Intelligent and Adaptive Student Support in FLIP
Early Computer Programming

Sokratis Karkalas and Sergio Gutierrez-Santos
Department of Computer Science and Information Systems,
Birkbeck, University of London, Malet Street, London, WC1E 7HX, U.K.

1 STAGE OF THE RESEARCH

The general purpose of this work is to develop techniques by which a significant proportion of teaching of elementary computer programming can be shifted from human tutors to intelligent agents. An extensive literature review has been made and a concrete research objective has been formulated. The techniques that have and will be developed as part of this work are being delivered and tested through a computerised tutoring system. A framework has been designed for this purpose and a first prototype of an Exploratory Learning Environment (ELE) has been deployed. This ELE is called FLIP (FLexible, Intelligent, Personalised) and it has already been used and tested in the classroom. Currently FLIP integrates a combination of off-the-shelf and own components to provide intelligent support to students of Javascript programming. This is task-independent support provided in the context of open-ended exploratory programming sessions. FLIP's natively supported AI component is a rule-based reasoner (Karkalas and Gutierrez-Santos, 2014a) that is based on a concept inventory (CI) of student initial misconceptions (Karkalas and Gutierrez-Santos, 2014b). The development of this CI was based on previous research projects (Goldman et al., 2008; Kaczmarczyk et al., 2010) and original exploratory field reseach. The data elicitation process took place in University computer laboratories and involved collection of primary data through direct observations and face-to-face interviews. The system is also adaptable and can adjust the level of support depending on previous student activity. This is based on a learner model that takes into account the amount of support that has already been provided by the system.

An issue that is inherently problematic in all rule-based systems is the method of conflict resolution between competing rules that get activated concurrently. This is an important aspect of the system's automated support as it is directly related to the human-computer interaction component and may significantly influence the applicability and effectiveness of the technique. This is therefore an emerging problem that needs to be addressed and resolved in the remaining part of this research.

In parallel the techniques employed in the system should be assessed both with formative and summative evaluations. If these techniques prove to be effective and time permits, then the same system can be tested for task-dependent support.

2 OUTLINE OF OBJECTIVES

The general aim of this work is to optimise the learning process during practical sessions in computer laboratories. Students should be able to receive timely support about issues that hinder their learning cycle. The learning process should be continuous and uninterrupted so that the maximum possible outcome can be achieved. The level of support must be adaptive to the individual needs and circumstances of the students.

The two basic constraints that have to be taken under consideration are time and human resources. The objectives given above have to be met under the following condition: The available time for laboratory work and the human resources available for help during that time are fixed. Provision of extra support whenever needed should be a fully automated process. Adaptability, which is highly related to learner models and processing of students' logs must be fully supported by technology.

Another consideration is the ability of the intelligent support component to be reflexive (Maes, 1988) and evolve using its own learning analytics as feedback. This can be particularly useful for automatic conflict resolution between competing rules in the reasoner.

3 RESEARCH PROBLEM

Learning computer programming is particularly hard especially during the early stages (Soloway, 1986; Jenkins, 2002; Robins et al., 2003). Programming is a craft and requires the development of practical skills that can only be learnt through practical training (Vihavainen et al., 2011). Typically, learning takes place either in a workplace through apprenticeships or in University computer laboratories through training courses. In the latter case students are given problem- and/or inquiry-based learning scenarios (Savery, 2006) and work individually or in pairs under the supervision of tutors. Students are not expected to follow instructions and repeat actions. They are encouraged to explore their own strategies, designs, patterns and techniques through experimentation. They are expected to discover knowledge in an exploratory manner (Huitt, 2003; Vygotskiĭ et al., 1978). Learning this way is painful. It involves investigation, planning, tactics and action. Tutors play a crucial role in this process. They are not just people that merely give instructions and expect answers. They actively engage in the process as facilitators and they contribute by identifying problems, giving directions and confirming acceptable solutions. It has been established that considerable effort is required by tutors to ensure effective learning in such open-ended contexts (Kirschner et al., 2006; Kynigos, 1992; Mayer, 2004).

Time and human resources in computer laboratories are limited. The effectiveness of this process highly depends on whether utilisation of these resources is optimal or not. Students and tutors have their own individual characteristics, problems and idiosyncrasies. Students expect individualised support that reflects their particular misconceptions and practices. Tutors are expected to make bias-free and informed decisions about the type and level of support needed in every case and respond accordingly. The latter presuposses that tutors have a deep knowledge of students' profiles and the ability to analyse previous activity on the spot in order to provide suitable and adequate support in every case. Support in this context is a multi-faceted and complex task that requires a lot of preparation, expertise, time and resources. Decisions must be based on a multitude of criteria and a considerable amount of data about students. It is evident that human tutors cannot respond effectively to these challenges without help.

Another important aspect of learning in a computer laboratory is the sequence of actions that take place during a learning cycle (Kolb et al., 1984; Konak et al., 2014). There is a pattern that students follow when they engage with a task. The sequence of actions they execute follows a cyclical process. In every round students attempt to code something that brings them closer to the completion of the task at hand. Sometimes this is interrupted by the inherent inability of the student to move forward. That can be lack of knowledge or a misconception. This is the point where the student hits the inner circle of their particular ZPD (Vygotskiĭ et al., 1978). The only way to overcome the problem in this case is to receive enough and relevant help in a timely fashion. Typically the tutor intervenes and provides the help needed so that the student can move on and complete the cycle. The student conceptualises the issue and then confirms the validity of new knowledge through active experimentation. Computer laboratories are especially busy during the early stages of learning. These interruptions are very frequent and support may not be enough. Tutors have to prioritise and provide help to many people in a very limited time and that apparently can have a negative effect on the quality and quantity of service provided. If these cycles get interrupted, then learning gets interrupted. If support is not adequate and cannot be provided timely, then inevitably the learning process becomes less effective.

Both of the above (limiting) factors influence to a great extend the effectiveness of the learning process. Inadequate support potentially means that students may not be able to engage with the subject in the most costructive way and as a concequence of that they may not be able to exploit their full potential and achieve the best possible learning outcome. If technology can be used to compensate for these limitations then it is expected that the learning process will be much more effective, fair and inclusive.

4 STATE OF THE ART

The first attempts to utilise technology in this context started in the late 70s. The resulting systems (Brown and Burton, 1978; Reiser et al., 1985; Johnson and Soloway, 1985) were quite impressive since they were intelligent, adaptive and able to provide personalised support. These Intelligent Tutoring Systems (ITS) were used to teach both computing and other subjects in a fairly controllable manner. They were used in problem-based scenarios to direct students' activities to the desired outcome. Support was provided in an intrusive way in order to align the students' behaviour with what was thought to be acceptable. These implementations, despite the sophistication of the techniques used, failed to promote discovery of knowledge through exploration and constructivism in general.

A system with a somewhat different orientation is ELM-ART (Brusilovsky et al., 1996). This is a courseware delivery system equipped with intelligence and adaptivity. Other more recent systems are (Mitrovic, 2003; Sykes and Franek, 2003; Holland et al., 2009; Peylo et al., 2000). The system in (Mitrovic, 2003) is intelligent but not adaptive. It is a constraint-based approach that does comparisons between student solutions and model solutions defined by tutors. Another system based on constraint-based modelling is J-LATTE (Holland et al., 2009). J-LATTE teaches Java and provides support for both design and implementation issues. Another system that teaches Java is (Sykes and Franek, 2003). This system is both intelligent and adaptive and the underlying technology used in an expert system supported by decision trees. The system in (Peylo et al., 2000) teaches Prolog. In this case domain knowledge is represented as an ontology. All these systems, like their predesessors, focus on task-dependent support for well-defined problem-based scenarios. They provide guidance in a controllable manner and they fail to support discovery of knowledge through exploration.

A lot of work has also been done in the field of teaching-oriented Integrated Development Environments like BlueJ (Kölling et al., 2003), Greenfoot (Kölling, 2010), Alice (Dann et al., 2000), Karel (Bergin et al., 1997; Bergin et al., 2005; Becker, 2001), ToonTalk (Morgado and Kahn, 2008), LOGO-based Microworld (Jenkins, 2012) and Scratch (Maloney et al., 2008). These systems have been used extensively in the classroom and with remarkable results but they are not intelligent and/or adaptive. They cannot be used to make intelligent decisions on how to teach the subject and adapt to students' particular circumstances. These deficiencies innevitably limit their applicability and thus their value.

Another category of systems is non-academic on-line platforms like Khan Academy (https://www.khanacademy.org) and Code School (https://www.codeschool.com) that offer these services in a distributed manner. These systems offer well-structured tutorials accompanied by limited but useful learning analytics. These tutorials are typically sequential or hierarchical and the course of actions during the interaction with the system is pretty much deterministic. There is no intelligence and adaptability and automated support is non-existent. Web-based IDEs like JSFIDDLE (http://jsfiddle.net) and CodeSkulptor (Ben-Ari, 2011) are not designed to provide automated assistance and individualised support either.

5 METHODOLOGY

The issues addressed in this project are multi-faceted problems and as such they require the application of different approaches in terms of research methodology. The requirements elicitation for the construction of the Concept Inventory that is used as part of the rule-based reasoner was a field research. Although there was material from previous research that could have been used directly for that component, we decided not to use it without further investigation. The first part of this research was exploratory. We used direct observation and interviews and we systematically recorded every issue that took place in laboratory sessions of three introductory programming courses. The process followed is similar to Grounded Theory (Strauss and Corbin, 1994). We intentionally did not use findings of previous research in this process because we did not want to constrain our perception and therefore influence the resulting domain of knowledge. Our results were then compared with the existing classifications of already recognised misconceptions in the literature. That was the confirmatory part of the research.

The ELE that has been developed can be evaluated both as a software engineering project and as educational software. Direct comparison with other systems is not possible since there is no other system that targets the same educational objectives. This system is intended to be used for teaching introductory JavaScript programming in an exploratory manner through inquiry-based scenarios and open-ended ill-defined problems (Savery, 2006). A possible evaluation is to measure the extent to which the system's functionality corresponds to the requirements and expectations of stakeholders. The system so far has been designed, developed and evolving using both top-down and bottom-up approaches depending on the case. In general there is a preference for Agile Model (AM) methods. As the system evolves and improves it undergoes continuous formative evaluations by students and active practitioners in the field. The project will complete with a full-scale summative evaluation that includes a pre and post-test in the classroom.

6 EXPECTED OUTCOME

The contribution of this work is two-fold. The first part is the design of a framework for an ELE and the development of a prototype for it. The second part is the design and implementation of AI techniques that provide automatic task-independent feedback in an adaptive manner. More specifically the expected outcome is:

1. A framework for the deployment of ELEs specialising in teaching computer programming languages.
2. An implementation prototype that corresponds to the above framework. This is expected to be a web-based platform for easy distribution of the services and dissemination of the results.
3. An intelligent support component able to provide fully automated task-independent assistance on coding tasks.
4. A rule editor that can be used by experts to insert knowledge into the knowledge base component of the reasoner.
5. An intelligent component able to provide fully automated adaptive rule prioritisation in the reasoner (reflexivity).
6. A learner model that can be used to provide adaptability to students' individual circumstances.

REFERENCES

Becker, B. W. (2001). Teaching cs1 with karel the robot in java. In *ACM SIGCSE Bulletin*, volume 33, pages 50–54. ACM.

Ben-Ari, M. M. (2011). Moocs on introductory programming: a travelogue. *ACM Inroads*, 4(2):58–61.

Bergin, J., Stehlik, M., Roberts, J., and Pattis, R. (1997). *Karel+: A Gentle Introduction to the Art of Object-oriented Programming*. Wiley.

Bergin, J., Stehlik, M., Roberts, J., and Pattis, R. (2005). *Karel J Robot: A gentle introduction to the art of object-oriented programming in Java*. Dream Songs Press.

Brown, J. S. and Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills*. *Cognitive science*, 2(2):155–192.

Brusilovsky, P., Schwarz, E., and Weber, G. (1996). Elm-art: An intelligent tutoring system on world wide web. In *Intelligent tutoring systems*, pages 261–269. Springer.

Dann, W., Cooper, S., and Pausch, R. (2000). Making the connection: programming with animated small world. In *ACM SIGCSE Bulletin*, volume 32, pages 41–44. ACM.

Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., and Zilles, C. (2008). Identifying important and difficult concepts in introductory computing courses using a delphi process. *ACM SIGCSE Bulletin*, 40(1):256–260.

Holland, J., Mitrovic, A., and Martin, B. (2009). J-latte: a constraint-based tutor for java.

Huitt, W. (2003). Constructivism. *Educational psychology interactive*.

Jenkins, C. W. (2012). Microworlds: Building powerful ideas in the secondary school. *Online Submission*.

Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, volume 4, pages 53–58.

Johnson, W. L. and Soloway, E. (1985). Proust: Knowledge-based program understanding. *Software Engineering, IEEE Transactions on*, (3):267–275.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., and Herman, G. L. (2010). Identifying student misconceptions of programming. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 107–111. ACM.

Karkalas, S. and Gutierrez-Santos, S. (2014a). 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*, pages 84–88. IEEE.

Karkalas, S. and Gutierrez-Santos, S. (2014b). Intelligent student support in the flip learning system based on student initial misconceptions and student modelling. In *Knowledge Engineering and Ontology Development (KEOD), 2014 6th International Conference on*, pages 353–360.

Kirschner, P. A., Sweller, J., and Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist*, 41(2):75–86.

Kolb, D. A. et al. (1984). *Experiential learning: Experience as the source of learning and development*, volume 1. Prentice-Hall Englewood Cliffs, NJ.

Kölling, M. (2010). The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):14.

Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. (2003). The bluej system and its pedagogy. *Computer Science Education*, 13(4):249–268.

Konak, A., Clark, T. K., and Nasereddin, M. (2014). Using kolb's experiential learning cycle to improve student learning in virtual computer laboratories. *Computers & Education*, 72:11–22.

Kynigos, C. (1992). Insights into pupils and teachers activities in pupil-controlled problem-solving situations: A longitudinally developing use for programming by all in a primary school. In *Mathematical Problem Solving and New Information Technologies*, pages 219–238. Springer.

Maes, P. (1988). Computational reflection. *The Knowledge Engineering Review*, 3(01):1–19.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., and Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin*, 40(1):367–371.

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? *American Psychologist*, 59(1):14.

Mitrovic, A. (2003). An intelligent sql tutor on the web. *International Journal of Artificial Intelligence in Education*, 13(2):173–197.

Morgado, L. and Kahn, K. (2008). Towards a specification of the toontalk language. *Journal of Visual Languages & Computing*, 19(5):574–597.

Peylo, C., Teiken, W., Rollinger, C.-R., and Gust, H. (2000). An ontology as domain model in a web-based educational system for prolog. In *FLAIRS Conference*, pages 55–59.

Reiser, B. J., Anderson, J. R., and Farrell, R. G. (1985). Dynamic student modelling in an intelligent tutor for lisp programming. In *IJCAI*, pages 8–14.

Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172.

Savery, J. R. (2006). Overview of problem-based learning: Definitions and distinctions. *Interdisciplinary Journal of Problem-based Learning*, 1(1):3.

Soloway, E. (1986). Learning to program= learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9):850–858.

Strauss, A. and Corbin, J. (1994). Grounded theory methodology. *Handbook of qualitative research*, pages 273–285.

Sykes, E. R. and Franek, F. (2003). A prototype for an intelligent tutoring system for students learning to program in java (tm). In *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education, June 30-July 2, 2003, Rhodes, Greece*, pages 78–83.

Vihavainen, A., Paksula, M., and Luukkainen, M. (2011). Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 93–98. ACM.

Vygotskiĭ, L. S., Cole, M., and John-Steiner, V. (1978). Mind in society.