 DOI: 10.1109/ICALT.2015.75

Towards an Integrative Learning Environment for Java Programming

Sebastian Gross and Niels Pinkwart

Humboldt-Universität zu Berlin, Department of Computer Science
Unter den Linden 6, 10099 Berlin, Germany
{sebastian.gross,niels.pinkwart}@hu-berlin.de

***Abstract*—Learning programming can be a challenging task for students that not only requires them to acquire knowledge but also to make use of their knowledge in solving real-world problems. In this paper, we introduce an intelligent, adaptive and adaptable learning environment for Java programming called FIT Java Tutor. The learning environment integrates several pedagogical approaches in order to help learners learn programming considering individual needs. For testing purposes, we prepared a set of learning resources consisting of video tutorials, programming tasks, quizzes and multiple-choice tests, and deployed the learning system in an introductory programming class at Humboldt-Universität zu Berlin. Based on experiences gained from this setup, we derived three research questions for investigation in future studies.**

***Index Terms*—learning environment, integrative, adaptive tutoring, Java, programming**

I. Introduction

Programming is a useful skill and teaching basics of programming is part of many curricula in universities and higher education. Learning a programming language, however, cannot be approached theoretically only but requires a lot of practice in order to develop technical expertise regarding programming concepts as well as algorithmic thinking for mapping real-world problems to program code structures. Several studies (e.g. [1], [2]) have shown that many students already experience serious difficulties with the basics of a programming language. In order to deal with this issue, many academic institutions make use of computer-supported learning systems to facilitate the process of learning programming and to give students an additional tool to learn whenever and wherever they want to. In this paper, we introduce an integrative learning system for Java programming that considers learner's individual needs in learning, and provides instructional support for problem-solving.

The remainder of this paper is organized as follows: First, in Section II, we review the state of the art of computer-supported learning systems for programming and derive requirements for such systems. In Section III, we then describe our system in more detail. In Section IV, we outline our experiences from an introductory programming class in which our system was used, and we derive and discuss research questions for investigation in future studies. Finally, in Section V, we conclude and give an outlook on future work.

II. Computer-supported Learning Environments for Programming

There are several approaches for computer-supported teaching and learning of programming languages. These approaches can be distinguished by the pedagogical principles that are implemented to support learning.

Learning Management Systems (LMS) help educators organize learning resources, and enable them to provide learning tools to learners. For instance, the GREWPtool has been integrated into the Moodle LMS to enable students to collaborate as they learn Java programming. An evaluation of this approach has shown that students using this collaborative tool have significantly higher success rates than those who learned in a traditional way [3].

Adaptive learning systems (ALE) are personalized environments that aim at tailoring computer assisted learning process to learner's needs. For instance, the JavaGuide provides adaptive navigation support to prevent learners from selecting too simple or too complicated problems [4]. Another approach for adaptive support is to recommend learners appropriate content. For instance, Proteus is a tutoring system for Java programming that uses principles of learning style identification in order to provide content recommendations for course personalization [5].

Exploratory learning environments (ELE) help learners acquire knowledge allowing or scaffolding them to freely explore the environment. For instance, Hohl and colleagues [6] proposed a hypertext system that facilitates exploratory learning of Common Lisp using a hyperspace of topics. A more recent approach that supports exploratory learning is the 'FLIP Learning' system for learning JavaScript [7].

Intelligent Tutoring Systems (ITS) are a specific type of ALEs. They use AI techniques in order to instruct learners, and thus facilitate learning in many fields such as algebra [8] or intercultural competence [9]. In the domain of programming, Le and colleagues [10] reviewed AI-supported tutoring approaches and classified them into six categories: example-based, simulation-based, collaboration-based, dialogue-based, program analysis-based, and feedback-based approaches. The NavEx tutor, for instance, pursues the example-based learning approach in order to give learners explanations on how to solve programming problems by providing annotated program code examples instead of bare solutions [11].

As stated above, there are several approaches to support learning programming using different pedagogical principles. While some systems support learning by providing pedagogically structured learning resources such as LMSs, or support exploratory learning such as ELEs, these systems, however, may have no adaptive and intelligent support (as ITSs provide if learners got stuck in problem-solving), or they do not support interactions among groups of learners (as it is supported by collaborative programming tools). Furthermore, systems rarely consider how to support learner groups containing students of different expertise, from very beginners to more advanced or expert learners.

Based on our review of the state of the art of computer-supported learning approaches, we determined the following requirements to be met by learning supporting systems for programming in order to support heterogeneous groups of learners (e.g. from very beginners to advanced). Hence, we argue that a computer-supported learning environment is supposed to

- R1 instruct a learner if she got stuck in problem-solving,
- R2 provide a learner with pedagogically structured learning resources,
- R3 enable a learner to explore the environment taking into account her (lack of) knowledge,
- R4 enable a learner to adapt the environment to her individual needs, and
- R5 enable a learner to interact with other learners in collaborative activities.

In this paper, we propose an integrative approach that addresses these requirements. We therefore implemented an intelligent, adaptive and adaptable learning environment for Java programming that integrated several pedagogical principles.

III. System Design and Implementation

In previous work [12], we proposed and implemented an ITS middleware architecture which facilitates the construction of ITSs. In our approach, we replaced domain models typically used in ITSs for providing feedback to learners by solution spaces. We developed machine learning techniques [13] that are capable of automatically inferring structures from data sets consisting of student solution attempts and sample solutions. Based on these structured solution spaces, we proposed feedback provision strategies [14] that employ example-based learning methods comparing student solution attempts to appropriate sample solutions. We implemented our approach in a web-based programming environment that provides feedback to learners while solving programming problems.

In recent work, we elaborated the web-based programming environment to such an extent that the system does not only implement instructional support (requirement R1) but also integrates several pedagogical strategies in order to consider individual learner's needs. We implemented our approach in the FIT Java Tutor, an integrative learning environment for Java programming. Technically, it is a web-based system (running in modern web browsers) that uses standard web technologies such as HTML5, JavaScript and SVG: the JavaServer Faces (JSF) framework is used to create dynamic web pages, and JavaScript libraries such as JQuery and D3.js [15] are used to implement dynamic behavior on client side and to generate dynamic and interactive visualizations based on SVG. For (a)synchronous, secure and reliable communication between client and server, the FIT Java Tutor uses HTTP, AJAX and web socket connections over SSL. In order to provide instructional support to learners, the learning environment communicates with the above mentioned ITS middleware via web services.

The client application of the FIT Java Tutor basically comprises two components: the programming environment component enables learners to directly solve programming problems by writing Java program code, and the resource presentation component provides learners with an overview of resources they can use for learning.

A. Programming Environment Component

Problem-based learning (PBL) is an instructional approach that requires students to solve an authentic problem and aims at enhancing students' knowledge, problem-solving and self-directed learning skills while solving a given problem [16]. Since learning programming requires learners not only to *know* basics of a programming language but also to be able to apply their knowledge, solving typical problems is essential for learning programming. In computer-supported learning programming, Kumar, for instance, investigated the effectiveness of a problem-solving tutor, and has shown that using this tutor improves students' programming skills [17]. The FIT Java Tutor supports PBL by integrating a programming environment (illustrated in Fig. 1) which enables users to solve programming problems by writing, compiling and executing Java programs. It supports code-highlighting as implemented in the popular Eclipse development environment. Compiler and interpreter outputs are displayed, thus users have access to standard error messages generated while compiling/executing a Java program. Furthermore, learners can request feedback on their programs. These requests are then forwarded to and processed by the ITS middleware which provides feedback, and thus instructs learners in problem-solving (R1) based on sample solutions (as illustrated in the right panel in Fig. 1).

To give learners an indication of their progress in learning, the programming environment provides four indicator lamps reflecting syntactic and semantic correctness of a program in respect of a given problem: the first indicator lamp reflects whether a program can be compiled, the second reflects whether a program can be executed, the third indicates whether the program terminates in some predefined time, and the fourth indicates whether the program's output is the expected one.

Learner's problem-solving processes are often characterized by making and correcting errors. To support learners in stepping forward and backward while solving a given problem, the programming environment provides an interactive visualization of a graph (as illustrated in Fig. 2) that represents steps in problem-solving. This visualization enables a learner to go back to a previous program version (e.g. if she wants to apply another problem-solving strategy), or to restore a more advanced version.

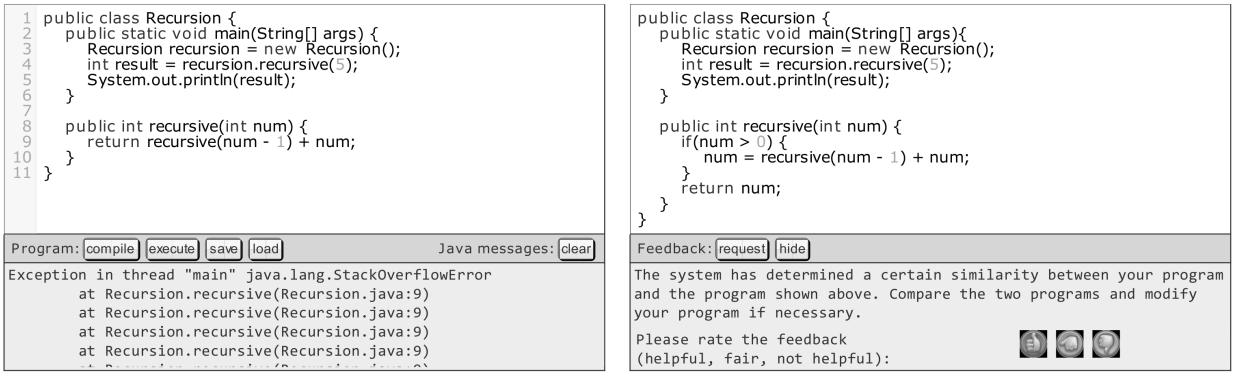

Fig. 1: Screenshot of the programming environment component integrated in the FIT Java Tutor.

B. Resource Presentation Component

The purpose of the second component is to provide users with an overview of learning resources they can utilize for learning. SCORM, LOM, and the Tin Can API help modelling resources for use in learning systems (e.g. in the domain of programming [18]), and thus ease the sharing of learning objects between different systems. These standards, however, also impose special requirements for presenting contents of learning objects. Since, in our approach, we focused on how to design views on resources in order to address different pedagogical requirements, we simplified the representation by designing learning objects as digital artifacts containing text and multimedia data. Here, an artifact can be related to a set of tags that describe its content. We implemented several views on these artifacts in order to address the requirements defined in Sec. II.

1) Course-oriented View: The presentation of learning objects in computer-supported learning systems often relies on pedagogical concepts that reflect learning goals. Some LMSs, for instance, provide courses and make use of hypermedia to structure learning objects taking into account different learning objectives. In the FIT Java Tutor, we implemented a course view that visualizes artifacts structured into course units representing concepts in Java programming such as "Recursion" (illustrated in Fig. 3). Since a course often consist of several units, the course view is also able to visualize dependencies between units implementing a logical sequence of learning objectives (e.g. the concept of backtracking requires the concept of recursion). Addressing R2, this view aims at providing learners with pedagogically structured resources.

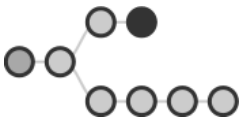

Fig. 2: Graph of program versions where nodes represent steps in problem-solving. Users can restore a previous program version by clicking on the corresponding node.

2) Problem-oriented View: As stated in Sec. III-A, PBL is a promising approach to support learning programming. To give learners an overview of all available programming problems, we implemented a dynamic task list where each entry of the list includes a description of the task, and a colored bar indicating the difficulty of the task (ranging from green for easy tasks to red for hard tasks). The difficulty of a task is calculated from learners' and experts' ratings, respectively. Addressing R2, this view aims at enabling a learner to identify appropriate problems that correspond to her level of knowledge.

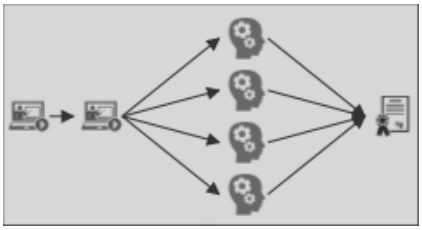

Fig. 3: Course unit "Recursion" composed of two consecutive video tutorials, followed by four programming tasks, and a final test.

3) Resource-oriented View: In the FIT Java Tutor, we implemented a dynamic visualization (based on a force-directed layout) of artifacts represented as resource space (illustrated in Fig. 4) that emphasizes artifacts, relations and learners' contributions. In contrast to the course view which requires educators to manually structure artifacts into course units, the resource space view visualizes structures based on logical relations (e.g. a sample solution is related to a programming task) and semantic proximities. As mentioned above, an artifact can be related to a set of tags that describe its content. To calculate semantic proximities among a set of artifacts, we use a similarity measure based on tag sets proposed by Zhou and colleagues [19]. These proximities are then mapped to (invisible) edges between artifacts that define the visual structure of the resource space. To consider individual needs of learners, users can apply filters to the resource space in order to hide or display artifacts considering quality (e.g. to display highly rated artifacts only), artifact type (e.g. to display programming tasks only), and related tags (e.g. to display artifacts related to "recursion" only). Addressing R3, this view aims at supporting learners as they explore the environment. The structuring of learning resources into semantically similar

Fig. 4: Excerpt from the resource space view: visualization of artifacts (represented by nodes) and their logical relations (represented by edges). The distance between two artifacts reflects their semantic proximity.

Fig. 5: Excerpt from the tag cloud: visualization of keywords which describe concepts in Java programming. The font size of a tag indicates how many artifacts are related to the corresponding tag.

clusters has the goal of allowing learner to identify knowledge (represented by sets of artifacts) she has already acquired and she is still lacking, respectively. Addressing R4, the filtering options aims at enabling a learner to adapt the resource space view to her individual needs.

4) Concept-oriented View: There are several approaches (such as semantic networks) to model relations among data sets of knowledge. Here, concept maps are a common approach to visualize these relations. In the FIT Java Tutor, we employed this approach by implementing a tag cloud (illustrated in Fig. 5) that reflects concepts in Java programming. By clicking on a tag, related artifacts are displayed in the resource space (see Sec. III-B3). When selecting two or more tags, learner can see concepts that are interrelated within the resource space.

C. Community-oriented Features

The FIT Java Tutor implements awareness features that notify a user about learning activities of other users (e.g. if a user has worked on a programming problem). Addressing R5, the system also provides several community-oriented and cooperative features that enable a learner (I) to rate the difficulty of programming problems, (II) to rate and to comment on artifacts, (III) to add tags to artifacts, (IV) to create artifacts (e.g. a programming problem a learner is unable to solve on her own), (V) to share her own solution attempts for programming problems with other users, and (VI) to review solution attempts that other users have shared before.

In summary, these features aim at engaging a learner to enrich learning resources by metadata (such as ratings, comments and tags) and, thus, to reflect on them, to extend educational data sets by new learner created resources, and thus to support the emergence of learning communities.

IV. Experiences from an Introductory Programming Course for Java

In the winter term 2014/15, we deployed the FIT Java Tutor in an introductory programming class at Humboldt-Universität zu Berlin which was attended by about 400 students. In addition to lecture and exercise sessions, the use of the FIT Java Tutor was voluntary for students. In advance, a student assistant with teaching experience (supported by two experienced Java programmers) developed an online course for Java beginners composed of 12 course units covering basic concepts such as conditional statements or loops, and also more advanced topics such as recursion or backtracking. He also defined a logical sequence between these units. For each course unit the student assistant also identified and prepared learning resources resulting in a set of artifacts consisting of 25 video tutorials (hosted on YouTube and embedded using the YouTube Video Player), 39 programming tasks (and sample solutions for feedback provision), 4 quizzes (each consisting of a program and several questions about its behavior while executing), and 12 multiple choice tests. Finally, he defined a logical sequence (with respect to learning objectives) of artifacts within each unit. This course data is visualized in the course view (see Sec. III-B1). The set of learning objects represents the artifacts in the resource space (see Sec. III-B3). From this data set, the learning system automatically extracted all programming tasks to be displayed in the task list (see Sec. III-B2). In addition, the expert tagged each artifact with Java related keywords. These tags were the basis of the tag cloud (see Sec. III-B4). Overall, the preparation of the learning resources summed up to an effort of about 100 hours whereof 20 hours were spent on designing the course, 70 hours were spent on identifying and preparing appropriate learning resources, and 10 hours were spent on technical issues.

At the time of writing this paper, the FIT Java Tutor is still in use. We therefore outline our experiences after 10 weeks of usage in the introductory programming class. Up to that point, more than 100 of the 400 students (regularly) used the tutor as an additional learning tool for watching video tutorials, writing programs for programming tasks, and solving quizzes/tests. Students have made use of artifacts for learning about 1,900 times: video tutorials have been used 355 times, programming tasks 1,317 times, quizzes 77 times, and tests 148 times. In terms of usage of the different views the system provides, the course-oriented view (see Sec. III-B1) has a usage percentage of 90%, followed by the problem-oriented view (see Sec. III-B2) with 8%. For solving programming tasks, students could write Java programs using the integrated

programming environment (see Sec. III-A). Students have used this environment to compile and execute more than 3,200 Java programs. The feedback option has been used 390 times.

Programming tasks were learners' preferred artifact type. Moreover, the data indicate that instructing a learner while solving a problem (as addressed in R1) is crucial for learning programming. While the course view (see III-B1) has been students' preferred view, the resource-oriented view has surprisingly been used only a few times. A possible explanation for students' preference could be that, since most of them were probably very beginners, students rather needed orientation in learning as offered by the course view (that addresses R2) than exploration support (as addressed in R3) or adaptation features (as addressed in R4). However, learners' behavior might change over time (e.g. when students prepare for exam or get more experienced). Assuming that learners' needs change over time, we derive the following three research questions: (1) How do learners' needs change over time while learning programming? (2) When do learners need additional support (as addressed in R3 to R5)? Manually designing an online course as it is necessary for the course view means a huge effort – whereas the resource-oriented view automatically structures learning resources based on semantic proximities. This leads to question (3) whether it is possible to derive learning objectives from learning data (e.g. learners' activities during learning) automatically in order to structure learning resources based on pedagogical relations, and thus to enhance the resource-oriented view so that it not only visualizes semantic proximities among a set of artifacts but also contains reasonable learning objectives via clusters of artifacts.

V. Conclusion & Outlook

In this paper, we introduced a learning environment for Java programming called the FIT Java Tutor. The environment integrates several pedagogical principles in order to provide instructional support for problem-solving, to structure learning resources taking into account learning objectives, to support exploratory and collaborative learning, and to enable a learner to adapt the environment to her needs. The system has been deployed in an introductory programming class at Humboldt-Universität zu Berlin. The students of this class extensively used the learning environment overall, however some of the views/features provided by the system were paid little attention to by them. From these experiences, we derived three research questions. In future research, we plan to investigate these questions and to evaluate the FIT Java Tutor in terms of usability, scalability and its effect on learning outcomes.

Acknowledgment

This work was supported by the German Research Foundation (DFG) under the grant "FIT – Learning Feedback in Intelligent Tutoring Systems." (PI 767/6).

References

[1] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '10. New York, NY, USA: ACM, 2010, pp. 86–93. [Online]. Available: http://doi.acm.org/10.1145/1930464.1930480

[2] T. Jenkins, "A participative approach to teaching programming," in *Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education: Changing the delivery of computer science education*, ser. ITiCSE '98. New York, NY, USA: ACM, 1998, pp. 125–129. [Online]. Available: http://doi.acm.org/10.1145/282991.283090

[3] N. Cavus and D. Ibrahim, "Assessing the success rate of students using a learning management system together with a collaborative tool in web-based teaching of programming languages," *Journal of educational computing research*, vol. 36, no. 3, pp. 301–321, 2007.

[4] I.-H. Hsiao, S. Sosnovsky, and P. Brusilovsky, "Guiding students to the right questions: adaptive navigation support in an e-learning system for java programming," *Journal of Computer Assisted Learning*, vol. 26, no. 4, pp. 270–283, 2010. [Online]. Available: http://dx.doi.org/10.1111/j.1365-2729.2010.00365.x

[5] B. Vesin, M. Ivanovi, A. Klanja-Milievi, and Z. Budimac, "Protus 2.0: Ontology-based semantic recommendation in programming tutoring system," *Expert Systems with Applications*, vol. 39, no. 15, pp. 12 229 – 12 246, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417412006495

[6] H. Hohl, H.-D. Bcker, and R. Gunzenhuser, "Hypadapter: An adaptive hypertext system for exploratory learning and programming," in *Adaptive Hypertext and Hypermedia*, P. Brusilovsky, A. Kobsa, and J. Vassileva, Eds. Springer Netherlands, 1998, pp. 117–142. [Online]. Available: http://dx.doi.org/10.1007/978-94-017-0617-9_5

[7] S. Karkalas and S. Gutierrez-Santos, "Enhanced javascript learning using code quality tools and a rule-based system in the flip exploratory learning environment," in *Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on*, July 2014, pp. 84–88.

[8] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark, "Intelligent tutoring goes to school in the big city," *International Journal of AI in Education*, vol. 8, pp. 30–43, 1997.

[9] A. Ogan, V. Aleven, and C. Jones, "Advancing development of intercultural competence through supporting predictions in narrative video," *International Journal of AI in Education*, vol. 19, no. 3, pp. 267–288, Aug. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1891970.1891972

[10] N. T. Le, S. Strickroth, S. Gross, and N. Pinkwart, "A review of AI-supported tutoring approaches for learning programming," in *Accepted for the International Conference on Computer Science, Applied Mathematics and Applications 2013, Warsaw, Poland*. Springer Verlag, 2013.

[11] P. Brusilovsky and M. Yudelson, "From webex to navex: Interactive access to annotated program examples," *Proceedings of the IEEE*, vol. 96, no. 6, pp. 990 –999, june 2008.

[12] S. Gross, B. Mokbel, B. Hammer, and N. Pinkwart, "Towards a domain-independent its middleware architecture," in *Proc. of the 13th IEEE International Conference on Advanced Learning Technologies (ICALT)*, N.-S. Chen, R. Huang, Kinshuk, Y. Li, and D. G. Sampson, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2013, pp. 408–409.

[13] B. Mokbel, S. Gross, B. Paassen, N. Pinkwart, and B. Hammer, "Domain-independent proximity measures in intelligent tutoring systems," in *Proceedings of the 6th International Conference on Educational Data Mining (EDM)*, S. K. D'Mello, R. A. Calvo, and A. Olney, Eds., Memphis, TN, 2013, pp. 334–335.

[14] S. Gross, B. Mokbel, B. Paassen, B. Hammer, and N. Pinkwart, "Example-based feedback provision using structured solution spaces," *Int. J. of Learning Technology*, vol. 9, no. 3, pp. 248–280, 2014.

[15] M. Bostock, V. Ogievetsky, and J. Heer, "D3: Data-driven documents," *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011. [Online]. Available: http://vis.stanford.edu/papers/d3

[16] W. Hung, D. H. Jonassen, R. Liu *et al.*, "Problem-based learning," *Handbook of research on educational communications and technology*, vol. 3, pp. 485–506, 2008.

[17] A. N. Kumar, "Learning programming by solving problems," in *Informatics Curricula and Teaching Methods*. Springer, 2003, pp. 29–39.

[18] C. Qu and W. Nejdl, "Towards interoperability and reusability of learning resources: A scorm-conformant courseware for computer science education," in *Proc. of the 2nd IEEE International Conf. on Advanced Learning Technologies (ICALT), Kazan, Tatarstan, Russia*, 2002.

[19] J. Zhou, X. Nie, L. Qin, and J. Zhu, "Web clustering based on tag set similarity," *Journal of Computers*, vol. 6, no. 1, 2011. [Online]. Available: http://ojs.academypublisher.com/index.php/jcp/article/view/06015966