A Problem Based Approach to Teaching Programming

W. Pullan[1], **S. Drew**[2], **and S. Tucker**[1]
[1]School of ICT, Griffith University, Gold Coast, Queensland, Australia
[2]Griffith Institute of Higher Education, Griffith University, Gold Coast, Queensland, Australia

Abstract - *Java Programming Laboratory (JPL) is a cloud based learning environment used for teaching object-oriented programming at Griffith University, Australia. JPL incorporates a number of features found in other successful programming learning environments and builds upon them with a range of innovative features. JPL provides a database that tracks individual students' successes and progression through scaffolded programming exercises and assessment items and gives students immediate feedback on their use of programming language syntax and correctness of problem solutions. A data querying and visualisation facility allows analysis of the database to provide real-time performance indicators from the overall course / problem level down to the individual student / specific problem level. Programming instructors and curriculum designers will find that this facility allows a responsive approach to student engagement, assistance and progression; as well as course problem tuning in a just-in-time manner.*

Keywords: *Object-oriented Programming; Problem-Based Learning; Real-Time Progress Tracking; Scaffolded Development;*

1 Introduction[1]

Learning object-oriented programming is a difficult process for most first-year students but, once mastered, it is transformative in that it provides a means to represent and solve a range of computing related problems in many application areas. Compounding the difficulty in learning programming is the fact that it is normally taught, in first year, to a cohort of students with a considerable range of academic abilities and experiences. During the pre-2000 dot.com boom, by far the majority of ICT students at the authors' university were high achievers who had studied relevant preparatory courses at high school and were motivated by the prospect of a relatively highly paid career in an industry where they were in high demand. More recent moves towards massification of higher education (Altbach, Reisberg and Rumbley, 2009) and a local reduction in popularity of many ICT related professions has resulted in the lowering of entry requirements for ICT programs to maintain budget quotas. While these moves have increased the difficulty of teaching programming, it should be noted that, even in the past, teaching introductory programming was challenging and often resulted in student avoidance of further programming based courses. What is reported in this paper are the results of a project that addresses these challenges by implementing a successful learning and teaching system based on engagement theory (Kearsley and Schneiderman, 1998) and constructivist design philosophy (Jonassen, 2003; Karagiorgi and Symeou, 2005).

In the professional setting, where most academics are untrained in educational theory, a simple and effective approach is provided by having a focus on "active learning" strategies. Biggs (1999) strata of conceptions of teaching progress from concentration on student as recipient of the teacher's wisdom, or *what the student is*, to a concentration on executing the act of teaching, or *what the teacher does*, and ultimately a focus on *what the student does* in the quest for knowledge and eventual understanding. It is this final conceptual level of teaching that leads to a focus on active learning (Jonassen and Rohrer-Murphy, 1999; Scanlon et al., 2002; Tetard and Patokorpi, 2005; Baugh, 2009) and student learning through experience (Kolb, 1984; Boud, Cohen and Walker, 1993). Active learning, in the context of teaching introductory programming, where a student is to learn a programming language and the tools of programming they must write programs so task setting is not difficult. However, the best effect will be obtained in an environment that facilitates and motivates them to write programs and, with a wide range in student ability, this is where the challenge arises - how to keep the poorer students progressing while not causing the more able students to lose interest.

A programming learning environment, Java Programming Laboratory (JPL) was developed to address the design criterion of providing scaled levels of challenge to maintain student engagement with learning for a cohort of varied ability. An active learning approach allows teaching staff to concentrate on *what the student does* on their journey to learning programming in an immersive software development environment. There are a number of key design features of the JPL learning environment to address relevant instructional design theory. *Firstly*, it presents a component approach (Pintrich and de Groot, 1990; Biggs, 1999b) that is as 'concrete' conceptually as possible with students having to focus only on the topic currently being taught. *Secondly*, it presents a problem-based learning approach (Savery and Duffy, 1995) and provides a large range and number of problems, selected from an even larger range of problems available for each topic. Problems are designed to scaffold development (Simons and Klein, 2007) to cater for the wide range of student abilities, providing multiple entry points into

[1] This paper uses the naming convention program and courses to refer to a degree structure where a program is a set of courses. This is in contrast to the course / (unit/subject) naming convention.

each topic, and additional problems for better students. *Thirdly*, it promotes responsive, formative feedback (Nicol and Macfarlane-Dick, 2006), with continuous access to personal and class (group) performance measures, and access to just-in-time assistance. *Fourthly*, it provides a consistent, simple environment available for any computing platform; and flexible access to the learning environment that is available in university computer laboratories or off campus via the Internet. *Fifthly*, as a client-based application it ensures students can maintain learning momentum and time-on-task even while the Internet may not be accessible. *Finally*, to maintain originality of student learning it does not allow solutions to problems to be passed from year to year which mitigates the learning process.

JPL incorporates a large number of features, some of which are based on existing open-source programming teaching systems. These include the Stanford University (Stanford, 2011) teaching initiative titled *CodingBat* that has been used to teach Java programming at that institution, *VideoNotes* (Cornell, 2011) and interactive web-pages (Kjell, 2006). Using the lessons learned from these teaching systems, this paper now describes the JPL approach that has been taken to dealing with the issues highlighted above in teaching introductory programming.

2 Java Programming Laboratory

Java Programming Laboratory is an educational system designed to assist students learning Java as their first programming language. Basically, JPL provides a software-based environment, available on both the universities and students computers, that allows students to develop their programming skills by starting with simple, targeted, Java code fragments and slowly transitioning to complete Java programs. The fundamental concept underlying JPL is – "*you can learn programming skills by writing many, small, targeted code fragments*". Within JPL, learning a programming language is devolved into smaller steps of learning and practicing computer-based problem solving techniques. This simultaneously aids learning of programming language constructs, their syntax and semantics.

The overall system structure of JPL is shown in Figure 1 and the components of this system are described in the following bullet points / sub-sections.

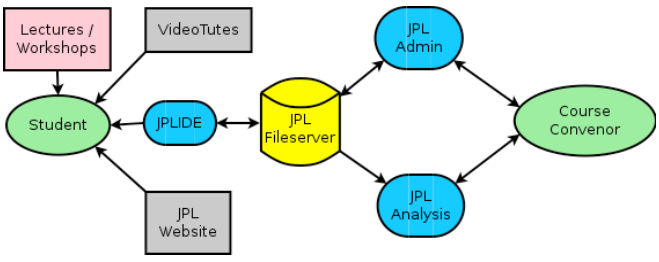


Figure 1 : JPL System Overview

- Lecture / Workshops : The lectures / workshops are very much a "*teach by example, learn by doing*" exercise in that the lectures are a 50/50 combination of topic discussions and joint (lecturer / students) JPL problem solving and implementation. The on-campus workshops are solely based on the student working in JPL with tutorial assistance to solve a specific issue directly available if required.
- JPL VideoTutes : An integral part of JPL is the use of short (~10 minutes) video tutorials explaining key programming concepts and problem solving techniques. In effect the student is able to 'look over the shoulder' at the computer screen as an experienced programmer demonstrates both programming language features and computer based problem solving from problem analysis to program implementation. Shorter versions of video-tutes are also used to provide hints to solve problems. These video micro-tutorials that support student learning of individual constructs provide an indexed system of instruction and reviewable examples to support incremental development of problem solving and programming knowledge through guided practice.
- JPL Website : The JPL website contains a number of interactive learning tools, some of which were originally created by Bradley Kjell, Central Connecticut State University (Kjell, 2006). For the object-oriented programming course in Java, these include multiple choice quizzes, each with correct answers supplied on completion, fill-in-the-blank review sessions and multi-page interactive topic discussions. The JPL Website is available at www.ict.griffith.edu.au/JPL.
- JPL Fileserver : The JPL Fileserver acts as the central repository for all JPL related files. When JPLIDE is initiated, if needed and the Internet is available, a merge of the local copy of the student's log and the student's log on the JPL Fileserver is performed and a new copy of the JPL Course Control File downloaded from the JPL Fileserver. As the student performs JPL activities, both copies of the student's log are kept up to date. If the Internet is not available, the student is still able to use JPLIDE and the logs will be merged when the Internet does become available.

2.1 JPL Integrated Development Environment (JPLIDE)

Students perform all programming activities completely within the JPLIDE, which is based on the DrJava open source IDE and has the user interface shown in Figure 2. To choose a problem, students access the Problems item on the Toolbar and this produces the JPL Problem Selector display shown in Figure 3. The problem selector contains a table of problem identifiers for the course, where the current status of the problem (template obtained, compiled, failed test, successfully completed (Green)) for the student is colour coded as background to the problem identifier. The leading character of the problem identifier gives the problem type (W -> Workshop problem, H -> Homework problem). Students start

at the bottom row of the problem selector table and work their way up the table (which allows for the situation where problem rows are released as the semester progresses) with problem difficulty increasing from left to right.

For straight-forward problems, such as that shown in Figure 2, the complete problem specification is stored in the problem template and the Java code after the Problem Statement is not present in the initial template. For more difficult object oriented problems, the problem specification, including items such as UML diagrams, are stored in a PDF file that is automatically downloaded when the associated problem is selected. In addition, all source files for supplied classes for a more advanced problem will be automatically downloaded directly into JPLIDE and any input disk files required will also automatically downloaded on the first execution of the student's Java program. This automatic downloading of everything that is required for a problem, allows the student to focus solely on developing a class which demonstrates a particular object oriented concept (e.g. inheritance, polymorphism).

Once a problem has been selected, the student will start creating code and use JPL automated testing to check for correct logic. Figure 2 shows the completed program and the response to a successful execution for a simple, non-object oriented problem.

A brief summary of the JPL related commands available on the JPLIDE ToolBar is:

- News/Workshops : Display current course news and expected workshop activity for each week of the course.
- Statistics : Provides information on how well a student is performing with regard to the rest of the class and a weekly / overall breakdown of the JPL tasks they have performed. In particular, this shows the student their JPL Performance Indicator and how it compares to the performance indicators for the complete class (Figure 4). This data is available immediately after the first workshop in Week 1 of semester.
- Problems : Presents the JPL Problem Selector (Figure 3) that displays all the problem identifiers, the current status of the problems for this student and allows a problem to be selected.
- Compile : Compiles the current Java program and saves a copy of the source file to the JPL Fileserver.
- Test : Using the current Java program, performs the JPL automated testing which uses test data and expected outputs stored in the JPL Course Control File.
- Run : Executes the current Java program outside the JPL testing environment (i.e. with manual input and normal output).
- Stop : DrJava command which terminates the currently executing Java program. Primarily used to terminate an infinite program loop.
- Examples : This allows students access to many Java code fragments to perform specific tasks and also to complete Java programs that they can compile and execute.
- Hints : These include a solution flowchart available as a hint; solution pseudo-code available as a hint; and hint suggesting Java methods to use – e.g. for a String problem, which String methods (of the 43 available) is best to use in the solution.

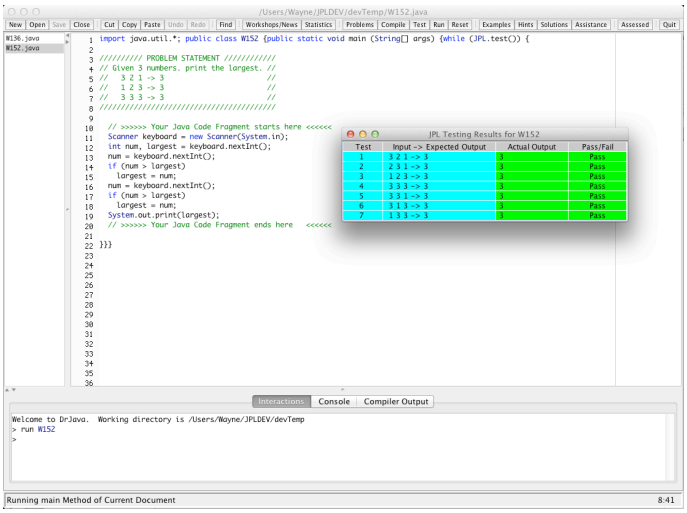
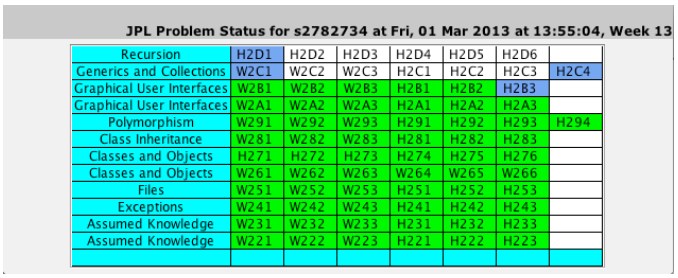


Figure 2 : JPLIDE User Interface and Automated Testing Result



JPL Problem Status for s2782734 at Fri, 01 Mar 2013 at 13:55:04, Week 13

Recursion	H2D1	H2D2	H2D3	H2D4	H2D5	H2D6	
Generics and Collections	W2C1	W2C2	W2C3	H2C1	H2C2	H2C3	H2C4
Graphical User Interfaces	W2B1	W2B2	W2B3	H2B1	H2B2	H2B3	
Graphical User Interfaces	W2A1	W2A2	W2A3	H2A1	H2A2	H2A3	
Polymorphism	W291	W292	W293	H291	H292	H293	H294
Class Inheritance	W281	W282	W283	H281	H282	H283	
Classes and Objects	H271	H272	H273	H274	H275	H276	
Classes and Objects	W261	W262	W263	W264	W265	W266	
Files	W251	W252	W253	H251	H252	H253	
Exceptions	W241	W242	W243	H241	H242	H243	
Assumed Knowledge	W231	W232	W233	H231	H232	H233	
Assumed Knowledge	W221	W222	W223	H221	H222	H223	

Figure 3 : JPLIDE Problem Selector

- Solutions : Provides, via a problem selector, solutions to selected problems. These are released after the relevant topic has been covered and students are unable to access a solution until they have made a reasonable effort at solving the problem.
- Assistance : Allows a student to request assistance with a problem. This results in an automatic email being sent to the course convenor who, working in Admin mode, is able to take over the student's JPL environment, make code corrections or just insert suggestions into the java source files. Once help has been given, the course convenor returns to their own JPL environment and an email is automatically generated and sent to the student.
- Assessed : Causes JPLIDE to enter Assessed Mode so that an Assessed Workshop can be performed. In Assessed Mode, JPLIDE is 'locked down' and the only file that can be opened is the Assessed Problem that is not normally assessable. In addition, a number of other

JPLIDE commands (e.g. Paste) are disabled. Assessed mode can only be entered as the first command after JPLIDE is started and, while JPLIDE is in Assessed Mode, all other executing programs on the computer are monitored to detect disallowed programs.

- Quit : Terminates JPLIDE after updating the JPL Fileserver (if the Internet is available).

Less commonly used JPLIDE commands that are available via a menu items include: Last Test Result, Student History, Restart Problem, Change Course, Java Tutorials, JPL Videos, JPL Website, and Administration items such as those described in the next sub-section.

2.2 JPL Course Administrator (JPLAdmin)

The JPLAdmin interface allows the course convener to create the workshop contents and also update the News/Workshop display. For the course convener only, the menu items available in JPLIDE under the Admin menu item include:

- Assist Student : In response to a student request for assistance, the course convenor is able to directly assist the student using the mechanism described for menu item Request Assistance above.
- Workshop Problems : This allows the course convenor to scan a library of problem templates (over 500 are available) for possible inclusion in a workshop. Problems are grouped by topic and difficulty ranking. There is a simple Course Definition text file that is updated to include a problem in a workshop.
- Build Workshop : This command takes the Course Definition text file edited above and creates the JPL Course Control File.
- Update Server : Loads the current JPL Course Control File up to the JPL Fileserver ready for automatic download by JPLIDE.

2.3 JPL Course and Student Analysis (JPLAnalysis)

JPLAnalysis is a stand-alone program that performs analysis of all student logs held on the JPL Fileserver. Statistics can be obtained at the problem, student, course level and any combination of these through a command line interface.

2.3.1 Student Level Reports

As an example of tracking an individual student, Figure 3 and Figure 4 show some of the information available at the student level. From Figure 3 it can be seen that this student, from the 74 JPL problems available in this course, successfully completed 60 problems, attempted 4 more and did not attempt 10 problems. The first graph in Figure 4 shows how a students JPL Performance Indicator (horizontal blue line) compares with the performance indicators for all the class (red columns). This graph is constantly available to students from Week 1 of semester and gives them constant feedback on their performance in relation to other students in the course. Obviously this student is performing very well however the second graph shows the students work pattern and clearly they were able to work ahead during the first portion of the semester and then able to make less effort during the latter part of semester.

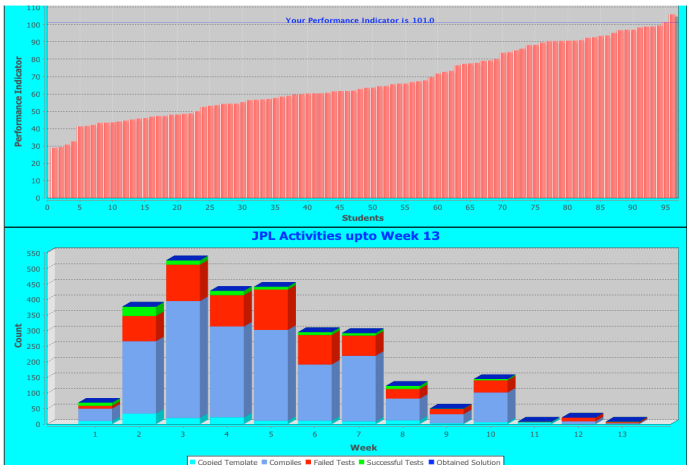


Figure 4 : JPL Performance Indicator and activity for a particular student

Tracking individual student activity within JPL has a number of uses in the university context. These range from identifying points in the course where students 'switch-off' to evaluating the amount of effort a student has put into producing an assignment so as to flag possible plagiarism cases.

2.3.2 Course Level Reports

An example of a course level report is shown in Figure 5 which shows the total student JPL activity on a week by week basis for the course.

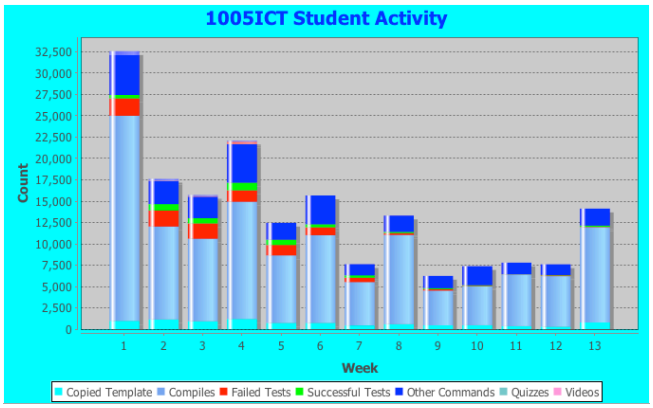


Figure 5 : Total student activity for each week of the semester

The slow decline in JPLIDE commands during semester is mostly due to the fact that the JPL problems became more difficult as the semester progressed so the rate at which students were inputting commands decreased.

2.4 JPL Course Results

The topics covered in this course are shown on the left-hand side of the JPL Problem Selector in Figure 3 and a summary of student activity is given in Table 1.

JPL Activity	Semester Total
JPLIDE Starts	8,065
Total JPLIDE Commands	189,805
Open a new JPL Template	7,794
Compiles	123,734
Failed Tests	10,239
Passed Tests	4,495
Total JPLIDE Dev. Commands	157,105
Home activity as a % of total activity	62%

Table 1 : JPLIDE Usage Counts for the Object Oriented Programming course with around 100 Students

A frequency analysis of constructs derived from student survey responses under the headings of "*what worked well*" and "*what needs improvement*" in the JPL based course revealed constructive information. Most popularly, JPL was perceived as helpful to students learning programming and that they appreciated the learning environment as being easy to use. Next most popular responses relate to the instructor facilitated laboratory learning experience based around practical exercises. Here students appreciated the quality of the tutorial assistance, the tutorial/laboratory learning experience, and the level and access to help that they found most effective. Frequent mentions were made of the flexible access to JPL from outside of laboratory classes and the appreciation of the lecturer and lecture/workshop classes. Interestingly, students found that JPL assisted them in understanding the curriculum and course structure and that the curriculum design assisted their learning. Several other lower frequency constructs were mentioned relating to various JPL and course design features.

In the area of what students felt needed most improvement there was a wide range of issues of low frequency suggesting a range of mainly individual learning needs that could be addressed. For most students a lack of understanding or experience of curriculum design for learning programming may limit the number of ideas they can offer for course improvement. Many, above, experienced an effective learning experience and would not seek to make changes. Indeed, the most frequent response in the survey was that they could think of no improvements. Next most frequent responses were from the percentage of the class having had no experience of learning to program. Some of these students suggested more help, and a softer introduction to the course, technical language, and requirements for undertaking programming. Relating to this there were requests for more tutorial assistance, more tutorials, and making the course less difficult. In direct contrast, and indicating the mixed levels of ability and experience amongst students, there were also requests for more problems with higher levels of difficulty and challenge. From the broad range of responses it appears that JPL and the course design managed to succeed in meeting the needs of such a diverse class.

3 Conclusions

JPL is based on the premise '*to learn programming you have to do programming*' and places students in an environment where they can totally focus on that. An observation from the current semester is that students are motivated by the JPL testing feedback they receive and are also motivated by the JPL Performance Indicator and the colour indicators on the JPL Problem Selector panel. These factors enhance student engagement with learning to program.

Within JPL, learning a programming language is devolved into smaller steps of learning and practicing computer based problem solving techniques. This simultaneously aids learning of programming language constructs, their syntax and semantics. Problems available in JPL are designed to scaffold student learning through a number of stages. These problems range from modifying existing programs, 'fill in the blanks' type exercises, developing Java code fragments and finally developing full Java programs. All student work in JPL is automatically tested and, as each student performs work on an exercise, the event is registered by the system. At a glance, at any point in the semester, the convenor or tutor can see how each student is performing and investigate any potential problems immediately.

Flexibility and continuity of access is important for our cohorts so the complete JPL system is available to students both in the ICT computer laboratories and also on their home computers. JPL has been used in a number of programming courses including high schools, an Australian university and a Chinese university. Within the university context, JPL seeks to address student retention and poor learning outcomes by improving scaffolding and learning support for key ICT programming courses. For the flagship ICT undergraduate programs, the courses that involve learning programming languages have been identified as containing key threshold concept areas. In all major strands of these degrees, learning to program is normally compulsory and starts from first semester, first year. Computer programming is considered a difficult learning area and can have a high student failure rate. Building upon a successful international blended-learning model, JPL provides an integrated program development environment which also includes automated testing and a comprehensive set of construct level, video-tute resources to aid computer program development and self-paced learning. This online program development and problem-based experiential learning environment enables academics and students to monitor progression through automatically evaluated learning objectives. JPL instructor access to students' achievements on tutorial and assessment tasks allows earliest possible identification of students who are at risk of failing in order to provide timely remedial assistance. JPL also provides feedback to the academic/teaching team

designing problem sets and curricula to identify where extra learning assistance or redesign is required.

Outcomes of the JPL approach to teaching introductory programming have thus far been very encouraging in terms of impacting positively on student learning experience and learning outcomes.

4 References

[1] Altbach, P. G., Reisberg, L., & Rumbley, L. E. (2009). Trends in global higher education: Tracking an academic revolution. Center for International Higher Education.

[2] Baugh, J. M. (2009). Let's Have Fun with That Required Computer Information Systems Introduction Course. Information Systems Education Journal, 7(73).

[3] Biggs, J. B., (1999a). Teaching for Quality Learning at University, Open University Press / Society for Research into Higher Education.

[4] Biggs, J. B. (1999b). What the Student Does: teaching for enhanced learning. Higher Education Research & Development, 18(1), 57 - 75.

[5] Boud, D., Cohen, R., & Walker, D. (1993). Using experience for learning. Buckingham [England]; Bristol, PA: Society for Research into Higher Education and Open University Press.

[6] Cornell University, (2011). VideoNote, from http://www.videonote.com/study.aspx

[7] Jonassen, D. H., & Rohrer-Murphy, L. (1999). Activity theory as a framework for designing constructivist learning environments. Educational Technology Research and Development, 47(1), 61-79.

[8] Jonassen, D. H. (2003). Learning to solve problems with technology: a constructivist perspective (2nd ed.). Upper Saddle River, N.J.: Merrill.

[9] Karagiorgi, Y., & Symeou, L. (2005). Translating Constructivism into Instructional Design: Potential and Limitations. Journal of Educational Technology & Society, 8(1), 17-27.

[10] Kearsley, G., & Schneiderman, B. (1998). Engagement Theory: A Framework for Technology-Based Teaching and Learning. Educational Technology, 38(5), 20-23.

[11] Kjell, Bradley , (2006), Introduction to Computer Science Using Java, Central Connecticut State University http://chortle.ccsu.edu/CS151/cs151java.html

[12] Kolb, D. A. (1984). Experiential learning : experience as the source of learning and development. Englewood Cliffs, N.J.: Prentice-Hall.

[13] Nicol, D., & Macfarlane-Dick, D. (2006). Rethinking Formative Assessment in HE: a theoretical model and seven principles of good feedback practice. Studies in Higher Education, 31(2), 199-218.

[14] Pintrich, P. R., & de Groot, E. V. (1990). Motivational and self-regulated learning components of classroom academic performance. Journal of Educational Psychology, 82(1), 33-40.

[15] Savery, J. R., & Duffy, T. M. (1995). Problem-based learning: An instructional model and its constructivist framework. In B. G. Wilson (Ed.), Constructivist learning environments: Case studies in instructional design (pp. 135-148). Englewood Cliffs, NJ: Educational Technology Publications.

[16] Scanlon, E., Morris, E., diPaolo, T., & Cooper, M. (2002). Contemporary approaches to learning science: technologically-mediated practical work. Studies in Science Education, 38(1), 73 - 114.

[17] Simons, K. D., & Klein, J. D. (2007). The Impact of Scaffolding and Student Achievement Levels in Problem-based Learning Environment. Instructional Science, 35(1), 41-72.

[18] Stanford University, (2011), CodingBat, from http://www.codingbat.com

[19] Tetard, F., & Patokorpi, E. (2005). A Constructivist Approach to Information Systems Teaching: A Case Study on a Design Course for Advanced-Level University Students. Journal of Information Systems Education, 16(2), 167-176.