SAMTool, a Tool for Deducing and Implementing Loop Patterns

José Luis Fernández Alemán
Department of Computer Science and Systems
University of Murcia
Espinardo, Spain
aleman@um.es

Youssef Oufaska
Department of Computer Science and Systems
University of Murcia
Espinardo, Spain
oufaska@um.es

ABSTRACT
This paper presents a proposal aimed at active learning of iterative design in CS1. Two main contributions are made: first, a novel approach to tackle iterative algorithms design problems based on inductive reasoning, sequential access models and algorithmic schemes; second, a web tool called SAMTool (Sequential Access Model Tool) to support this approach. Students use SAMTool in three stages: (1) to obtain problem-dependent code to solve an iterative problem (2) to define an algorithmic scheme in a sequential access model; (3) to generate a program starting from a problem, an algorithmic scheme, a programming language and a sequence in the chosen programming language. The effectiveness of SAMTool as a tool to enhance learning has been demonstrated through empirical evaluation on a course with 114 undergraduate students. In a controlled study, programming students trained in the use of SAMTool were found to make fewer errors than a control group, which suggests that our approach helps students to construct iterative algorithms correctly.

Categories and Subject Descriptors
K.3.1 [Computers and education]: Computer Uses in Education; K.3.2 [Computers and education]: Computer and Information Science Education - computer science education

General Terms
Algorithms, Design, Experimentation, Human Factors

Keywords
Web tool, programming, e-learning

1. INTRODUCTION
Most of the literature concurs that learning to program is difficult, particularly the logic structures, conditionals, arrays and recursion. Most students have deficits related to problem solving, design, and expressing a design as an actual program [10]. In particular, a large number of studies have shown that CS1 students do not properly understand basic loops [13]. Although this control abstraction is apparently simple, many errors are made by novice (and intermediate) programmers. Several works have reported on the high rate of bugs associated with loops and conditionals, as opposed to those associated with other constructs [10].

Theory and research show that experts, when designing programs, unconsciously categorize problems through applying patterns which novices lack [5][13]. As pointed out by Pratt [8], most loop control computations fall in a few standard patterns, mostly related to the sequential processing of elements of data structures. In a pattern-based instruction, students should acquire the ability to recognize what pattern can be applied to solve a given problem [5][7]. SAMTool is a web tool that supports the approach proposed in [1], which describes a method to deduce and implement loop patterns, taking into account four sequential access models.

The rest of the paper is structured as follows: Section 2 justifies the importance of this research by reviewing the related literature. Section 3 presents SAMTool, a tool that supports the construction of iterative algorithmic schemes and the generation of customized iterative programs. Section 4 describes an experiment in which students who used SAMTool had more success in debugging, comprehending and developing iterative algorithms than those who used a traditional learning strategy. Finally, section 5 shows some conclusions and outlines future work.

2. RELATED WORK
After studying loops in a large Pascal program, Pratt [8] concludes that most control computations follow a few general patterns, mostly related to the sequential processing or the sequential searching of the elements of data structures. Waters [12] presents a proposal for devising loops from four plan building methods: augmentation, filtering, basic loop and interleaving. In a study described by Soloway et al. [11], the relationship between the programming language constructs and the preferred cognitive strategy of the programmers is tested. The authors identify two traversing strategies called: process/read and read/process. The outcomes of the study show that students find it easier to construct loops correctly when the programming language offers statements which allow them to choose their preferred strategy. East and colleagues [5] present a pattern-based learning model. They affirm that programmers compare any new problem-solving situation with previous experience. Five patterns...
were selected for developing relatively small programs and modules. Astrachan and Wallingford [2] describe some basic loop patterns. The authors classify loop patterns according to different types of traversing and searching. Proulx [9] presents a framework for an introductory computer science course. The proposed course is based on elementary programming and eleven collections of design patterns such as repetition, traversal, and cumulative. To our knowledge, there are few tools that support learning based on loop patterns. The PROUST system [6] uses patterns to teach Pascal programming. The programming patterns are used internally by the system, so students do not have access to them. In [4], a tool called PROPAT to learn how to program pedagogical patterns in C is presented. The PROPAT system allows teachers to gather, insert, and remove patterns and exercises, while the students can solve a list of proposal exercises by using the previously introduced patterns. We take these tools one step further, by allowing users to select the programming language used to implement the loop patterns.

In the papers cited above, the loop patterns are described in a particular access model. Each pattern is given as a recipe, with no a discovery and reflection process. Our approach also offers instructive material to teach students to cope with the variability of algorithmic schemes according to the statements unrolled from the loop, such as processing an element of a sequence, checking if a certain property holds or reading a sequence element. The aim is to achieve learning through exploration, in which students create their own mental models rather than merely receive and store knowledge transmitted by the teacher.

3. SAMTOOL

SAMTool is a server/client application which has been implemented using JSP/Servers. The client side is used for composing queries and visualizing the corresponding results by using JavaScript and AJAX. An open-source servlet container, Tomcat, is used to provide HTTP services. Data are stored in and retrieved from memory using MySQL database through Hibernate system. A prototype of this tool is working in /docentis.inf.um.es:8180/SAMToolv1.1/en/.

Interesting to our purposes, SAMTool is used as a web tool for deducing and implementing iterative algorithms. To illustrate the proposal, this section will describe how students can obtain traversing algorithmic schemes in a sequential access model and generate code starting from a problem, an algorithmic scheme, a programming language and a sequence type in the chosen programming language. The assignments proposed in our course are designed to cover the cognitive domain of Bloom’s Taxonomy [3]. The correspondence between these categories and some of our educational activities is shown in Table 1. SAMTool aids in achieving these pedagogical goals.

SAMTool has a web-based interface, which is different for the users (students) and system administrators (teachers). For example, a student can construct algorithmic schemes and generate programs. In contrast, a teacher can introduce new programming languages, including their looping constructs and conditional statements, as well as create and associate sequence types to a programming language. Both students and teachers can select the language of the interface (currently, English or Spanish).

Table 1: Educational activities in the cognitive domain of Bloom’s Taxonomy

<table>
<thead>
<tr>
<th>Category and educational activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge</td>
</tr>
<tr>
<td>Comprehension</td>
</tr>
<tr>
<td>Application</td>
</tr>
<tr>
<td>Analysis</td>
</tr>
<tr>
<td>Synthesis</td>
</tr>
<tr>
<td>Evaluation</td>
</tr>
</tbody>
</table>

The following steps are to be taken to generate a program in SAMTool:

1. Configure SAMTool. Introduce control constructs and conditional statements of the programming languages used. Several sequence types with their operations are also introduced by the teacher through SAMTool (teacher’s view).

2. Solve the proposed exercise (iterative problem) by using inductive reasoning. Students choose a sequence type \( S \) in the programming language \( L \) used in laboratory practices. Students then write both the base and general cases (treatments) in \( L \) on their own and introduce the resulting code in SAMTool (student’s view).

3. Create an algorithmic scheme according to the sequential access model associated to \( S \) by using SAMTool (student’s view). The resulting algorithmic scheme is displayed in pseudocode and can be stored in the SAMTool’s database.

4. Produce a program. SAMTool allows students to automatically implement a solution to any problem proposed by mapping the algorithmic scheme (obtained in Step 3) from pseudocode to \( L \). The problem-independent and problem-dependent code in \( L \) is generated from the data gathered in Steps 1 and 2, respectively.

To illustrate the resolution of algorithmic problems using SAMTool, let us consider the following problem: "Problem 1. Given an integer sequence represented by a simple linked list in \( C \), count the number of integers greater than 0".

3.1 Paving the Way for Implementing Loop Patterns (Step 1)

One of the strengths of SAMTool is its flexibility. The tool can be customized for several programming languages according to the needs in the laboratory practices. From the administrator’s view, teachers can introduce looping constructs and conditional statements of any programming language with the only restriction that the structure of such statements is defined in terms of a common abstract grammar. Figure 1(a) shows the teacher’s interface used to insert loop constructs, particularly a while loop in \( C \).

Sequence is a fundamental concept in programming. It is found in the values taken by variables (implicit sequences), keyboard buffer or data types such as files and strings. Furthermore, sequences can be represented by means of other data types like arrays, or implemented using pointers.
Two operations are used to access a sequence. When the sequence is a data type, these operations belong to the data type interface. The first access operation is used to start the sequential access. In accordance with the underlying model, two kinds of start operations are found. The former, named `Start` here, obtains the first element of the sequence, if the sequence is not empty. The latter, named `Initiate` here, obtains none of the elements, it only prepares the sequential access. The second access operation, named `Advance` here, is used to continue the sequential traversal. Its execution gives rise to the following element. According to the underlying model, after executing an `Advance` operation, the end of the sequence is reached when the last element is obtained or when the last element is exceeded, by using a query function, here named `IsLast` and `IsEnd`, respectively. As a result of combining the two ways of initiating and ending the traversal of a sequence, four sequential access models emerge [1]. Traversing algorithmic schemes are constructed starting from these four abstract models of control. In most cases, one of the above models will be present in any sequential traversal. The first decision of a programmer is to determine which model is suitable for the sequence of the problem.

To complete Step 1, the teacher must define the sequence types, with their operations, for each programming language previously inserted in the tool. Figure 1(b) shows the teacher’s interface used to add the C statements and declarations (Table 2(a)) of a linked list (Problem 1) in C to SAMTool, according to the first sequential access model. Table 2(b) shows the implicit mapping between the abstract model (heading `Sequence op.`) and the operations of the linked list in C (heading `C statement`). Note that, if the `IsLast` operation is used instead of the `IsEnd` operation, then a linked list according to the third sequential access model can also be defined.

Table 2: Representation of the sequence S by pointers in C

<table>
<thead>
<tr>
<th>(a) Declarations</th>
<th>(b) Mapping</th>
</tr>
</thead>
</table>
| ```
#include <stdlib>

typedef struct n {
    int value;
    struct n *sig;
} node;

typedef node *node;

void *node;
``` | Sequence op. | C statement |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IsStartEmpty</code></td>
<td><code>l==NULL</code></td>
</tr>
<tr>
<td><code>CheckIsStartEmpty</code></td>
<td><code>l==NULL</code></td>
</tr>
<tr>
<td><code>CheckIsLast</code></td>
<td><code>l&gt;sig==NULL</code></td>
</tr>
<tr>
<td><code>Start</code></td>
<td><code>l=cab</code></td>
</tr>
<tr>
<td><code>Advance</code></td>
<td><code>l=l&gt;sig</code></td>
</tr>
<tr>
<td><code>Current element</code></td>
<td><code>l=value</code></td>
</tr>
</tbody>
</table>

3.2 Iterative Problems (Step 2)

Problem 1 is identified as a traversing problem. The sequential traversing schemes can vary according to the underlying sequential access model, but the treatments of each sequence element are independent of the model. In Problem 1, the base case (trivial treatment) consists of giving an initial value (0) to a variable called `counter`, whereas the general case (non-trivial treatment) would be to increment the variable `counter` by one if the current element is greater than 0, using inductive reasoning. Figure 2 shows the student’s interface used to add the initial and general treatments of the Problem 1 in C.

3.3 Building Algorithmic Schemes (Step 3)

In SAMTool, the algorithmic schemes are constructed from the student’s view. The sequential access model is chosen by using a group of radio buttons called “Sequential access model” (top left side of Figure 3). In some data types of a programming language, pairs of these operations such as (IsEnd, IsEmpty) or (Start, Advance) might be confused. For example, IsEnd and IsEmpty are mapped into `l=0` in the linked list used for the Problem 1. These options can be selected by using the group of check boxes: "Syntactic equality" (top right side of Figure 3).

To explain the construction of a traversing algorithmic scheme to students, a sequential deployment of operations is visualized in a window (bottom left side of Figure 3). Loop unrolling and rolling actions can be performed by pressing the "Unroll" button and the "Roll" button. The rolling action allows students to reduce the length of the algorithmic scheme. In contrast, the unrolling action is necessary when one or more processing statements should be moved outside the loop. From a point in any sequential deployment, operations repeat themselves. The last three operations of the segregation (Checking, Treatment and Advance in the example of Figure 3) are used to construct the loop of the algorithmic scheme (bottom right side of Figure 3). Depending on the position of the checking operation in the repeated subsequence, the right iterative statement is chosen: while-endwhile (checking is performed at the beginning), repeat-until (checking is performed at the end) or loop-exit-endloop (checking is performed in a middle point). The bottom right side of Figure 3 shows the algorithmic scheme generated for the Problem 1 on clicking on the "Create As" button.

3.4 Generating Programs (Step 4)

We believe that the best way to learn computer programming is by programming. The students have not only to reflect on algorithmic problems by themselves, but also create their programs. We find two main benefits for the students in this stage. First, producing programs by instantiating
algorithmic schemes and customizing them to a particular context shows students the advantages of thinking about the problem before writing the code. Second, coding and running programs on a computer generates motivation and enthusiasm among students.

Figure 4 shows a sequential traversing scheme instantiated for the Problem 1. This is obtained from an algorithmic scheme of the first sequential access model (Figure 3), the problem-dependent code (Figure 2) and the problem-independent code (Figure 1).

4. EMPIRICAL STUDY

SAMTool and the underlying learning approach have been used effectively on an introductory computer programming course at the University of Murcia (Spain). *Introduction to Programming* (IP) is a course which focuses on the basic procedural programming constructs of sequence, selection and iteration, using an algorithmic notation. A strong emphasis is placed on inductive reasoning and loop patterns. Students attend 2 hours/week of lectures and 1.5 hours/week of closed labs.

To measure the educational effectiveness of the scheme-based learning, an experiment was designed and conducted during the first term of the academic year 2009/10. Data were collected from 114 programming students, divided into two groups. An experimental group of 57 participants had gathered experience in constructing iterative algorithms using sequential access models. A control group of 57 subjects lacked experience of work on control models. Simple linked list was used to assess the ability of the students to adapt the algorithmic schemes to a new data structure. All students had received training in linked lists, apart from the sequential access models. The tasks were organized into four blocks. Students were given 40 minutes to perform the following tasks:

- **Algorithm debugging.** Two traversing problems and two searching problems were proposed. For each traversing problem, four algorithms according to the four sequential access models were given out. The algorithms of the first problem (D1-D4) used an array, whereas the algorithms of the second problem (D5-D8) used a simple linked list. For each searching problem, two algorithms were given out, two using an array (D9-D10) and two using a simple linked list (D11-D12). The sequential access model associated with each algorithm was not mentioned in the wording of the problem. Students had to indicate whether each algorithm was a correct solution for its corresponding problem. If not, they were asked to identify and fix the errors.

- **Algorithm comprehension.** Two searching algorithms and two traversing algorithms were given out. The first searching algorithm used an array (C1) and the second a simple linked list (C2). The first traversing algorithm used an array (C3) and the second a simple linked list (C4). Students had to explain each algorithmic solution.

- **Algorithm development.** A searching problem (DE1) and a traversing problem (DE2) were proposed. Students had to provide an algorithmic solution to each problem. The experimental group students were not required to use one of the sequential access models studied during the course.

- **Survey.** The subjects of the experimental group were asked to fill in a questionnaire about their experience with the sequential access models. It contained nine questions (Q1-Q9) to be answered by rating each item on a five-score Likert-type scale (0: *Completely disagreed*; 1: *Roughly disagreed*; 2: *Unsure*; 3: *Roughly agreed*; 4: *Completely agreed*). The tenth question (Q10) asked the subjects to give the marks (scale of 0-10) obtained in IP.

This study used one independent variable (learning method) and three dependent variables: ability to debug (DEBUG) comprehend (COMPREHEND) and develop (DEVELOP) iterative algorithms. Figure 6 shows the results of the test (questions D1-D12, C1-C4 and DE1-DE2) as two overlapping histograms. The scores cover quite a large range, with the mean (11.01) just slightly above the 60 percent mark, with a standard deviation of 4.24, and with a mode of 11, for
the experimental group. The mean and the standard deviation were 8.57 and 3.40, respectively, and with two modes of 6 and 7, for the control group. With the usual 95% confidence interval, a t-test for independent samples determined these results to be statistically significant, \( t(112) = 3.38, p = 0.010 \). As is seen from Figure 5, the experimental group had a significantly larger percentage of students who were able to correctly solve at least ten problems, with a rate of 73% for the experimental group against that of 40% for the control group.

**Figure 5: Test score distribution.**

The data also suggest that students trained with SAMTool were able to comprehend, debug and develop algorithmic schemes with the greatest of ease, with a rate of 82%, 57% and 44%, respectively, for the experimental group, and 70%, 41% and 27%, respectively, for the control group. Note that, by using SAMTool, students who have not received debugging training in CS1 become much better debuggers. The t-test for paired samples was used to compare the means of the dependent variables. It revealed that, on the average, DEBUG was below COMPREHEND \( t(56) = -6.52, p < 0.01 \), DEBUG was higher than DEVELOP \( t(56) = 2.17, p = 0.01 \), and COMPREHEND was above DEVELOP \( t(56) = 5.56, p < 0.01 \), for the experimental group. Similar differences were found in the control group. Therefore, we can conclude that, in both groups, program comprehension was the simplest activity, and that students had more difficulty developing than debugging iterative programs.

Means and standard deviations were calculated for each question on the survey. In short, participants said that they "remembered the sequential access models" (mean: 2.86; SD: 0.71). Furthermore, they did not agree that "the sequential access models were difficult to learn" (mean: 1.64; SD: 0.85). For algorithm debugging items, students were in agreement with the usefulness of the sequential access models. For algorithm comprehension items, students were unsure, but slightly biased toward roughly agreed. However, students found the sequential access models specially helpful in tackling the complexity of the most difficult task: algorithm development (mean: 3.87; SD: 0.70).

The relationship between previous knowledge on programming and the marks of the IP exam was investigated using Pearson correlation analysis. No correlation (0.06) between previous programming experience and success with the sequential access models was found. The data indicated that programming background is not required to implement algorithmic schemes based on the four sequential access models. Furthermore, results of the survey showed that there is a positive correlation (0.41) between students' acceptance of sequential access models and the marks of the IP exam.

## 5. CONCLUSIONS AND FUTURE WORK

In this paper, a novel tool-assisted approach to design loops has been presented. SAMTool provides support for constructing searching and traversing algorithmic schemes in four sequential access models and generating programs starting from iterative schemes customized for a particular sequence in a programming language. The results of a comparative study between an experimental group and a control group show that a significant improvement in the learning process can be obtained using our approach. In future work, we plan to extend SAMTool to support new algorithmic schemes. Another area for future work on SAMTool may be that of student feedback and statistics on the system's usage. Real time feedback has the potential to enhance active learning outside lecture hours. Statistical data collected by the tool could help teachers to analyze the difficulty of the problems and the evolution of students.

## 6. REFERENCES


