Web Teaching and Learning Programming Environment Based on Plan Method and Constructs

Alireza Ebrahimi, Ph.D.
Management, Marketing, and Information Systems
School of Business – SUNY College at Old Westbury
Old Westbury, New York 11568 ebrahimia@oldwestbury.edu
WWW home page: http://www.drebrahimi.com

Abstract. Plan integrations and misconception of programming language constructs have been two major errors of novice programmers. A good design model can have a great impact on the effectiveness of these systems. A plan is an abstraction that visually provides a solution to a problem or to a sub-problem representing a problem from its macro level down to its micro level. Any important concept is a plan and it is up to the educator and level of learner to decide a particular plan. A plan can be visually represented by a dot, a geometric shape or an image. The Web Visual Learning System (WVLS) divides the process of learning and its enforcement into three selectable phases known as plan observation, plan integration, and plan creation. WVLS initially provides a library of sample problems (plans) working with all three phases. A learner can observe the process of solving a problem, become involved in a partial solution, or solve the problem entirely from beginning to end. A mixture of learning strategies and techniques is incorporated in WVLS to satisfy a wide range of learners. WVLS will identify and report the cause of problems to the learner. A systematic approach to analysing a solution based on plan relationships indicates whether a plan is missing, misplaced, malformed, or has a misconception.

1 Web tools for a learning design

In the design of a learning system on the web, like any other web system, there are a number of tools available. With the current technology, learning a web tool should be as easy as learning word processing. A common available web tool could be Microsoft Office FrontPage or even Microsoft Word when a document is saved as a web page. Alternatively, a sophisticated learning system can be created using Dreamweaver and Flash from Macromedia company. WebPages created with FrontPage, Word, or any other web tool automatically generate HTML (Hyper Text Markup Language) tags which are the backbone of every webpage [4]. To view a webpage's HTML, one can right-click on the mouse to view the source. By viewing a sample source, it is possible to figure out some of the styles and approaches in the design of a learning system. While it is not necessary to learn HTML tags to build a learning system, learning (memorizing) the 10 most important commands of HTML makes up about 90 percent of all web page design [1].

By adding a little program, which is called a script, to HTML, it is possible to make the learning system powerful and innovative. Two known script languages are JavaScript and VBScript; most designers choose JavaScript over VBScript. While it is not necessary to become a programmer, with a little knowledge of common built-in script functions, one can make a better learning system. Many novice designers become frustrated and overwhelmed with the current web tools and surrounding technology. The same group has also enjoyed the learning and experience as a result. While some stayed with their own design, others ask professional designers to design the new learning system, with one difference that they were able to delegate the instruction and build a customized web learning system. An important lesson learned from the design a web learning system is to separate what is important and what is less important. Designers should focus on the major issues rather than getting involved in the jargon of instructions and details.

2 Problem, plan, visualization, and web

Breaking down a problem into smaller units, makes the problem easier to understand and easier to solve. Therefore, a plan to solve a problem is set aside and sub-plans are made for sub-problems. Once the plan is broken down into sub-plans, then each sub-plan can be tackled separately. To build a learning system one needs to identify the plans, explain the plans using a language that is easy to understand, and propose a plan solution.

Visualization plays an important role in illustrating the plan and sub-plan and their relationships. A plan is an abstraction that represents the solution to a problem, such as the entire solution or the smallest component that is crucial to solving a sub-problem. The plan abstraction applies information hiding so that the learner is not overwhelmed with information that is not necessary at the moment. WVLS is an environment that provides an incremental way of learning and problem solving, rather than requiring the problem to be solved all at once. WVLS assures the learner's capability and assesses the learner’s problem at an early stage and grows as the learner grows.

Another example of an interactive web-based system is Environment for Learning to Program (ELP), which helps teaching programming to the novice by not having the need for a program development environment and installation of programming language. Using ELP, students use template exercises through the web to build their programming and problem solving skills [6].

2.1 Phase 1 of learning: Plan observation

By observing how a problem is solved step by step, the learner will be able to repeat the solution process over and over until it is understood. Plan observation enforces the retention of the plan and enables the learner to relate it to other similar situations. The WVLS plan observation phase is an automatic process that illustrates the steps involved in a task, starting with the initial specifications of the problem to its final solution. In the plan observation phase, the learner goes through the entire process of problem solving such as plan decomposition and plan integration and can repeat the process as many times as needed.

2.2 Phase 2 of learning: Plan integration

A problem's solution consists of several sub-plans that must be integrated correctly to form the final solution. After breaking down the plans, it is important for the learner to understand the relationship between the plans such as how these plans are put together and their spatial relationships. WVLS has four ways to integrate plans: an appended plan, an interleaved plan, a branched plan, or an embedded plan. Plan integration is a good test to ensure and examine the understanding of the solution of the problem at an abstract level, rather than its detailed steps.

2.3 Phase 3 of learning: Plan creation

In order to solve a problem entirely, a learner must start from problem specification and requirements to plan decomposition, plan integration, and finally testing the correctness of the plan and its explanation. The learner is responsible for creating a plan and demonstrating problem-solving skills. The problem could be one that already exists in the WVLS library, or it could be a new problem. If the problem exists in the system, there are three incremental learning scenarios: 1) full access 2) limited access 3) denied access.

When a learner requests to create a new plan that is not part of the system library, the entire WVLS will be at the service of the learner, including all phases. However, the new plan may consist of several sub-plans that already exist in the system and are shared by other plans. Shared sub-plans can be reused as they are or modified to suit the purpose. In order to assist the learner, WVLS will monitor and profile the learner as problem-solving steps are taking place. The system is intention based and will collect clues and make suggestions along the way.

3 VPCL: A WVLS for novice programmers

VPCL (Visual Plan Construct Language) is a tool for teaching beginner and novice programmers. Textual representation of programs and sub-programs (functions) contributes to misconceptions of how sub-programs interact with each other. Visual plan representation has resolved some of these problems. Plans are independent of programming functions. It is emphasized that a function is a plan, but a plan is not necessarily a function. Functions (sub-programs) have restraints on how they are used and how they interact with each other. A plan does not have these restrictions. Plans can either be dependent or independent of the programming language used. One task of VPCL is focused in terms of plans and sub-plans structurally, to build abstraction on each level and not overwhelm the learner. The idea of plan programming has been used in different ways as early as the mid eighties [5]. A novice programmer's support environment using plans can be created using hypertext to show the relationship between plans as well as breaking down the program to the smallest piece possible [3].

The plan creation phase of VPCL deals with writing actual programming code. In this phase, the user is given problem specifications and is required to develop all the steps to arrive at a solution. To translate a problem into a written program, several plans will be needed. These plans may exist and need to be modified or may be created from scratch. After all necessary plans are created, the plans must be integrated, and then the program is executed.

Several learning strategies and systematic ways of evaluating the learner's problem solving skills are incorporated in VPCL. The effectiveness of VPCL as a learning and instructional tool has been shown by the result of empirical studies on novice programmers [2].

4 Conclusion and future work

WVLS ensures the understanding of the subject matter by taking advantage of the web and visualization and using three incremental phases of learning. Different levels of abstraction are applied at each phase in order not to distract the learner from reaching the main goal. WVLS promotes a standard way to communicate by decomposing a plan, integrating plans, and investigating the cause of errors and building a new plan. The WVLS visualization and standardization bridges the gap of misconceptions. While WVLS has been tested on novice programmers, it is a generic environment that can be applied to other problem-solving disciplines. Future work on WVLS can be done to build an expert system in reporting errors, assisting the learner, and profiling the learner more accurately.

References

1. A. Ebrahimi (2002) C++ Programming Easy Ways, American Press, Boston.
2. A. Ebrahimi (1992) VPCL: A Visual Language for Teaching and Learning Programming (A Picture is Worth a Thousand Words). In Journal of Visual Languages and Computing 3, 299-317.

3. B. Liffick, R. Aiken (1996) A novice programmer's support environment. Proceedings of the 1st conference on Integrating technology into computer science education, Volume 28, 24 Issue SI , 1-3.

4. J. Niederst (2001). Web Design in a Nutshell (2/E). Sebastopol, CA: O'Reilly & Associates.

5. Soloway, E. (1984). A Cognitively-Based Methodology for Designing Languages / Environments / Methodologies. Proceedings of the ACM Symposium on Practical Software Development Environments.

6. N. Truong, P. Bancroft, P. Roe (2005). Learning to Program through the Web. Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE '05.