West Chester University
Digital Commons @ West Chester University

Computer Science | College of Arts & Sciences

2011

P2N: A Pedagogical Pattern for Teaching Computer Programming to Non-CS Majors

Zhen Jiang
West Chester University of Pennsylvania, zjiang@wcupa.edu

Eduardo B. Fernandez
Florida Atlantic University

Liang Cheng
Lehigh University

Follow this and additional works at: http://digitalcommons.wcupa.edu/compsci_facpub

Part of the Computer Sciences Commons, and the Educational Methods Commons

Recommended Citation

Jiang, Z., Fernandez, E. B., & Cheng, L. (2011). P2N: A Pedagogical Pattern for Teaching Computer Programming to Non-CS Majors. *PLoP '11 Proceedings of the 18th Conference on Pattern Languages of Programs* Retrieved from http://digitalcommons.wcupa.edu/compsci_facpub/17

This Conference Proceeding is brought to you for free and open access by the College of Arts & Sciences at Digital Commons @ West Chester University. It has been accepted for inclusion in Computer Science by an authorized administrator of Digital Commons @ West Chester University. For more information, please contact wcressler@wcupa.edu.

P2N: A Pedagogical Pattern for Teaching Computer Programming to Non-CS Majors

Zhen Jiang
Computer Science Department
West Chester University
West Chester, PA 19383
zjiang@wcupa.edu

Eduardo B. Fernandez
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
ed@cse.fau.edu

Liang Cheng
Department of Computer Science and Engineering
Lehigh University
ethlehem, PA 18015
cheng@cse.lehigh.edu

ABSTRACT

We introduce a new method for non-computer-science majors to learn computer programming, in order to quickly prepare them for their own major study or research work. Traditional computer science programs ignore the need for such quick training, forcing them to take several semesters and many foundation courses with computer-science (CS) majors. Because those students lack sufficient background knowledge, they cannot achieve the education goal as a CS graduate may have in those courses. On the other hand, the existing entry-level training focuses on the systematic study of fundamental materials for the long-term CS career. It lacks attraction to students who are pursuing immediate support for their specified applications. An effective approach is needed to attract non-CS majors and to keep them working hard on those materials with a significant technical depth. Loops are one of the basic programming structures but we often overlook the challenge in its learning process. By using our practice at West Chester University as an example, we demonstrate the challenges as well as our success for our non-CS majors to quickly learn to develop loops correctly. On one hand, we adopt the disciplined training model with many subtasks that is commonly used in China in order to cover all the required materials. On the other hand, we adopt the model that is commonly used in American classes and use commercial-off-the-shelf products, games, work templates, etc. in order to help students form the abstractions, understand the corresponding materials, gain the appropriate skills, and achieve each intermediate task goal. This pattern provides a solution for a complex education problem in a short time scale.

Categories and Subject Descriptors: L.3.6 [Science and Technology of Learning]: Technology Enhanced Learning.

General Terms: Pattern Learning, STEM Education.

Keywords: Course Curriculum, Pedagogical Pattern, Programming Teaching.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission. A prelimnary version of this paper was presented in a writers' workshop at the 18th Conference on Pattern Languages of Programs (PLoP). PLoP'11, October 21-23, Portland, Oregon, USA.
Copyright 2012 is held by the author(s). ACM 987-1-4503-1283-7.

1. INTRODUCTION

Computers have changed our world in many significant ways. Students, especially senior students, from other science majors with various backgrounds require computer skills for their own major study or research work. This creates a demand for a quick training that can prepare them in one or two semesters. Existing entry-level courses in CS0, CS1, or minor programs (e.g., [5, 6, 7, 8, 9, 10, 11, 12, 13]) that are available for those inexperienced non-CS majors usually aim to a long-term, systematic, and broad study for the CS major/career. The need for quick training is ignored. On the other hand, most of the computer applications on which they are working are commercial-off-the-shelf (COTS) and use many advanced techniques in industry that are beyond what we currently teach in the entry-level courses. A new effective learning process is needed to quickly prepare students with widely divergent background to be able to have the programming abilities of a CS graduate.

Loops [15] are one of the basic program structures. However, as indicated by Elliot Solloway, even experienced programmers have but a 50% chance of developing the correct loop when confronted with a do-while (or do-until) choice. To ensure correctness, the programmers must learn many theoretical aspects such as axiomatic semantics [22]. This is the main obstacle (or challenge) for non-CS majors in our training especially when the time of learning and practice is limited.

To our knowledge, there is no existing method to completely solve the above training problem. With the pedagogical pattern P^2N proposed in this paper, we illustrate our approach and share its success. Inspired by the Chinese training model for amateurs to quickly obtain a high level of proficiency in iPhone application development, we adopt a disciplined learning process in which the educational outcomes can be assessed easily in different steps. For instance, we teach two types of loops in class (e.g., [21]). The students will learn the development of the counter-control loop in a 6-step procedure and that of the event-control loop in a 10-step procedure respectively. To solve the common problem of the fragmented and superficial learning in the disciplined training, we provide a 3-phase solution to guarantee the accumulation of achievements in each step in an efficient way, saving the time to reach an ambitious education goal. Our approach and its contributions are summarized as follows:

1. We adopt ***COTS computer equipment***, with research activities relevant to the topics discussed in class to give students a vivid picture of the use of the entire

training process, motivating their study from the beginning and verifying their ultimate learning outcomes at the end.

2. We interpret the programming experience into ***templates***. The advanced materials can be transparent, making the programming easy to learn and to practice. Reaping the general benefits of repeating their uses, i.e., reusability and concept abstraction [19], students easily form abstraction, understand materials, and gain skills. In this way, the obstacles of each step can be conquered with limited time and the corresponding achievement can be accumulated successfully and quickly.

3. We develop ***a software tool*** to encourage students to have sufficient programming practice. Organized under our training model, each step has its own learning outcomes verified by practicing this customized software tool in different levels. This helps to assess the achievement in each step (or subtask).

The remainder is organized as follows: Section 2 describes the pattern P^2N, while Section 3 presents some conclusions.

2. PEDAGOGICAL PATTERN FOR TEACHING PROGRAMMING TO NON-CS MAJORS (P2N)

This pattern describes a method to attract a broad range of students (e.g., non-CS majors) to a condensed computer training that teaches structural programming, such as loop development. It keeps them working hard on materials with a significant technical depth. The students do not have much background in programming, so we need to introduce concepts without requiring much previous knowledge. Our approach helps taming the complex teaching tasks in a very short time frame and guarantees their quality.

2.1 Context

This pattern is applicable to a course or program with complex tasks and ambitious goal, for instance, industry training in centers that needs to prepare programmers in a short time; or computer training in universities that teaches structural programming to non-CS majors. For example, we taught loop development to junior students of the Physics department (and other 11 departments such as the Accounting department) at West Chester University (WCU) who had little or limited computer experience but required computer techniques to process the experimental data for their senior research projects.

2.2 Problem

When we teach the entry-level programming course with materials more advanced than the standard ones (e.g., [1]), such as the concept of correct program with axiomatic semantics, the time of learning and for practice becomes relatively short. Students may not have enough time to finish the practice as we usually expect for CS majors in a much longer training. The learning is fragmented and superficial. Therefore, our proposed education goal looks like too ambitious and our training tasks are complex especially when our students are non-CS majors and they lack the sufficient programming experience (or background) as prerequisites.

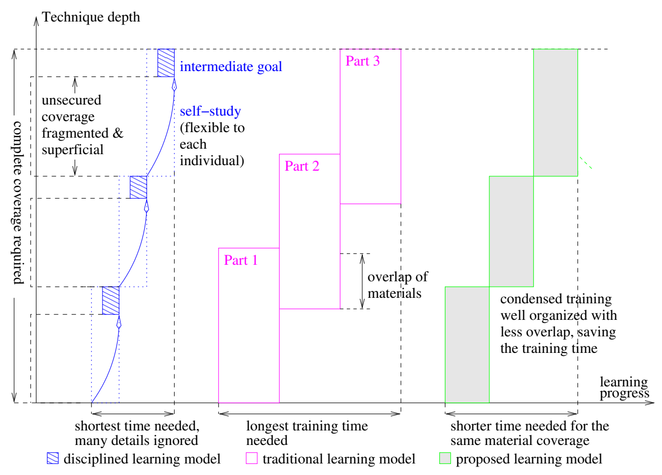

Technique depth
complete coverage required
unsecured coverage fragmented & superficial
intermediate goal
self–study (flexible to each individual)
Part 1
Part 2
Part 3
overlap of materials
condensed training well organized with less overlap, saving the training time
learning progress
shortest time needed, many details ignored
longest training time needed
shorter time needed for the same material coverage
▨ disciplined learning model
□ traditional learning model
🟩 proposed learning model

Figure 1: Comparison of different teaching styles.

The students in our class come from up to 11 different departments. Their background and motivation for taking our class are widely divergent. Thus, the synchronization of student learning progress in our class is also difficult, which increases the complexity and difficulty of our training. The above challenges in our programming training are summarized as the following constraints:

1) the difficulty of learning with critical time constraints

2) the asynchronous learning progress of students with widely divergent backgrounds, and

3) the lack of sufficient programming background as prerequisites.

To achieve the ultimate education goal in our condensed and accelerated training, we need a pedagogical method that is not only able to draw students' interest but also to keep this interest during the entire training procedure.

2.3 Forces

Traditional class training

- **Pros**: uses many examples to help students form the concept abstraction and gain programming skills;

- **Cons**: is not feasible in a course with critical time constraints.

The training is not only limited to the class time but also extended to project practice in the succeeding courses. Such training in existing bachelor's study takes several semesters and requires many core courses before any advanced topic course can be taken (i.e., the traditional teaching style shown in Figure 1 in magenta).

Many educators have noticed the advantages of teaching in different cultures (e.g., [18]). In the Chinese teaching style,

- **Pros**: the trainees are disciplined so that they can be guided to finish the work quickly;

- **Cons**: the learning is fragmented and superficial and faces a high failure ratio.

The success of iPhone application development in China proves that it is possible for amateurs to quickly obtain a high level of proficiency. In such a model (i.e. the disciplined teaching style shown in Figure 1 in blue), the trainees accomplish the study by achieving many intermediate goals. However, each sub-task does not have sufficient help to guide the trainee's analysis and thought. It is difficult for the trainee to extend his/her knowledge due to the lack of a complete background on the subject (i.e., fragmented and superficial). Moreover, the ultimate training goal cannot be reached whenever any sub-task becomes an obstacle for the trainee and this person cannot have the required thinking and analysis by him- or herself.

2.4 Solution

To schedule the learning progresses of students within a very limited time frame, we adopt the *Chinese discipline model* and design the learning process via *multiple steps*. Each step has its own intermediate goals that are relatively easy to achieve and has its outcomes quickly verified. The overlap among different steps can be reduced to the minimum, in order to avoid wasting time on any unnecessary replication. By accumulating all intermediate achievements, the ultimate learning goal can be achieved quickly with this method. In our sample training which will be discussed next, the learning of the counter-control loop and the event-control loop is divided into 6 steps and 10 steps respectively, according to their roles in the execution in the time sequence (i.e., operational semantics).

Our solution focuses on realizing the above steps that are disciplined by the Chinese training model with the American style. We find a balance of the forces of different teaching styles, taking advantages of both of them. It is implemented in 3 phases as illustrated in Figure 2 (a) for the sample loop training.

1. In phase one, in order to handle the class diversity, we adopt ***COTS computer equipment*** with research activities relevant to the training to motivate student study at the beginning. Such equipment and its sample work are also used at the end of training to verify the ultimate outcomes and the success of our synchronization of student learning progresses.

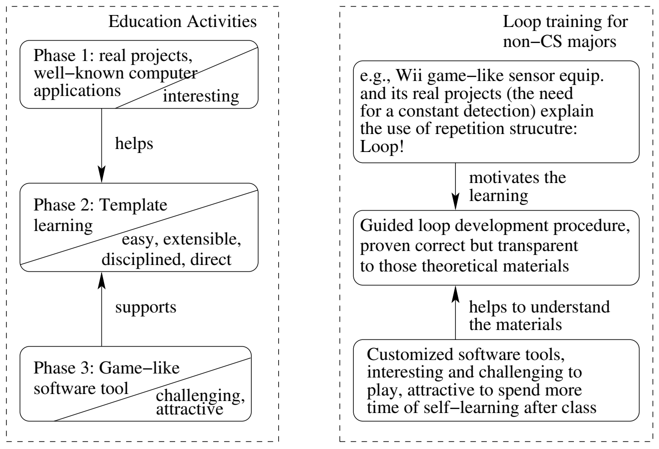
2. In phase two, in order to help students successfully conquer the obstacles in each step (or subtask), we adopt a ***template*** for the program development. In this way, the advanced materials can be transparent, making them easy to learn and to practice even when the students lack the required background (or experience) as the prerequisites.

3. In phase three, we develop ***a software tool*** to encourage students to obtain sufficient programming practice when the class time is limited. With the sample cases testing the learning progress at each step and at each level, the student achievement can be verified and synchronized. Those achievements will encourage students to continue the learning.

We make the training at each step fit the disciplines under the Chinese training style by the follows:

- The computer grading in the test on each step will force students to accomplish a good quality work, helping instructor(s) to assure the students' progress in a very limited time frame.

- The tool for the student work after class saves time in class, making the condensed training feasible.


Education Activities
Phase 1: real projects, well−known computer applications
interesting
helps
Phase 2: Template learning
easy, extensible, disciplined, direct
supports
Phase 3: Game−like software tool
challenging, attractive
Loop training for non−CS majors
e.g., Wii game−like sensor equip. and its real projects (the need for a constant detection) explain the use of repetition strucutre: Loop!
motivates the learning
Guided loop development procedure, proven correct but transparent to those theoretical materials
helps to understand the materials
Customized software tools, interesting and challenging to play, attractive to spend more time of self−learning after class

(a) Pattern design (b) Pattern application in loop study

Figure 2: The proposed pedagogical pattern (P^2N).

To guarantee the education quality at each step for inexperienced trainees to obtain the expected outcomes, we integrate the necessary processes of analyzing and thinking into learning tasks under the traditional American training mode. The student learning is organized under a pre-set template. The advanced materials can be transparent in the description of the success experience with this template. The concept abstraction provides the disciplines that we need to guide the student study. Its extensibility provides room for students to have a process of analyzing and thinking.

To guarantee the success of each step in learning, we developed a game-like tool. It balances the superficial education and its high failure rate under the Chinese style in the following aspects:

- The human-computer interaction attracts the students to spend more time after class in a self-learning process, guaranteeing the required practice.

- The computer tool will test the students' learning progress with different difficulty stages, encouraging students to learn the materials gradually and continuously.

- The computer tool uses a comprehensive set of testing cases, which helps instructor(s) to verify the completeness and quality of the training and to synchronize the students' progress.

In order to handle the background diversity in training, we use COTS products and relevant research projects that can be recognized by most students. This will motivate the students by giving a big picture of what they can do. The software tool adopted in our learning provides the flexibility for students to schedule their learning after class by themselves according to different background and level of each individual person. This also helps to synchronize the student learning progress and to assess the result of each intermediate step.

2.5 Example for pattern implementation

We use the loop training at WCU to demonstrate the proposed approach. In the following, we first list our ultimate education goals as the training requirements. Then, we show the challenges in the proposed training. The learning of loop development is organized in 3 phases, for both the 6-step procedure of the counter-control loop development and the 10-step procedure of the event-control loop development, following the criteria of national training standard for information systems security professionals [17]. Each step has its own education goal according to its different role in the execution of loop in the time sequence (i.e., the operational semantics). Phase 1 is applied when the syntax of loops and the need for the development are discussed in class. Phase 2 is applied to introduce the correct development process in class. Phase 3 is applied for students to practice in and after class in order to enhance the understanding of class materials and to obtain the required knowledge and skills. The rest of the discussion in this part focuses on how practical and effective the above 3-phase solution under the American training model helps the learning of loop development that is disciplined by the Chinese training model. Such a method will guarantee the students' necessary analyzing and thinking, while meeting all challenges. We will discuss our implementation of that 3-phase solution.

2.5.1 Requirements of a loop program training for non-Computer-Science majors

The loop is an essential program structure. Our practice is not only limited to the learning of its syntax, but also has the following highly expected outcomes in order to prepare students for the programming of the research projects in their own major:

- to know when and where a loop is needed,
- to be able to interpret a sequence of events/activities of program execution into a loop program,
- to have knowledge of the correctness of a loop program,
- to be able to verify a loop program and to correct any possible error,
- to know how to maintain the program when the situation or requirements change.

2.5.2 Constraints

The challenge is to achieve the above education goals as fast as in 3 classes (75 minutes each) or equivalently 10 days. Note that many students in such a class are from other departments such as the Performance & Arts Department. Those students are from 11 different departments and may not have sufficient background knowledge. Imagine that those inexperienced students will be asked to ensure that all code lines are correct since their very first program. An effective education model is needed.

2.5.3 3-phase solution

In phase 1, we use work on real research projects to motivate students. Without the appropriate motivation, the non-CS majors will work for credits only to meet the general education requirement for their graduation. They can easily be distracted from the required learning material according to our classroom experience and the planned education will

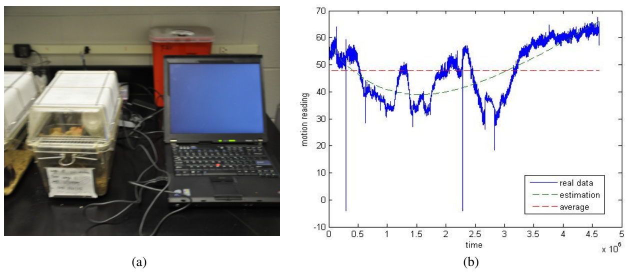


Figure 3: (a) Research work in biology laboratory at WCU with the iMote2 sensing platform. (b) Experimental results that are analyzed and processed by a loop program demands for an easy and quick development.

become very difficult. Due to the variety of backgrounds of those students, such a research result should also have a broad impact that can be recognized by most students.

For the loop study, we introduce our research [3] with the Wii [16] game-like equipment iMote2 [14] (see Figure 3). It is used to detect the change of environment in terms of the temperature and the light, continuously for every second in a 7/24 schedule. To analyze the data collected, even simply to obtain the average, a loop is needed. This introduction of our iMote2 work can bridge the student development in class with their daily life and even the jobs in industry. Unlike existing activity-driven teaching that adopts laboratory materials of wireless sensing for senior project development (e.g. [25]), our use of wireless sensing equipment focuses on the enhancement of students' interest. Once the students finish the loop training, they will have a chance later to apply the programming knowledge to change the control of sensor equipment in the sample code. This will be tested in our customized software (discussed later) and its success ratio will help to verify the ultimate learning outcomes that a CS graduate may have.

In phase 2, we adopts a template for students to easily learn the loops and their development in class. First, the trainees will follow a disciplined procedure to practice the correct loop program development. The development follows a standard, to avoid awkward codes that cause a potential hard work in the maintenance of the code. Advanced concepts and advanced techniques such as program correctness are transparent, but effectively guaranteeing the quality of programming. Then, when the trainees practice such a procedure of programming, they repeat the development of correct programs under the proposed template, accumulating the experience to gain the required skills and knowledge. As an abstract module, this procedure can satisfy different purposes for the use of loop statements, meeting the goal of our training. Note that the correctness proof for such a procedure is out of the scope of this paper and is omitted.

Figure 4 shows the statement syntax and the operational semantics of loop development. This is the template for us to explain the loop development in class. The *repetition body* consists of the code lines that are repeated many times in a sequence. The *condition* is used to stop the repetition when it becomes false. The *initialization* consists of all assignments changing the value of variables that are used in the

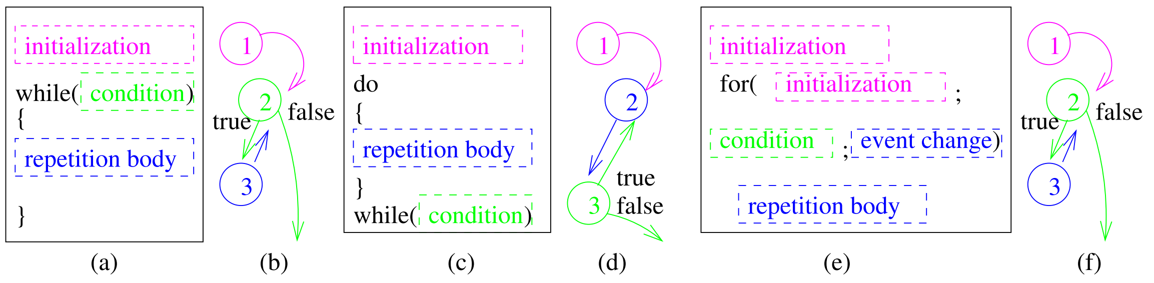


Figure 4: Statement syntax and operational semantics (i.e., the loop template for learning). (a)-(b) While loop, (c)-(d) do-while loop, and (e)-(f) for loop.

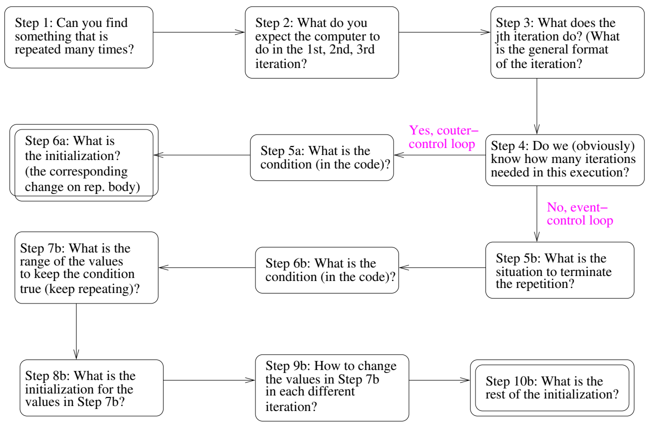

Step 1: Can you find something that is repeated many times?
Step 2: What do you expect the computer to do in the 1st, 2nd, 3rd iteration?
Step 3: What does the jth iteration do? (What is the general format of the iteration?
Step 4: Do we (obviously) know how many iterations needed in this execution?
Yes, couter–control loop
Step 5a: What is the condition (in the code)?
Step 6a: What is the initialization? (the corresponding change on rep. body)
No, event–control loop
Step 5b: What is the situation to terminate the repetition?
Step 6b: What is the condition (in the code)?
Step 7b: What is the range of the values to keep the condition true (keep repeating)?
Step 8b: What is the initialization for the values in Step 7b?
Step 9b: How to change the values in Step 7b in each different iteration?
Step 10b: What is the rest of the initialization?

Figure 5: Disciplined steps to develop loop programs (with the development template).

repetition body before the loop starts its 1^{st} iteration.

Figure 5 shows the details of those disciplined steps with our template in the loop development. The development starts from the determination of the repetition body. Then, the termination condition is decided. At the end, the initialization is finished to assure that the right code can be applied to the right pre-condition. This process is based on the operational semantics of the loop and it is easy to follow. When the number of loop iterations can be known easily, this loop is called counter-control loop; otherwise, it is called event-control loop. To simplify the development, we strongly suggest the use of the counter-control loop because it has a simple, structural regularity (see the template in Figure 6) and require fewer steps of development (i.e., less work and fewer mistakes). Note that the correctness of the loop now is easy to verify for inexperienced programmers by re-applying the same development procedure for a consistency check. Algorithm 1 summarizes the function of each step. Figures 7 and 8 show the development of a sample loop program in different ways: the counter-control loop and the event-control loop respectively. The development progress at each step is highlighted in red. The event-control loop requires more steps (i.e., steps 7−9) to determine the event descriptor and its value changes.

Our program development approach introduces the analysis and thinking to the life cycle as a) requirement analysis, b) design & coding, c) verification, and d) maintenance:

- **Analysis:** *When is a loop needed?*

  Answer: Whenever we find that something is repeated many times in a sequence (also called the repetition body).

- **Design & Coding:** *How is a loop developed?*

  Answer: First, we obtain the description of the repetition body. This forms the abstraction from all iterations, i.e., an "everyone" model that can represent all iterations. The process is similar to the development of the general process for each element in an array, but it focuses on the common part of iterations in the time sequence. This procedure also supports nested-loop development by recursively applying the same development process on the internal repetition body. Next, we complete the code (i.e., the terminating condition and the initialization part) in counter-control loop mode or event-control loop mode respectively.

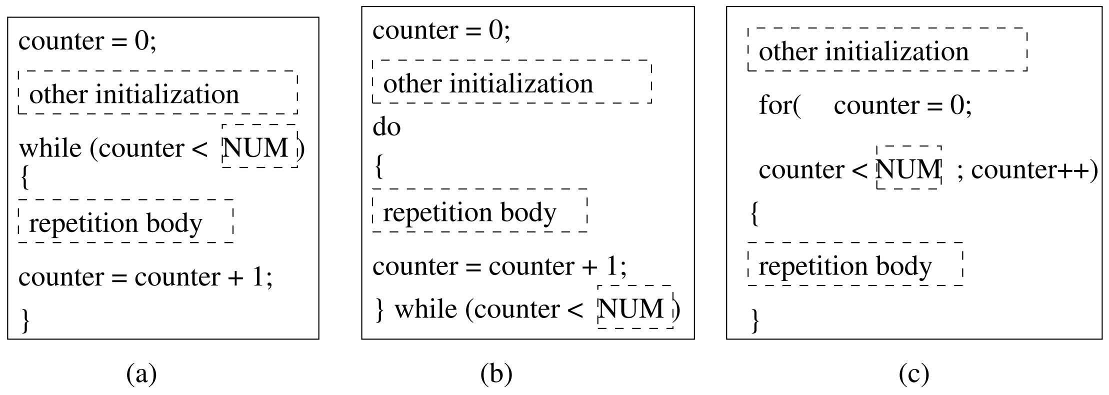
- **Verification:** *How do I know I am doing this correctly?*

  Answer: To avoid any careless mistake, we need to verify each step with the results obtained in all previous steps. For instance, the initialization development in Step 6a must be consistent with the 1^{st} iteration described in Step 2. This verification also identifies those identical programs, in order to avoid unnecessary change.

- **Maintenance:** *What about a change of code?*

  Answer: Simply repeat the above processes until everything is consistent after the check of the verification phase.

In phase 3, our approach adopts a customized software tool to help students complete this learning process by practice. Even though many advanced matters are clear in class, the students may still feel that the development procedure is too complex to follow because they are not convinced by the necessity of each part/step. Due to the limited practice time allowed in class, students need help to learn each programming step throughly by themselves after class. Our game-like software tool is attractive for students to practice the loop development with the proposed template. It will use one of 10 loop programs according to the difficulty level the student selects. For each selected loop, the computer

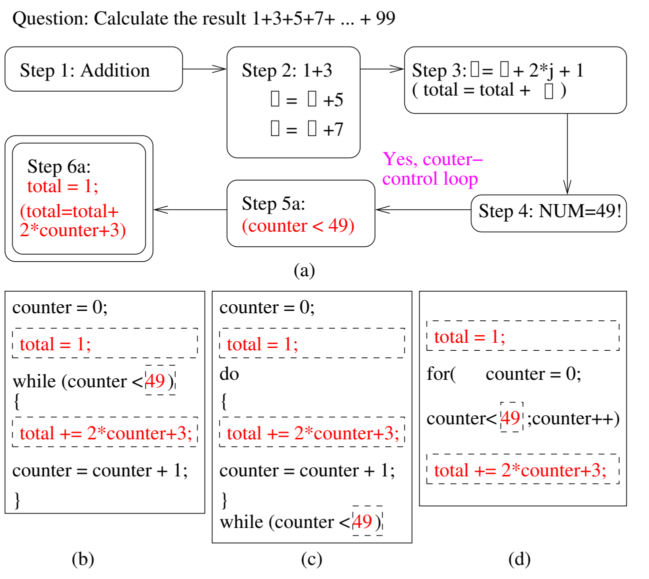


```
counter = 0;
other initialization
while (counter < NUM )
{
repetition body
counter = counter + 1;
}
```

```
counter = 0;
other initialization
do
{
repetition body
counter = counter + 1;
} while (counter < NUM )
```

```
other initialization
for(    counter = 0;
counter < NUM  ; counter++)
{
repetition body
}
```

Figure 6: Extensive template for Counter-control loop development (Dashed line encloses the part to be decided).

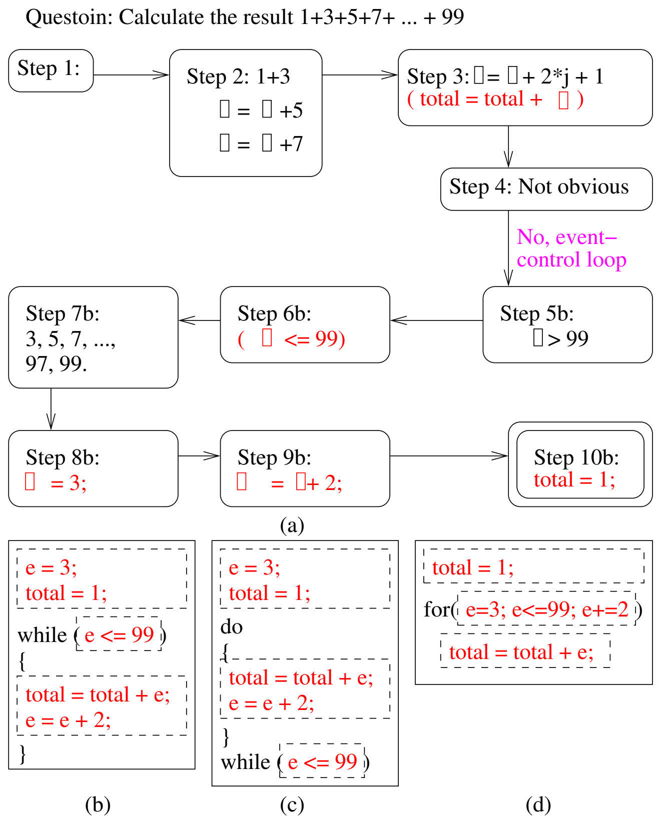


```
counter = 0;
total = 1;
while (counter <49 )
{
total += 2*counter+3;
counter = counter + 1;
}
```

```
counter = 0;
total = 1;
do
{
total += 2*counter+3;
counter = counter + 1;
}
while (counter <49 )
```

```
total = 1;
for(      counter = 0;
counter<49 ;counter++)
total += 2*counter+3;
```

Figure 7: (a) Development of a counter-control loop, highlighted in red, following the disciplined steps in Figure 5. (b), (c), and (d) The resulting program in different formats.



```
e = 3;
total = 1;
while ( e <= 99 )
{
total = total + e;
e = e + 2;
}
```

```
e = 3;
total = 1;
do
{
total = total + e;
e = e + 2;
}
while ( e <= 99 )
```

```
total = 1;
for( e=3; e<=99; e+=2 )
total = total + e;
```

Figure 8: Development (a) and resultant programs (b), (c), and (d) of an event-control loop following the disciplined steps in Figure 5.

Algorithm 1: Loop development in a disciplined procedure with the template.

1. Determine the repetition body and check the necessity of using a loop statement (Step 1 in Figure 5).
2. Find the general format of the repetition body that is derived from each iteration in the sequence of execution (Steps 2 and 3 in Figure 5).
3. If the number of iterations can be known, the program can be developed in a counter-control loop mode. Otherwise, an event-control loop is needed .
4. For a counter-control loop, provide the condition and initialization (Steps 5a and 6a in Figure 5) for the required work that has been shown in Figure 6.
5. For an event-control loop, decide the condition to terminate the repetition process (Step 5b in Figure 5). Then, determine the event description and its change in the repetition (Steps 6b − 9b in Figure 5). At last, provide the initialization (Step 10b in Figure 5) to finish the work.

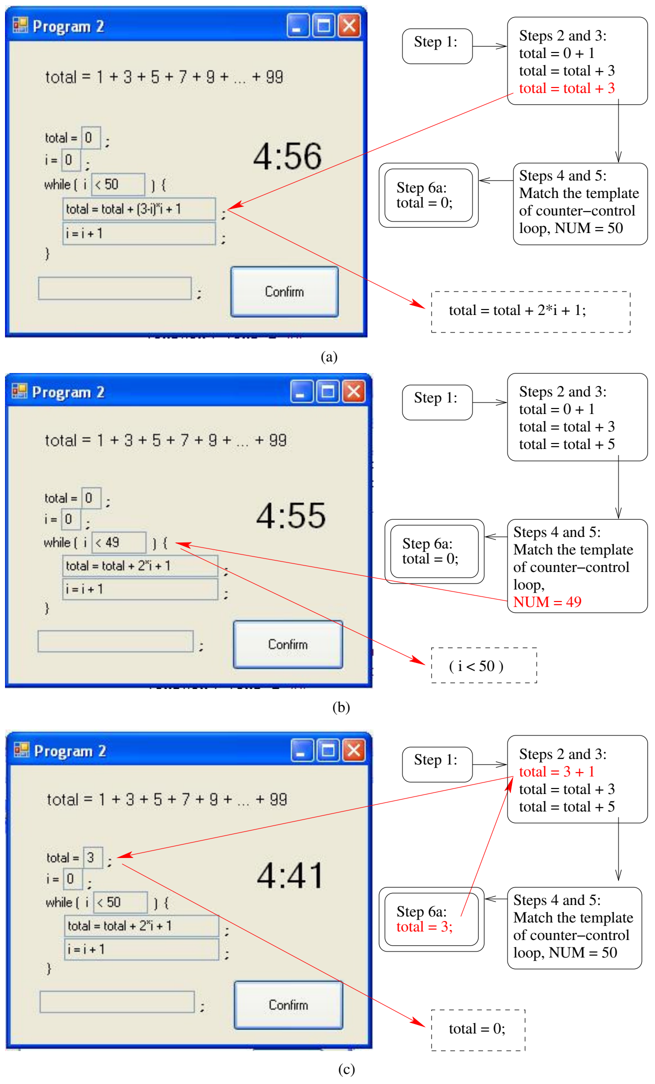
will randomly create an implementation which has one and only one error among all 12 different steps (in the template as well). The student will be asked to correct this and only this error, in 5 minutes. This hand-on exercise with up to 67 different cases for each selected loop will force this student to have a deep understanding of the entire development procedure and the corresponding template. It also provides certain flexibility for students to control the amount of exercise needed after class (e.g., hours spent on and times of trials) until they are confident with what they have learnt.

This kind of "one error to correct" game helps students to go through the development process of Algorithm 1 and to understand all the relevant details. Figure 9 illustrates the use of such a tool for assessing any step in the development procedure of the counter-control loop. For instance, in Figure 9 (a), an inconsistency is found when the 3^{rd} iteration has the result of "+3" which is different from the expected "+5", by following the development procedure. Since there is no other error, the body needs a new code to ensure all the iterations have the right addition operation. The required change "total=total+2*i+1" is obvious. In Figure 9 (b), the loop stops one iteration short (i.e., number of iterations $NUM = 49$), which demands for a change of condition ("i<50" or "i<=49"). In the case in Figure 9 (c), we can find two inconsistent places: one is in the execution of the 1^{st} iteration and the other is in the initialization. Considering that only one error produces both inconsistencies, the initialization must be changed ("total=0").

Figure 10 demonstrates a similar process for the event-control loop. We have 4 inconsistent places: two are in the iterations, one is in the value changes of the event descriptor, and one is in the initialization part. After the event initialization is changed ("i=3"), the code passes the verification and becomes 100% correct.

2.5.4 *Consequences*

Figure 11 summarizes our 9-year experience in teaching


(a)

```
total = 1 + 3 + 5 + 7 + 9 + ... + 99
total = 0 ;
i = 0 ;
while ( i < 50 ) {
    total = total + (3-i)*i + 1 ;
    i = i + 1 ;
}
```

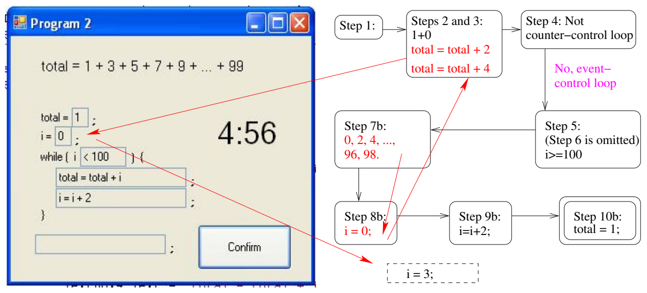
(b)

```
total = 1 + 3 + 5 + 7 + 9 + ... + 99
total = 0 ;
i = 0 ;
while ( i < 49 ) {
    total = total + 2*i + 1 ;
    i = i + 1 ;
}
```

(c)

```
total = 1 + 3 + 5 + 7 + 9 + ... + 99
total = 3 ;
i = 0 ;
while ( i < 50 ) {
    total = total + 2*i + 1 ;
    i = i + 1 ;
}
```

Figure 9: Test of the counter-control loop development with the computer tool ("one error to correct"). The red arrow indicates the thinking process.



```
total = 1 + 3 + 5 + 7 + 9 + ... + 99
total = 1 ;
i = 0 ;
while ( i < 100 ) {
    total = total + i ;
    i = i + 2 ;
}
```

Figure 10: Test of the event-control loop development with the computer tool.

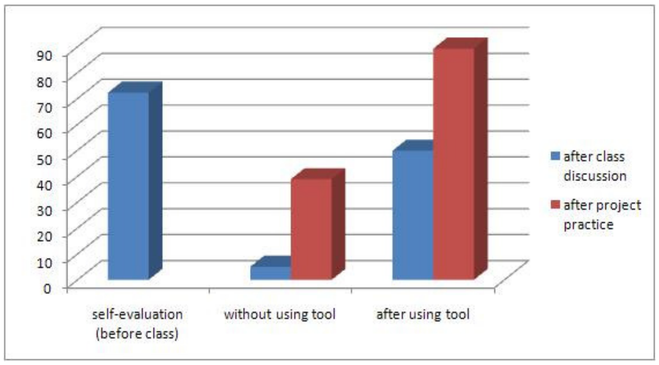


Figure 11: Outcome assessment.

loop development in a general education course for non-CS majors. After the introduction to loop syntax, the students are asked to do a self-evaluation. Most of them are confident with what they have learned and estimate that they can finish 70% of loop development work in this class. However, when we test with the software tool whether they can think about the subject matter as a CS graduate can (e.g., the analysis and thinking process in Figures 9 and 10), only 4% of the students can really finish the job.

After the introduction to our disciplined steps (in Figure 5) for the program development in our iMote2 research project, students are willing to use our proposed template and the corresponding development procedure in their exercises. After that, up to 40% of the students get the idea and pass the test. However, because the training is limited to one week only, up to 60% of the students still have a difficult time and cannot fully understand the materials. In the past 3 years, we introduced the tool in class and used it in student projects. The improvement is substantial and significant. After the discussion of such a tool in class, students learned how to develop program correctly. 50% of the students can successfully finish the required work in a quiz. Then, after their project practice with that tool, up to 90% of the students can pass the computer test, achieving our education goal (as a CS graduate can think in Figures 9 and 10) for such a condensed training.

2.6 Known uses

While many computer scientists focus on the quality of programming in large-scale and complex systems, the industry requires a more general education of computer programming for inexperienced people, in order for them to fulfill certain programming jobs. Recently, more and more universities have resorted to new training programs (e.g., various minor or certificate programs) to meet such a market demand. One of the NSA certified programs [17] at West Chester University offers undergraduate students a quick information technology (IT) training by taking only 6 courses. "CSC115 - Introduction to Computer Programming" is an entry-level programming course and is the only prerequisite course for non-computer-science majors to take our NSA-certified topic courses [12]. This course aims to structure a programming basis for the development of complex computer application in the succeeding topic courses.

Our CSC115 at WCU is a general education course for all non-Computer-Science majors, with up to 8 sessions each semester, four times the number of sessions of the entry-level programming course that we make available to Computer Science majors. Recently, its curriculum upgrade with our proposed pattern [23] has gained the attention from other departments at WCU and their students. For instance, a joint research has been initiated with the Biology department and the Physics department, in order to have more suitable projects for their students' computer training.

The corresponding education practice has also attracted the interdisciplinary collaboration from other schools. For instance, our training pattern has been adopted in a newly created summer programming training for CS majors in Shanghai Jiaotong University to enhance their programming ability, which has won the ACM International Collegiate Programming Contest 3 times in the past 8 years [2].

2.7 Related patterns

Pedagogical patterns [4, 20, 24, 26] are high-level patterns in documenting good education and training practices and experiences through design patterns. They have been recognized in many areas of training such as group work, software design, human computer interaction, education, and others.

Although many pedagogical patterns such as "Spiral" use steps (or fragments) in the learning process, they focus on the repetition or additional use to enforce learning. This requires extra time and delays our learning with critical time constraints.

The lack of sufficient background knowledge brings a new insight to our loop development. Our practice for non-CS majors who haven't obtained systematic training of computer programming provides a pedagogical solution for this complex programming education problem in a short time scale. Patterns such as "Consistent Metaphor", "Lay of the Land", and "Larger than Life" help us to determine the use of real research projects in the introduction to loop program in class. "Tool Box" helps us to decide the use of the Wii-game like wireless equipment, which motivates the students at the beginning and verifies the outcomes at the end. "Toy Box" and "Fixer Upper" motivate us to create the "one error to correct" software tool, in order to help students learn the proposed development template and the corresponding disciplined steps (educational sub-tasks).

3. CONCLUSION

Our goal is to make a condensed training practical for inexperienced trainees. As a result, our computer training such as the one for loop development has achieved a good success. In this paper, we have shown the efficiency and effectiveness of our education model to organize the learning in and after class through its three phases. The students are motivated to follow a disciplined procedure and learn to handle the complex aspects of a routine. As an abstract module, our approach can satisfy a variety of educational purposes. The Computer Science Department of WCU has adopted our approach to attract more non-CS majors to enroll in the CSC115 course and to develop their computer skills. Similar training has been planned at the the inter-departmental and interdisciplinary level, in order to repeat the success of our CSC115 education approach.

The specific problem in teaching trainees who have little experience is of intrinsic interest because of its economic importance and potential market value. It is clear that general education, certificate programs, and minor programs are used in many places. These programs vary in language, application and teaching method. However, our search did not yield any complete comparison. Based on the discussion above, it is clear that learning programming with our proposed education model works better with practice. More importantly, by reaping the benefits of using patterns, our approach can evolve to a better teaching model for computer education.

Acknowledgments

This work was supported in part by NSF REU grant CCF 0936942. We would like to thank Dr. James Fabrey and Dr. Anthony Nicastro for supporting the proposed training at WCU and Ms. Ayesha Begum for her assistance in the software development. We especially thank our shepherd, David West, who provided valuable comments which considerably improved this paper. Last but not least, we would like to thank the writers' workshop at PLoP'11 and the attendees for providing insightful suggestions.

4. REFERENCES

[1] ACM/IEEE-CS Computer Science Curricula 2013. Information is available at http://ai.stanford.edu/users/sahami/CS2013/strawman-draft/cs2013-strawman.pdf

[2] ACM International Collegiate Programming Contest. Information is available at http://en.wikipedia.org/wiki /ACM_International_Collegiate_Programming_Contest.

[3] An accurate measurement of infection on mice with wireless Imote2 sensor equipment, supported by the CAS dean's office at WCU. Information is available at http://www.cs.wcupa.edu/∼zjiang/RA_Spring11.htm.

[4] J. Bergin. *Pedagogical Pattern*. Information is available at http://csis.pace.edu/∼bergin/#pedpat.

[5] CS121−204, undergradate programming courses, Comptuer Science Department, Drexel University. Information is available at http://www.drexel.edu/catalog/ug/coe/cs-index.htm#.

[6] CIS101-Introduction to Computer Science: Principles of Information and Computation, Department of Computer and Information Science, University of Pennsylvania. Information is available at http://www.cis.upenn.edu/ugrad/all-courses.shtml.

[7] CIS1001−68, programming courses for all science majors, College of Science and Technology, Temple University. Information is available at http://www.temple.edu/bulletin/ugradbulletin/ ucd/ucd_cis.html.

[8] CISC101-Computers and Information Systems, University of Delaware. Information is available at http://primus.nss.udel.edu/CourseDesc/ info.action?searchKey=2011%7CCISC101.

[9] CISC103-Introduction to Computer Science with Web Applications, University of Delaware. Information is available at http://primus.nss.udel.edu/CourseDesc/ info.action?searchKey=2011%7CCISC103.

[10] CISC106-General Computer Science for Engineers, University of Delaware. Information is available at http://primus.nss.udel.edu/CourseDesc/ info.action?searchKey=2011%7CCISC106.

[11] CMPSC097−397, undergradate programming courses, Penn State University. Information is available at http://bulletins.psu.edu/bulletins/bluebook/university _course_descriptions.cfm?letter=C&dept=CMPSC

[12] CSC110 & CSC115, general education courses for non-CS majors, Computer Science Department, West Chester University. Information is available at http://www.wcupa.edu/_INFORMATION/OFFICIAL. DOCUMENTS/Undergrad.Catalog/compsci.htm.

[13] CSE15−17, programming courses for CS major and minor, Computer Science and Engineering Department, Lehigh University. Information is available at http://www3.lehigh.edu/academics/catalog/html/ index.html.

[14] iMote2. Information is available at https://docs.tinyos.net/index.php/Imote2.

[15] Loop. Information is available at https://en.wikipedia.org/wiki/Control_flow.

[16] Wii. Information is available at https://us.wii.com/.

[17] National security telecommunications and informationa system security (NSTISS). *National Training Standard for Information Systems Security*, June 1994. Document is also available at http://www.cnss.gov/Assets/pdf/nstissi_4011.pdf.

[18] A. Chua. *Battle Hymn of the Tiger Mother*, 2011. Penguin Press HC.

[19] F. Buschmann. *Pattern-Oriented Software Architecture*, 1996. Wiley.

[20] S. Frizell and R. Hubscher. Aligning Theory and Web-based Instructional Design Practice with Design Patterns. *Proceedings of E-Learning 2002: World Conference on E-Learning in Corporate, Government, HealthCare, & Higher Education.* pp. 298-304, 2002.

[21] T. Gaddis. *Starting out with Java: From control Structures through Objects*, 2008. Pearson.

[22] W. Groesbeck and S. Delaney. Program correctness. Document is available at http://www.cse.unr.edu/∼bebis/CS365/ StudentPresentations/ProgramCorrectness.ppt.

[23] Z. Jiang. CSC115-Introducation to Computer Programming, special session of Matlab for non-CS majors, Computer Science Department, West Chester University. Information is available at http://www.cs.wcupa.edu/zjiang/matlabindex.htm.

[24] D. Jones and S. Sharonn and L. Power. Patterns: using proven experience to develop online learning. *Proceedings of ASCILITE'99.* pp. 155-162, 1999.

[25] J. Mache and N. Bulusu and D. Tyman. Making sensor networks accessible to undergraduates through activity-based laboratory materials. *Proc. of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'08).* pp. 606-608, San Francisco, CA, June 16- 20, 2008.

[26] Y. Mor. Design for learning: a pattern approach. presented in *the Workshop of the Chairs Conference on Instructional Technolgies Research: "Learning in the Technolgical Era".* 2007. Document is also available at http://lp.noe-kaleidoscope.org/outcomes/chairs.