIED) **10**, 350–364 (1998)
6. Nienaltowski, M., Pedroni, M., Meyer, B.: Compiler error messages: what can help novices? In: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, pp. 168–172 (2008)
7. Marceau, G., Fisler, K., Krishnamurthi, S.: Mind your language: on novices' interactions with error messages. In: Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2011), pp. 3–18 (2011)
8. Hristova, M., Misra, A., Rutter, M., Mercuri, R.: Identifying and correcting Java programming errors for introductory computer science students. In: Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2003), pp. 153–156 (2003)
9. Murphy, C., Kim, E., Kaiser, G., Cannon, A.: Backstop: a tool for debugging runtime errors. In: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2008), pp. 173–177 (2008)
10. Hartmann, B., MacDougall, D., Brandt, J., Klemmer, S.: What would other programmers do: suggesting solutions to error messages. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2010), pp. 1019–1028 (2010)
11. Flowers, T., Carver, C.A., Jackson, J.: Empowering students and building confidence in novice programmers through Gauntlet. In: 34th Annual Frontiers in Education, FIE 2004, pp. T3H10–T3H13 (2004)
12. Holland, J., Mitrovic, A., Martin, B.: J-LATTE: a constraint-based tutor for Java. In: Proceedings of the 17th International Conference on Computers in Education, pp. 142–146 (2009)
13. Sykes, E.: Design, development and evaluation of the Java intelligent tutoring system. Tech. Inst. Cogn. Learn. **8**, 25–65 (2010)
14. Abu-Naser, S., Ahmed, A., Al-Masri, N., Deeb, A., Moshtaha, E., Abu-Lamdy, M.: An intelligent tutoring system for learning Java objects. Int. J. Artif. Intell. Appl. (IJAIA) **2**(2), 68–77 (2011)
15. Codeanywhere Inc.: Most Popular Desktop IDEs & Code Editors in 2014, 13 January 2015. https://blog.codeanywhere.com/most-popular-ides-code-editors/. Accessed 04 Nov 2016
16. Maple, S.: Java Tools and Technologies Landscape Report 2016, 14 July 2016. http://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016/. Accessed 4 Nov 2016
17. Biradar, M.: Popularity of Programming Languages, 28 July 2015. https://maheshbiradar.wordpress.com/2015/07/28/popularity-of-programming-language/. Accessed 4 Nov 2016
18. Stack Exchange Inc.: Developer Survey Results 2016, March 2016. http://stackoverflow.com/research/developer-survey-2016#technology-development-environments. Accessed 5 Nov 2016
19. Jackson, J., Cobb, M., Carver, C.: Identifying top Java errors for novice programmers. In: Proceedings - Frontiers in Education Conference, p. T4C (2005)
20. Jadud, J.: A first look at novice compilation behaviour using BlueJ. Comput. Sci. Educ. **15**, 25–40 (2005)

21. Altadmri, A., Brown, N.: 37 million compilations: investigating novice programming mistakes in large-scale student data. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 522–527 (2015)
22. Tabanao, E., Rodrigo, M., Jadud, M.: Predicting at-risk novice Java programmers through the analysis of online protocols. In: Proceedings of the Seventh International Workshop on Computing Education Research (ICER 2011), pp. 85–92 (2011)
23. McCall, D., Kölling, M.: Meaningful categorisation of novice programmer errors. In: Frontiers in Education Conference, pp. 2589–2596 (2014)
24. Becker, B.: An effective approach to enhancing compiler error messages. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE 2016), pp. 126–131 (2016)
25. Denny, P., Luxton-Reilly, A., Tempero, E.: All syntax errors are not equal. In: Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2012), pp. 75–80 (2012)
26. Gross, S., Mokbel, B., Hammer, B., Pinkwart, N.: Learning feedback in intelligent tutoring systems. KI - Künstliche Intell. **29**(4), 413–418 (2015)
27. Gross, S., Pinkwart, N.: How do learners behave in help-seeking when given a choice? Int. J. Artif. Intell. Educ. **9112**, 600–603 (2015)
28. Mokbel, B., Gross, S., Paassen, B., Pinkwart, N., Hammer, B.: Domain-independent proximity measures in intelligent tutoring systems. In: Proceedings of the 6th International Conference on Educational Data Mining (EDM), pp. 334–335 (2013)