Journal of Information Technology Education **Volume 5, 2006**

On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process

Samer Al-Imamy
Higher Colleges of Technology, United Arab Emirates

samer.alimamy@hct.ac.ae
(or samer.alimamy@gmail.com)

Javanshir Alizadeh
Ajman University of Science & Technology Network, United Arab Emirates

alizadeh41@gmail.com

Mohamed A. Nour
University of Sharjah, United Arab Emirates

mnour@sharjah.ac.ae

Executive Summary

One of the major issues related to teaching an introductory programming course is the excessive amount of time spent on the language's syntax, which leaves little time for developing skills in program design and solution creativity. The wide variation in the students' backgrounds, coupled with the traditional classroom (one size-fits-all) teaching strategy, and bounded course duration, makes it extremely difficult for an instructor to go beyond adequate syntax coverage, to developing and enhancing the student's problem solving abilities. The solution to this dilemma is provided by a teaching environment that transforms and enhances traditional classroom teaching with a customized software teaching tool. We proposed, developed, and tested a software tool for teaching a first course in programming languages. The tool was used in teaching one of four sections of a programming course in our institution with favorable comparative results. This paper presents the comparative results of testing the teaching tool on an introductory computer programming course. Our results indicate that the tool was very effective in improving the students' performance in learning the programming concepts and reducing the time spent on syntax coverage. Additionally, the tool has proved to be useful in the following:

- creating templates that enforce the important concepts and constructs,
- providing the flexibility to switch from a language to another,
- helping to bridge the gap between students from various backgrounds,
- increasing the student's enthusiasm and confidence in writing correct programs.

Editor: Lynn Hunt

- providing a web-based self learning tool that will reduce the need for teaching and learning assistance.

Keywords: Programming Language, First Programming Course, Teaching Tool, Language Syntax, Template-based Teaching

Introduction

Information systems have become a critical component of the competitive strategy of many of today's business organizations. The demand for programmers and software developers continues to increase, as information technology is increasingly being integrated into the architecture and strategy of organizations. However, the supply of sufficiently competent and skilled programmers remains an illusive goal. In particular, given the diverse educational background of the students, the task of preparing them to the job market by providing them with appropriate programming skills is becoming more important.

Many colleges require students to take programming courses not as a major but as a college requirement to satisfy a set of broad skills and knowledge for a degree in another field, such as finance. Many such students show interest in computer programming (Gayo-Avello & Fernandez-Cuervo, 2003), but the majority of them find it a difficult and complex cognitive task (Guindon, 1990; Jeffries, Turner, Polson, & Atwood, 1981; Kim & Lerch, 1997; Letovsky, 1986; Mayer, 1989; Simon, 1973). In our institution, an introductory programming course is compulsory to all the students of the college of business. This course is intended to provide a broad set of problem solving skills as a foundation for business problem solving, as noted by Soloway (2003) and Moor & Deek (2006). Furthermore, in many degree programs, such as computer science and information systems, students take a related set of programming courses, beginning with an introductory one.

The two fundamental issues related to teaching programming are: the selection of a programming language for the introductory course and the most effective approach for teaching this introductory course. The choice of the first programming language, and its teaching approach, are critical to the students' understanding and acquisition of programming skills as well as their preparation for the next higher programming course (Parker, Chao, Ottaway, & Chang, 2006). The question of which programming language, such as C, Basic, Pascal, Java, or COBOL, is ideal as an introductory course has been controversial, influenced by the evolution of the programming languages and software engineering disciplines. The teaching approach is even more problematic and open to a great deal of experimentation. Most of the challenges inherent in teaching an introductory programming course, especially within the constraints imposed by a semester system (Churcher & Tempero, 1998), are related to the excessive amount of time spent on teaching the language syntax and the wide variation in the students' backgrounds. Covering the language's syntax is but one of the three primary pedagogical goals of teaching a programming language: program design skills and creative thinking are probably the ultimate skills sought after, which are intended to provide and reinforce generic problem-solving skills that are so vitally important in today's business world (Moor & Deek, 2006).

Many academics report their experience in using various teaching methods in this regard. These methods include changing the programming language, using different textbooks, slowing the course down, switching between bottom-up and top-down approaches, or even lowering the standards or reducing the course requirements, as some academics suggest. None of these methods has produced any significant results (Baldwin & Kuljis, 2000). With the increase in enrollments and the diversity in the students' backgrounds, abilities, learning speeds, and attitudes, an alternative approach is needed to overcome the difficulties inherent in teaching an introductory programming course (Gayo-Avello & Fernandez-Cuervo, 2003). Computer programming demands complex cognitive skills such as reasoning and planning (Kurland, Pea, Clement, & Mawby, 1986). There is a general acceptance in the literature that learners need to acquire syntactic, conceptual and strategic knowledge, as McGill and Volet (1997) suggest.

The goal of this research paper was to develop a computer programming teaching tool that addresses the issues and challenges we have faced in our school, and have been reported also by many others (McCracken et al., 2001; Sheard & Hagan, 1997). The objective of this research focuses on the following:

1. Course management time, with syntax coverage taking a much shorter time.
2. More emphasis on design and creative thinking/problem solving skills.
3. Guiding the students, through the use of templates, in their learning processes.
4. Independent web-based learning, with less need for supervision.
5. Language-independent learning, where students can learn multiple languages in parallel and easily switch from one language to another.

The rest of the paper is organized as follows. A background on the challenges of teaching computer programming is presented in the next. Our research methodology appears in the third section. The fourth section provides our experimental results. Discussion appears in the fifth section, and conclusions with suggestions for future work are given in the final section.

Background

There have been discussions in the literature concerning the selection of the first programming language (Allen, Grant, & Smith, 1996; Galy & Waldron, 2002; Guzdial, McCracken, & Elliott 1997; Reinfelds, 1998). The choice of the first programming language, while not yet settled, is arguably less important than the issues of what to teach, how to teach it, and achieving an appropriate balance between knowledge of the language syntax, design skills, and creativity. A first programming course needs to introduce the programming principles, which are usually conceptual in nature, using any suitable computer programming language. The language itself is not as important as the construction of a solution to the problem under study. Programming languages are used sometimes solely for the development of creative thinking and design skills.

However, the objective for students of computer science and information systems is the ability to design and construct programs as creative solutions to real world problems. One programming language is not sufficient for these fields, as programming is usually their ultimate career. In our institution, “Principles of Business Programming” is the first course in the MIS major. This is a required course for all the students of the four departments of the college (management, accounting, economics and MIS). It is a vital course for today’s business students, as it helps to build and enhance their problem-solving and creative thinking skills. This, however, introduces a significant complication in teaching such a course, as students have various backgrounds and interests.

Our students enter the college from the high school with art (literary) or science concentrations. Initially, we chose the C language for teaching the Principles of Business Programming course. Although the selected language is essential for the MIS students, it was found to be far from practical or useful to the students from the other departments. Visual Basic.NET was then selected for the last couple of semesters, but due to the differences in syntax our MIS students now face a new challenge when moving up to the Java course which is required as a second programming course for the MIS students. We are, therefore, back to the old ugly question of which language should be selected as a first programming language. Thus, besides the problems related to teaching the three sets of skills (syntax, design, creativity) in the first course, MIS students face the additional challenge of switching to a completely different language once they finish the first. This problem is one of the reasons why some institutions are thinking to migrate or have already migrated to using Java as the introductory (first) programming course (Campbell & Bolker, 2002). The development of creative thinking and design skills is one of the main objectives for the first course regardless of the language used. This must occur in the first course or students will suffer in all the following programming courses.

Teaching a language's syntax often takes the greater part of a semester's time. However, with the syntactic knowledge, students are usually able to write programs that can be compiled, but the students would not necessarily possess the design and development knowledge required for difficult and realistic programming problems. A programming language's syntax deals with the facts and rules governing the language (Bayman & Meyer, 1998). Syntactical knowledge is an important step in learning a programming language by novice programmers and could, according to our experience, consume up to two thirds of the course period; nonetheless, some students still face serious problems understanding the language syntax (Sherad & Hagan, 1997). Our primary goal is to reduce the time required to cover the syntax and to alleviate the syntax problems facing students. Having a system that guides the students in writing the program's constructs correctly, avoiding thereby the most common and silly mistakes, will speed the learning process and help the students to become more confident (Ala-Mutka, Uimonen, & Jarvinen, 2004).

Research Methodology

This paper describes an experimental tool that was developed to achieve the above-mentioned goals. This web-based tool helps in mastering the various programming constructs and forces the students to follow the instructor's templates, which emphasize the conceptual knowledge as stated by Linn and Dalbey (1989), to achieve a well planned learning path. It reduces the need for a commercial development environment and at the same time provides valuable information back to the instructors. We discuss below the main architectural features of the tool.

System Architecture

The proposed system consists of the components shown on Figure 1. These components are divided into a client side (shown as shaded) that is a combination of three frames within the development page that is displayed to the user directly after logging-in.

A read only instructor's template is displayed within *dev.xml* frame using the *format.xsl* file. A template, simply, is a skeleton program structure that the students can easily com-

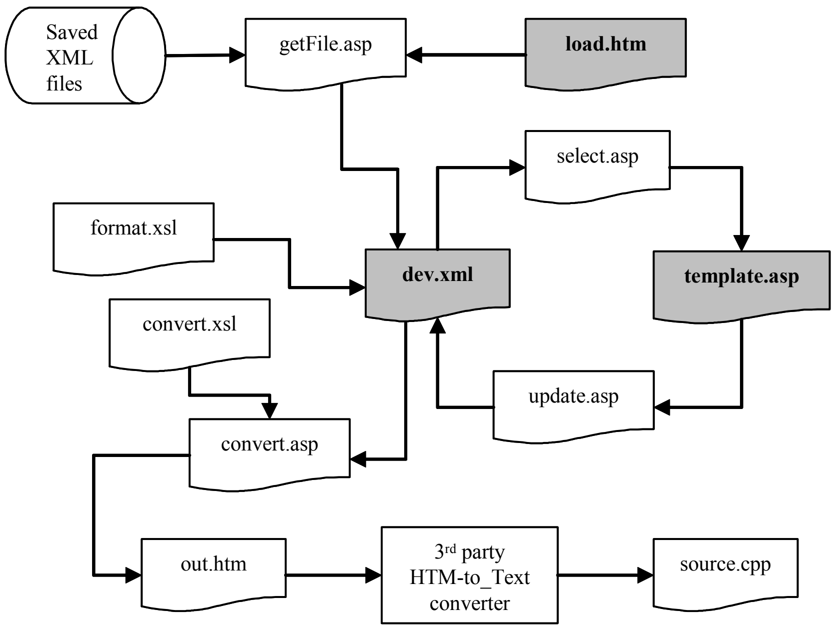


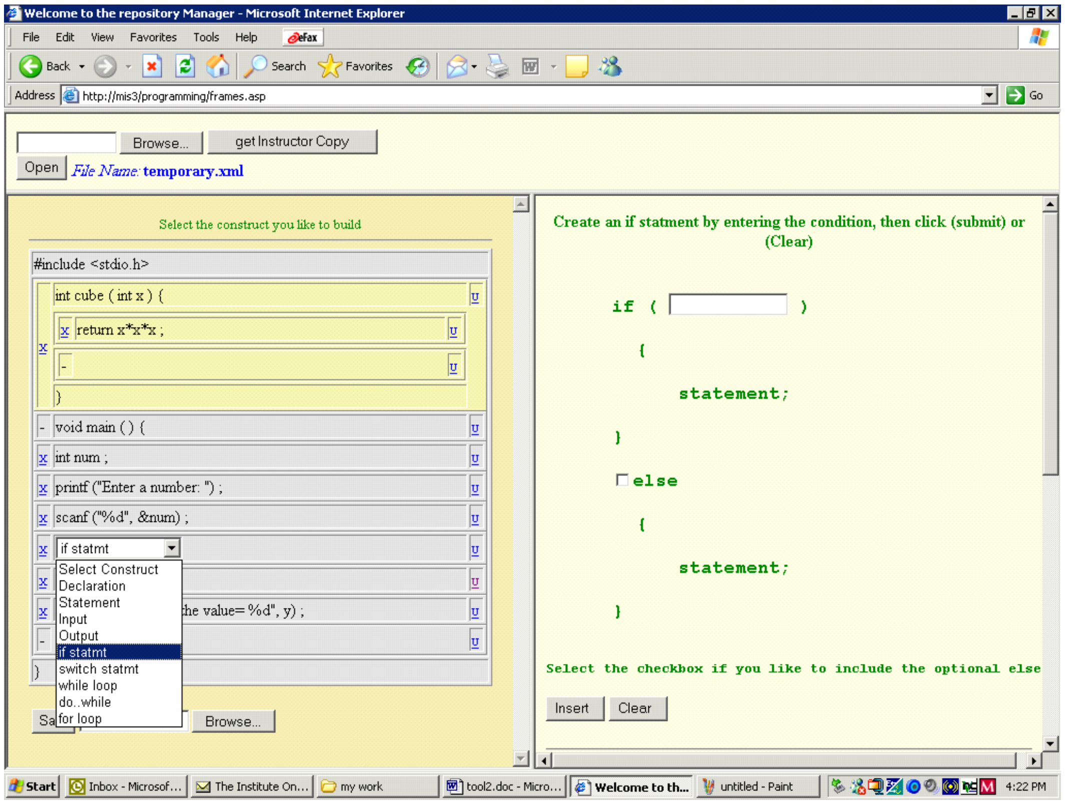
Figure 1: The system components (shaded documents represent the client GUI and the rest are running within the server).

plete to produce a source file. From the *load.htm* frame, a student can create his/her own copy of the instructor's template or retrieve one of his/her previously saved xml files using *getFile.asp.* The saved file is retrieved from the user's personal folder that is created in the server when the user's account is created. The system saves all programs in the XML format for integration and interoperability.

After loading the instructor's template or previously saved files, the user can add to or delete from the contents of the XML file. For any selected construct, the XML requests the related construct's template *template.asp* from the *select.asp* file. This template page contains the required fields, instructions, and example explaining the selected construct. When the template is filled in and submitted, *update.asp* will update and redisplay the contents of the XML file accordingly. This process is repeated until the development process is completed. The user then is able to save the produced program by invoking *convert.asp* that is using *convert.xsl* to produce *out.htm*. This file is also converted (using third party software) into *source.cpp* which is an ASCII text file. The text is popped-up in a simple program development editor where the user is able to compile and run it.

Program Templates

As the following figure shows (Figure 2), the user will have the program's main structure and a control (X) to delete any statement (except the main structure) and the control (U) to add a new statement. Pressing the control (U) will display a list of all possible valid statements at that location. The possible statements in the body of the main function include declaration, *if* statement, *for* loop, *while* statement and so forth. However, the selection list at the top of the program contains *#include, #define*, library inclusion, global variables and so forth.



```
#include <stdio.h>
int cube ( int x ) {
    return x*x*x ;
    -
}
void main ( ) {
int num ;
printf ("Enter a number: ") ;
scanf ("%d", &num) ;
if statmt
he value= %d", y) ;
-
}
```

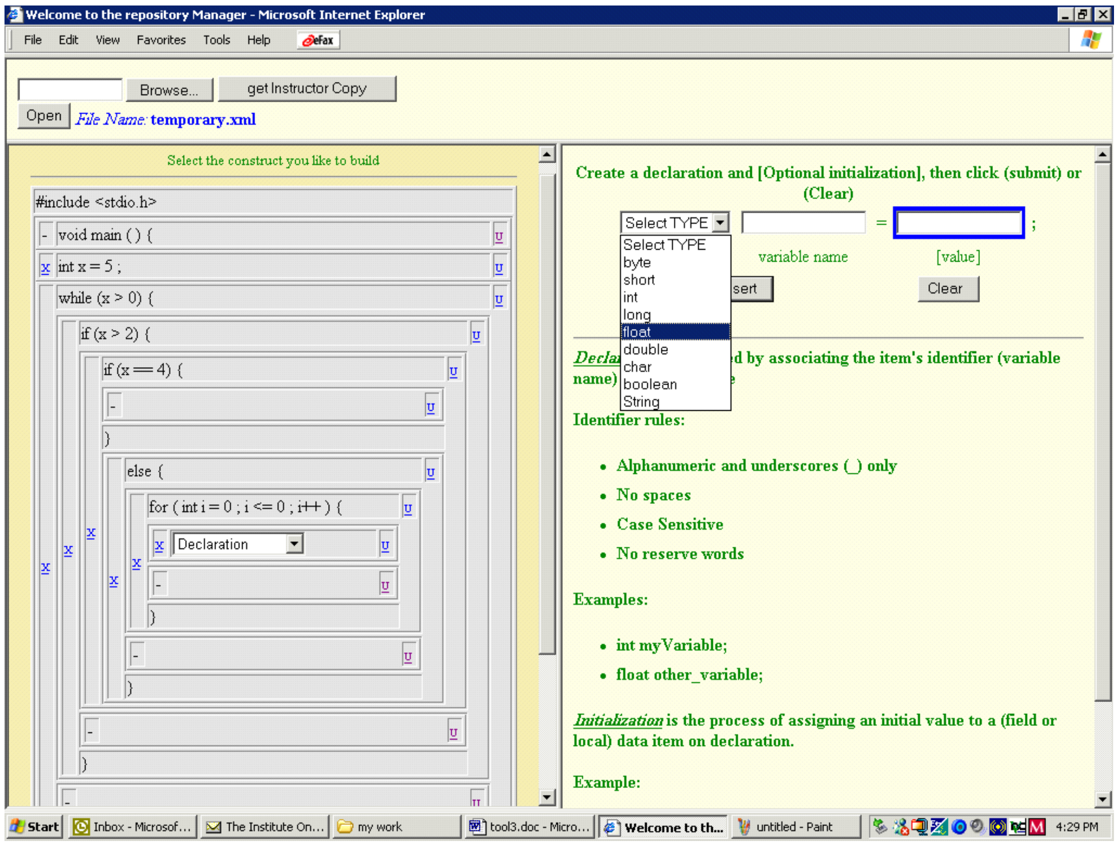
Create an if statment by entering the condition, then click (submit) or (Clear)

```
if (          )
  {
      statement;
  }
☐ else
  {
      statement;
  }
```

Select the checkbox if you like to include the optional else

Figure 2: A snapshot showing available structures within the body of the program and the if statement builder.

As mentioned previously, selecting any statement will display the necessary fields that are required to be filled by the user in the frame located at the right as well as a description and example for the selected statement. Filling the fields related to the selected statement on the right hand side and pressing the "insert" button will insert the statement in the program at the left hand side. This process will assure that no mistake can be made in creating the statements and in allocating them within the program. It has been found to be very useful in creating the program in general and nested statements in particular. A sample of a nested program is shown in Figure 3.



File Name: temporary.xml

```
#include <stdio.h>
void main ( ) {
int x = 5 ;
while (x > 0) {
if (x > 2) {
if (x == 4) {
-
}
else {
for ( int i = 0 ; i <= 0 ; i++ ) {
Declaration
-
}
-
}
-
}
```

Create a declaration and [Optional initialization], then click (submit) or (Clear)

Decla... d by associating the item's identifier (variable name)

Identifier rules:

- Alphanumeric and underscores (_) only
- No spaces
- Case Sensitive
- No reserve words

Examples:

- int myVariable;
- float other_variable;

Initialization is the process of assigning an initial value to a (field or local) data item on declaration.

Example:

Figure 3: A snapshot showing the nesting and the declaration statement builder.

Experiments

An experiment has been carried out on a class of 20 students during the Summer of 2004. The course was divided into three periods (two weeks each, equivalent to 16 contact hours). The first period was devoted to an introduction to programming languages in general, and C in particular. Basic concepts of hardware, software, memory, variables, expressions, input, output and so forth were introduced during the first period of the course. During the second period, selection and looping with nesting were introduced with and without using the teaching tool. The final period was found long enough to emphasize the previous concepts and the introduction of functions, arrays, variable scopes and so forth. During this period, students were found opening and using the teaching tool without instructions or encouragement. However, after learning and mastering program structure using the teaching tool, some students were found writing correct programs directly on a text editor. There was a mid-term exam at the end of the first and second periods (Exam1 and Exam 2), and a comprehensive final exam at the end of the third period (Final). We will refer to the experimental class mentioned above as the section under experiment (SUE). The students' performance results of the SUE were compared with the results from three other sections of the same course, as follows: A section taught by the same instructor of the experimental section (SUE), in a previous semester using the conventional class-teaching method, i.e. without using the teaching tool (we will refer to this as the Instructor's Conventional Section, or ICS). The Instructor made sure that the examination structure and its level of difficulty were kept similar for both sections, SUE and ICS. Two other sections (referred to as SEC 2 and SEC 3) were taught in parallel with the SUE, but by other instructors and without using the teaching tool. All the students of the above four sections had roughly the same level of academic preparedness. As demonstrated by the ANOVA paired sample tests for all the four sections (see Table 1), no significant differences in students' abilities (as measured by their cumulative GPAs) have been identified.

Table 1: ANOVA (Post Hoc) Tests of Differences in GPAs (*sig* in parenthesis)

	SEC 2	**SEC 3**	**ICS**
SUE	0.187 (*0.749*)	0.216 (*0.631*)	-0.149 (*0.883*)
SEC 2		0.030 (*0.999*)	-0.336 (*0.290*)
SEC 3			-0.365 (*0.198*)

Experimental Results

The examination results from the SUE were compared with the results from the other three sections mentioned above. The first and second mid-term exams were different but the final exam was uniform to all the three sections (i.e., SUE, SEC 2, and SEC 3). For the SUE, no significant difference was expected between the first and second mid-term exams, as the teaching tool was introduced towards the end of the second period and the real effect of the tool should appear during the final period. However, two distinct groups of students already started to form during the second exam as some students started to have a better control of the language, thanks to the teaching tool, whereas the majority concentrated around the mean value of the group.

As the contents, concepts and complexity of the material are different from one period to another during the semester where the level of difficulty was increasing from the first period, with the lowest, towards the final, with the highest difficulty, simple comparison between the scores achieved will therefore be inappropriate. To have a more realistic comparison, the following distribution around the mean was computed for each exam score:

$$Z_i = X_i - \mu$$

where Z_i is the deviation of the score X_i of the ith student from the mean μ. This shows the distribution of scores that can be compared around the zero center.

We conducted a (post hoc) ANOVA test on the differences between the means of the scores for the exams of the three sections. Table 2 shows the results for the SUE, where the significance value is shown is parenthesis. As demonstrated by Table 2, there are significant differences between the Final and both Exam 1 and Exam 2, but no significant difference between Exam 1 and Exam 2. This indicates that the teaching tool has been effective in improving the students' scores in the final exam.

Table 2: ANOVA (Post Hoc) Tests for the SUE (*sig* in parenthesis)

	Exam 2	Final
Exam 1	0.067 (*0.985*)	-1.122 (*0.027*) *
Exam 2		-1.198 (*0.018*) *

** Significant at 5%*

As a comparison, tables 3, 4, & 5 present similar tests for sections SEC 2, SEC 3, & ICS, respectively. As Table 3 indicates, there are no significant differences between the scores of the three exams for SEC 2. Similarly, no significant differences between the three exams can be reported for sections SEC 3 or ICS, as indicated by Table 4 and Table 5, respectively. For these sections (taught without the teaching tool), no significant performance differences can be identified during any of the three exams.

Table 3: ANOVA (Post Hoc) Tests for SEC 2 (*sig* in parenthesis)

	Exam 2	Final
Exam 1	-0.334 (*0.666*)	-0.707 (*0.186*)
Exam 2		-0.373 (*0.605*)

Table 4: ANOVA (Post Hoc) Tests for SEC 3 (*sig* in parenthesis)

	Exam 2	Final
Exam 1	0.283 (*0.857*)	0.386 (*0.752*)
Exam 2		0.102 (*0.980*)

Table 5: ANOVA (Post Hoc) Tests for the ICS (*sig* in parenthesis)

	Exam 2	Final
Exam 1	-0.390 (*0.615*)	-0.170 (*0.910*)
Exam 2		-0.220 (*0.885*)

It is interesting to note that the significant differences for the SUE resulted from an improvement in the students' performance in the final exam due to the use of the teaching tool, as pointed to above. To test the improvement impact on the final exam for the SUE, the final exams of the four sections were compared, as reported in Table 6. Even though there were certainly some extraneous factors related to differences in instructors, teaching approaches, etc., between the four sections, the final results for the SUE are still superior to those of sections SEC 3 and ICS, and slightly better than those for SEC 2. The authors suspect the existence of these extraneous factors to be the reason behind the insignificance of the difference between the final results for SUE versus SEC 2.

Table 6: ANOVA (Post Hoc) Tests of Differences in Final Exams (*sig* in parenthesis)

	SEC 2	SEC 3	ICS
SUE	1.026 (*0.183*)	1.370 (*0.023*)*	1.242 (*0.045*)*
SEC 2		0.344 (*0.891*)	0.217 (*0.969*)
SEC 3			-0.127 (*0.991*)

** Significant at 5%*

Discussion

To realize the significance of students' performances due to the use of the teaching tool, the three exams (two midterms and a final) of the SUE were compared, as reported above. The first exam covered the basic concepts with a minimal number of constructs. The teaching tool had not been used during the preparation period for this exam. The performance of students was expected to be high due to the introductory nature of the material covered.

The major programming constructs were introduced during the second period of the semester. The students' performance in the second exam that followed this period was expected to be lower than that of the first exam due to the relatively more difficult concepts covered. Since the teaching tool was introduced at the middle of this period, not much improvement in the performance of the experimental section was therefore expected. In fact, because the tool had just been introduced before the exam and due to the difficulty of the concepts covered, the result for the second exam was worse than that for the first exam.

As indicated by Table 2 for section SUE, the teaching tool has significantly improved the results for the final exam, compared with the results of the first two exams that did not benefit from the use of this tool. On the other hand, the relatively more difficult material and exams affected the performance of the other sections (SEC 2 & SEC 3) in which the teaching tool was not used (see Tables 3-5). The final exam was the hardest and was uniform for all the sections. However, a significant number of the students in SUE performed very well, whereas a small number of the students still had difficulties related primarily to the acquisition of design and creative solution skills.

Exam results in programming courses are usually not normally distributed. This phenomenon didn't appear clear to us during the first exam. However, it appeared clearer in the second and the final exams. For the final exam, the chart shows two peaks of low and high scores (see Figure 4). This pattern is noticed in most of programming courses. The reason for that is the influence of the ability to think logically on the design and creativity skills, particularly in a first programming course. This means that some of the students in our samples found the course a very difficult subject that even the teaching tool could not help them in enhancing these skills. However, other students have the ability of logical thinking and therefore

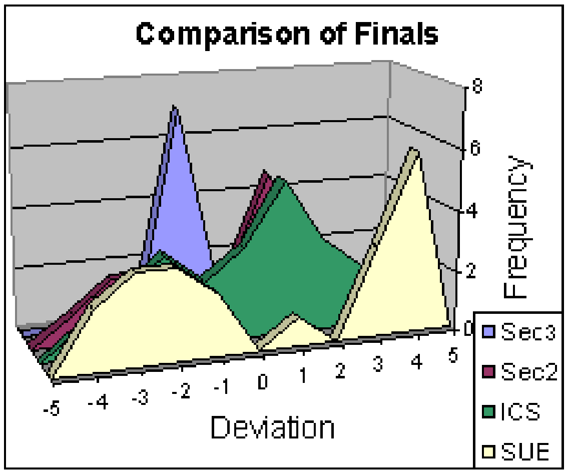


Figure 4: The final exam distribution comparison between the four sections.

gained superior performance that was evidenced during the second and the final exams. Having two peaks of performance in the final exam of a programming course appears normal as some students have weak logical skills and this issue needs further investigation. This pattern of two peaks has appeared in the entire four sections and has been found to be very clear in the case of the students who used the teaching tool, which means that students with logical thinking skills have improved on these skills. The reason for that is the help offered by the tool in eliminating the syntax problem and allowing the students to concentrate on the problem-solving ability. The idea is shown clearly when the final results are compared for the four sections as shown in Figure 4.

The introduction of the teaching tool has been found to be an effective solution to the syntax-learning problem. The tool has helped in solving the dilemma mentioned earlier regarding the time spent on teaching the language syntax, leaving little time for developing other skills. MIS students can now easily switch to C or Java syntax without falling in the syntax error trap. Some of the SUE students later used the tool heavily in building their programs at the beginning of the Java course (second programming course).

Students of first programming course are a mixture of different disciplines and background. The teaching tool has helped in bringing all students to the same level in a short period of time. This offered the instructor more time to concentrate on the concepts rather than syntax and its related errors.

Using the tool, the instructor can supply an incomplete program that is loaded by students to complete a specific requirement. This will encourage the collaboration through the gathering of students in groups to discuss the problem as a team. The instructor can regularly change or add more requirements to the same problem using the completion strategy approach used by Chang, Chiao, Chen, and Hsiao (2000). This approach emphasizes the design and creative thinking skills that are usually ignored when the students spend their time correcting the most common syntax errors.

Conclusions and Future Work

This paper presented an experiment with the development and testing of a web-based teaching tool for programming courses. The tool was motivated by our experience teaching an introductory programming course in an MIS degree program at our institution. The central issue was the excessive amount of class time spent on teaching the language syntax, which limits, and often precludes, the ability to achieve the other two fundamental programming objectives of developing program design skills and creativity in the problem solution.

The teaching tool we developed has produced interesting results that show clearly that the tool was effective in speeding up teaching and learning the language syntax. In addition to achieving the objective of speeding up syntax learning, the teaching tool includes the following salient features:

- The tool can be used in creating templates that enforce the important concepts and constructs.
- The tool is flexible and allows switching from a language to another.
- It helps to bridge the gap between students from various backgrounds and bring them up to an equal level at an early stage in the course.
- It helps to increase the student’s enthusiasm and confidence in writing correct programs.
- It can be used as a web-based self learning tool that will reduce the need for teaching and learning assistance.

Although most of the above mentioned issues have been addressed in this work, further investigations might be needed. In our next work a survey will be conducted to get a better understanding of students' opinions about the tool and its effectiveness. Based on the survey results, more efforts might be added to enhance the tool to include more features. The enhanced tool may include the ability of automatic marking of students' assignments based on the template that is continuously modified and controlled by the instructor who can prepare a pattern of learning curve.

More emphasis can be added to personalization and distance learning that will help enrolled students in enhancing their performance, as well as assist in self and distant-learning courses. Finally, the clearly distinguishable peaks of performances might be used as a measure in selecting students in the MIS program, which traditionally has a higher demand than other programs in our college.

References

Ala-Mutka, K., Uimonen, T., & Jarvinen, H.M. (2004). Supporting students in C++ programming courses with automatic program style assessment. *Journal of Information Technology Education, 3*, 245-262. Available at http://jite.org/documents/Vol3/v3p245-262-135.pdf

Allen, R., Grant, D., & Smith, R. (1996). Using Ada as the first programming language: A retrospective. *1996 International Conference on Software Engineering: Education and Practice (SE:EP '96)*, 234-241.

Baldwin, L., & Kuljis, J. (2000). Visualisation techniques for learning and teaching programming. *22nd International Conference on Information Technology Interfaces ITI 2000, June 13-16, Pula, Croatia.*

Bayman, P., & Mayer, R. (1998). Using conceptual models to teach Basic computer programming. *Journal of Educational Psychology, 80*(3), 291-298.

Campbell, W., & Bolker, E. (2002). Teaching programming by immersion, reading and writing. *32nd ASEE/IEEE Frontiers in Education Conference, T4G-23, November 6-9, Boston, MA.*

Chang, K., Chiao, B., Chen, S., & Hsiao, R. (2000). A programming learning system for beginners—A completion strategy approach. *IEEE Transactions on Education, 43*(2)*,* 211-220.

Churcher, N., & Tempero, E. (1998). Java as a first programming language. In S. MacDonell (Ed.), *Software engineering: Education and practice, Dunedin, January.* Workshop Report. IEEE Press.

Galy, C., & Waldron, J. (2002*).* Introductory programming, problem solving and computer assisted assessment*. The 6th International Computer Assisted Assessment Conference. Burleigh Court International Conference Centre, 9-10th July, Loughborough University, UK.* 95-107.

Gayo-Avello, D., & Fernandez-Cuervo, H. (2003). Online self-assessment as a learning method. *Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies (ICSL 03).*

Guindon, R. (1990), Designing the design process: exploiting opportunistic thoughts. *Human Computer Interaction, 5,* 305-344.

Guzdial, M., McCracken, W., & Elliott, A. (1997), Task specific programming languages as a first programming language. *1997 Frontiers in Education Conference.*

Jeffries, R., Turner, P., Polson, G., & Atwood, M. (1981). The processes involved in designing software. In J.R Anderson (Ed), *Cognitive skills and their acquisition* (pp. 255-283). Hillsdale, NJ: Lawrence Erlbaum Associates.

Kim, J., & Lerch, F. (1997). Why is programming (sometimes) so difficult? Programming as scientific discovery in multiple problem spaces. *Information Systems Research, 8*(1), 25-50.

Kurland, D., Pea, R., Clement, C., & Mawby R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research, 2*(4), 429-457.

Letovsky, S. (1986). Cognitive processes in program comprehension. In E. Solway and S. lyengar (Eds), *Empirical studies of programmers* (pp. 58-79). Norwood, NJ: Ablex Publishing.

Linn, M., & Dalbey, J. (1989). Cognitive consequences of programming instruction. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 57-81). Hillsdale, NJ: Lawrence Erlbaum.

Mayer, R. (1989). The psychology of how novices learn programming. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 129-159). Hillsdale, NJ: Lawrence Erlbaum.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., et al. (2001). An international multi-institutional study of introductory programming courses*. Report by the ITiCSE 2001 Working Group on Assessment of Programming Skills of First-year CS students.*

McGill, T. & Volet, S. (1997., A conceptual framework for analyzing students' knowledge of programming. *Journal of Research on Computing in Education, 29*(3), 276-297.

Moor, Brian D & Deek, Fadi P. (2006). On the design and development of UML-based visual environment for novice programmers. *Journal of Information Technology Education, 5*, 53-76. Available at http://jite.org/documents/Vol5/v5p053-076Moor29.pdf

Parker, K.R., Chao, J.T., Ottaway, T.A., & Chang, J. (2006). A formal language selection process for introductory programming courses. *Journal of Information Technology Education*, *5*, 133-151. Available at http://jite.org/documents/Vol5/v5p133-151Parker140.pdf

Reinfelds, J. (1998), Multiple or single first programming languages. *1998 International Conference on Software Engineering: Education & Practice, January,* 405.

Sheard, J., & Hagan, D. (1997). Experiences with teaching object-oriented concepts to introductory programming students using C++. *Technology of Object-Oriented Languages and Systems-Tools, 24, IEEE Technology,* 310.

Simon, H. (1973). The structure of ill-structured problems. *Artificial Intelligence, 4,* 181-201.

Soloway, E. (1993). Should we teach students to program? *Communications of the ACM, 36*(10)*,* 21-24.

Biographies

Dr. **Samer Al-Imamy** is currently working at the Higher Colleges of Technology, UAE and having previously been at Management Information Systems at the University of Sharjah, UAE, and held academic positions in Oman, New Zealand, and Yemen as well as positions in industry. Dr. Al-Imamy's research interests include Software Development, E-Learning, and Pattern Recognition. His teaching interests include Software Engineering, Object-Oriented Design, and Programming languages. Dr. Al-Imamy holds a Ph.D. in Computer from Reading University, UK.

Dr. **Javanshir Alizadeh** is currently an Associate Professor at Faculty of Computer Science and Engineering at Ajman University of Science & Technology Network, UAE and having previously been at Management Information Systems Department, University of Sharjah, United Arab Emirates. His current research interests' area includes Decision Support Systems, System Analysis and E-Commerce.

Dr. **Mohamed A. Nour** is an assistant professor of information systems at the University of Sharjah, UAE, having previously held academic appointments at the University of Central Arkansas, USA, and the United Arab Emirate University, UAE. He holds a Master of Accountancy and an MBA from Miami University, Oxford, Ohio and a Ph.D. in Information Systems from Kent State University, Kent, Ohio. His current research interests include Knowledge Management, Artificial Intelligence, heuristics and algorithms, data warehousing and data mining, E-Commerce and E-Government. He has published in such journals as *Government Information Quarterly, Information Systems Management, Information & Management, and European Journal of Operational Research.*