Designing Engineering Courses for Design

Authors:

James Jackson, United States Military Academy, West Point, NY, james.jackson@usma.edu
Michael Cobb, United States Military Academy, West Point, NY, michael.cobb@usma.edu
Karl Gossett, United States Military Academy, West Point, NY, karl.gossett@usma.edu
Curtis Carver, United States Military Academy, West Point, NY, curtis.carver@usma.edu

Abstract *¾ Over the last 18 months the Electrical Engineering and Computer Science Department at West Point migrated its mandatory freshman introduction to engineering/information technology course to a design-centric, problem solving course. This course emphasizes the use of a standard, iterative problem solving process: analyze, design, implement, and test. Course feedback from previous semesters indicates that students do not see the value of designing a solution prior to implementation. Our students struggle with two options. The first is to jump right in and try various implementations until they succeed (which is at odds with the course material and many times leads to non-working implementations). The second is to take the time to develop a coherent design and the corresponding implementation. To combat the practice of implementation before design, we refocused our course using a series of pedagogical measures. We broke down the course into loosely coupled modules. Each lesson within a module displays strong cohesion with the other lessons in the module. We ensured that each module integrated design principles. Next, we developed a set of pre-class learning objectives and a set of in-class learning objectives using Bloom's Taxonomy of Learning. The pre-class objectives are knowledge and comprehension based and lower on Bloom's scale and support the higher application and synthesis based in-class objectives. Our instructors do not directly cover any pre-class objectives in class. The in-class objectives emphasize design with a hands-on, in-class exercise during most lessons. We also integrated a design-only project that forces the students to think through the design of a moderately complex problem. This design-only project compels the students to use all of the common design principles presented in the course to develop a plan to solve the problem. We integrated a graded design in every project during the semester. Lastly, the final course project is of such complexity that students are unlikely to succeed without a well-conceived design. Student course feedback from last semester has demonstrated an overwhelming success. Students understand the criticality of design before implementation.*

Index Terms *¾ design, engineering course, pedagogy, problem solving*

BACKGROUND

The United States Military Academy at West Point, New York, was the first four-year engineering college in the United States. Throughout most of its 200 year history West Point has produced only graduates of engineering disciplines. In the mid-1980s other disciplines were offered; however, each student still takes a minimum five course engineering sequence. The first course in this sequence is the freshman introduction to engineering and information technology course. Every freshman takes this course regardless of motivation or aptitude. This course introduces freshman to a methodical problem solving process and fundamental information technology concepts. The freshmen at the military academy are subjected to many stressors such as academics, acculturation into the military lifestyle, physical fitness requirements, and a rigorous ethical environment. They are continually on the look out for techniques that save them time or they *perceive* will save them time. The freshmen each have an identical laptop computer that communicates wirelessly within the classroom to the campus network. Classroom size is limited to 18 students. Each instructor typically has four sections of 18 students.

PROBLEM

Our problem solving process is a traditional four-step waterfall model with modification to provide feedback to previous steps. The four steps are: analyze the problem, design a solution, implement the solution and test the solution. The analysis step has a standardized format for listing the problem specifications. The design step typically involves the creation of one or more flowcharts. The implementation step typically involves the creation of a Java program. The testing step involves creating and using a test plan to ensure the implementation meets the problem specifications. We emphasize that errors detected earlier in the process are easier to fix and have smaller ramifications that errors detected later in the process.

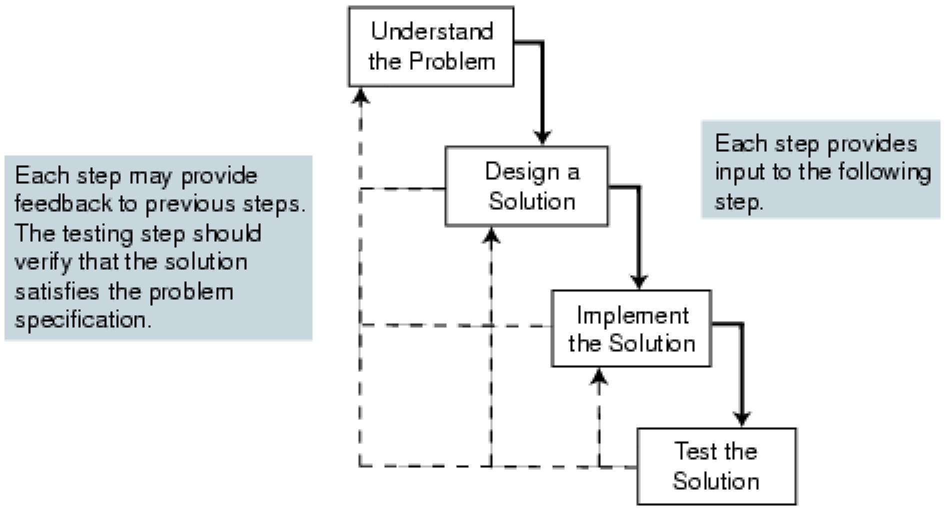

Understand the Problem
Design a Solution
Implement the Solution
Test the Solution
Each step may provide feedback to previous steps. The testing step should verify that the solution satisfies the problem specification.
Each step provides input to the following step.

FIGURE 1
PROBLEM SOLVING PROCESS

Our students tend to view the outcome of the process, typically a Java program, as opposed to the process itself, as the educational goal of their homework assignments. Although no civil engineer would consider building a bridge before designing it, rapidly building and testing softare implementations is feasible. Because we require products for each step to be submitted by the students for each homework project, we see most students working "backwards." They like to develop a Java program, then create the design, then the test plan and lastly the problem specification document. Our term for students that use this technique is *designing at the keyboard*. They tend to struggle simultaneously with design issues, syntax issues and logic issues which raise the level of student frustration and reduce their confidence in their ability to master the material.

An underlying objective of the course is to change the way students think about solving problems. We would like to have students spend their time thinking through and developing a good design instead of developing an implementation. The ideal design would allow a student to translate the flowchart into source code with minimal efffort.

SOLUTION

We have implemented a series of techniques that demonstrate the importance of design, in general, and *before* implementation. Our techniques are application of traditional pedagogical concepts applied to our particular circumstances.

BLOOM'S TAXONOMY

Starting at the course level we developed over-arching course-wide learning objectives. When evaluating the selection of material for individual lessons, we used the course objectives as a guide to whether or not material should be included. A graduate of this course must have achieved a satisfactory in each objective.

- Apply the Problem Solving Process to solve problems in a systematic manner.
- Understand the fundamental terms, limitations, capabilities, hardware and software associated with Information Technology.
- Develop XHTML and Java programs as implementations of automated solutions.
- Master common software applications in support of everyday tasks.

Benjamin Bloom in the mid-1950's proposed a cognitive learning model that can be applied to course design to improve student learning. His model has six levels of understanding loosely viewed as knowledge, comprehension, application, analysis, synthesis and evaluation [1]. Using this taxonomy we created a set of pre-class and in-class learning objectives. Our philosophy is that we must meet our students where they are, and that they are, especially during the fall semester, transitioning from a level of rigor associated with high school academics to a much higher level of academic rigor associated with an undergraduate education. The pre-class learning objectives help our students identify important concepts from the reading. These objectives mostly fall within the knowledge or comprehension levels of the taxonomy. The pre-class learning objectives are meant to give the students a context within which to understand the more difficult material. They are aligned with a set of in-class learning objectives. The in-class learning objectives build on the pre-class learning objectives and focus mainly at levels three, four and five of Bloom's Taxonomy. The critical aspect of this system is that each of the twelve instructors teaching the course ensures that the students realize that it is their responsibility to understand the pre-class learning objectives and the instructors do not, as a general rule, cover the pre-class learning objectives in class. We do not use our in-class time to review the material of the previous night's reading, but rather to expand and build from the reading. For example, our lesson on the design step of the problem solving process has the following three pre-class objectives:

- Describe the *Design the Solution* step of the Problem Solving Process.
- Describe a flowchart.
- Compare and contrast flowcharts and other design techniques.

There is one in-class learning objective aligned with these pre-class objectives.

- Develop a design for a solution given a problem statement and a problem specification.

Our philosophy toward the actual lessons is that we spend as much time as possible engaging the students in a hands-on event. The designs that they create in the course include web map diagrams, flowcharts, and physical network design diagrams.

A MODULE-BASED COURSE

To ensure that we cover design throughout the semester, we have grouped the major topics into modules. The nature of the course means that we cover dissimilar topics that loosely fall under engineering and information technology. The major modules in our course follow with the design products that we use in each module:

- Problem Solving (flowcharts)
- XHTML (web site map, layout designs using cascading style sheets)
- Logical Design and Control in Java (flowcharts)
- Sensors and Computers (printed circuit board schematics)
- Networks (network design diagrams)
- Information Assurance (personnel and network security design measures)

Throughout course design we made it an explicit goal to include producing a design-related product in each module. Many of the modules include many design products.

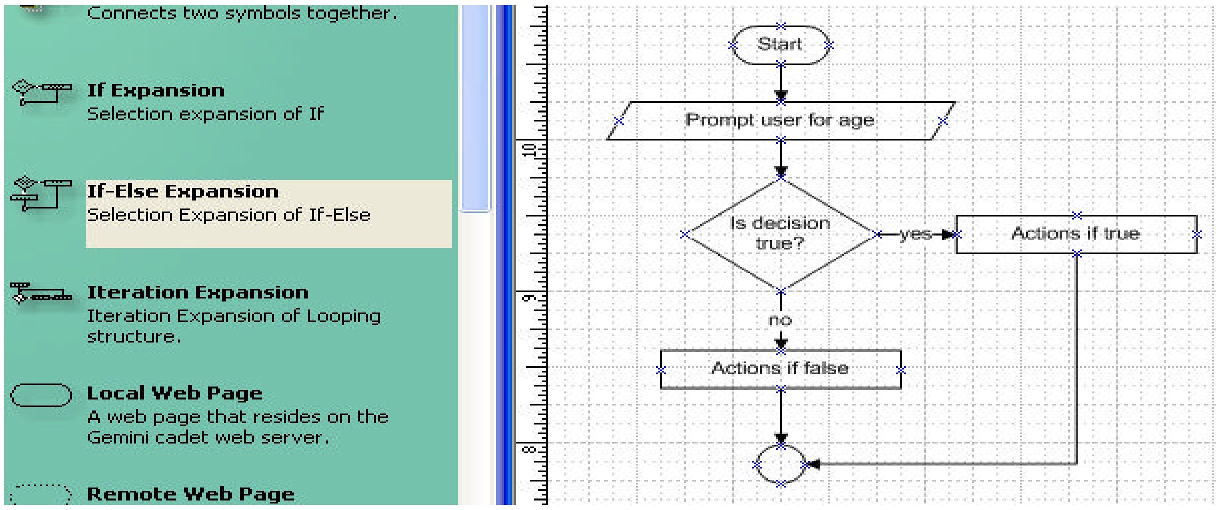
PROJECTS

Our students complete five projects over the course of the semester that take anywhere between 8 and 40 hours (for a team of two) to complete. Each of our projects has a design step that requires the students to create a design product prior to the implementation. Four of the five projects require flowcharts and the last requires a web map diagram. The very first project that they complete is a *design-only* project. They develop a problem specification, a flowchart and a test plan, but no implementation. This technique has proven very useful in that it forces our students to focus on design without having to worry about the implementation.

Each of the student projects weights the implementation no more than 40 percent of the overall grade. The design is typically weighted between 25 and 40 percent of the overall project. In previous semesters we weighted the project heavily toward the implementation and received projects that had very good implementations but fairly poor designs, indicating that the students had worked backwards to arrive at the design.

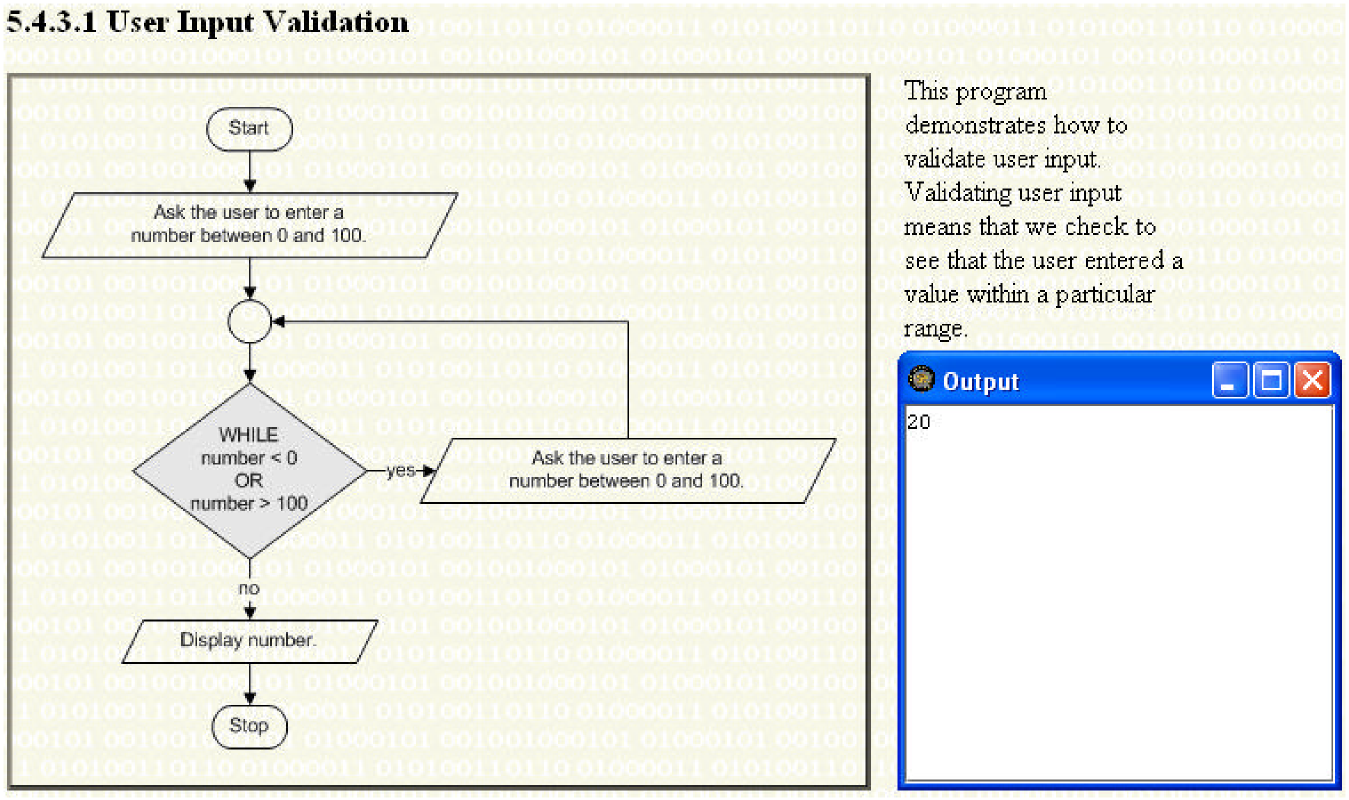
AUTOMATED TOOLS

Our students have a homogenous automation environment that we leverage when teaching our design lessons. Visio, from Microsoft, is the course standard flowcharting software. Each student laptop comes with this software pre-installed. The use of a standardized stencil with Visio has many positive impacts. By standardizing conventions right in the tool, we reduce instructor grading time. The stencil reduces common errors in flowcharts by providing standard starting points for common logical control structures like selection and iteration. We annotate selection (i.e., an *if-else* statement) on a flowchart through the use of a diamond. By convention, we always point the true branch of the condition to the right and the false branch downward. Prior to using the standard starting point expansion symbols students would attempt to add three or more arrows from a conditional symbol, not label arrows either yes or no and forget arrows or lines connecting symbols. Forcing students to use a standard stencil has greatly reduced these issues. Figure 2 demonstrates that the if-else expansion stencil icon expands into four symbols with notes as to what should go there. Students can then type right over the top of the text in each symbol, add and delete symbols and manipulate the arrows much faster than trying to draw the symbols themselves every time.



Connects two symbols together.
If Expansion
Selection expansion of If
If-Else Expansion
Selection Expansion of If-Else
Iteration Expansion
Iteration Expansion of Looping structure.
Local Web Page
A web page that resides on the Gemini cadet web server.
Remote Web Page
Start
Prompt user for age
Is decision true?
yes
Actions if true
no
Actions if false

FIGURE 2
EXAMPLE EXPANSION ICON FROM FLOWCHARTING STENCIL



5.4.3.1 User Input Validation
Start
Ask the user to enter a number between 0 and 100.
WHILE number < 0 OR number > 100
yes
Ask the user to enter a number between 0 and 100.
no
Display number.
Stop
Output

This program demonstrates how to validate user input. Validating user input means that we check to see that the user entered a value within a particular range.

```
20
```

FIGURE 3
FLOWCHARTS LINKED TO EXECUTABLE PROGRAMS

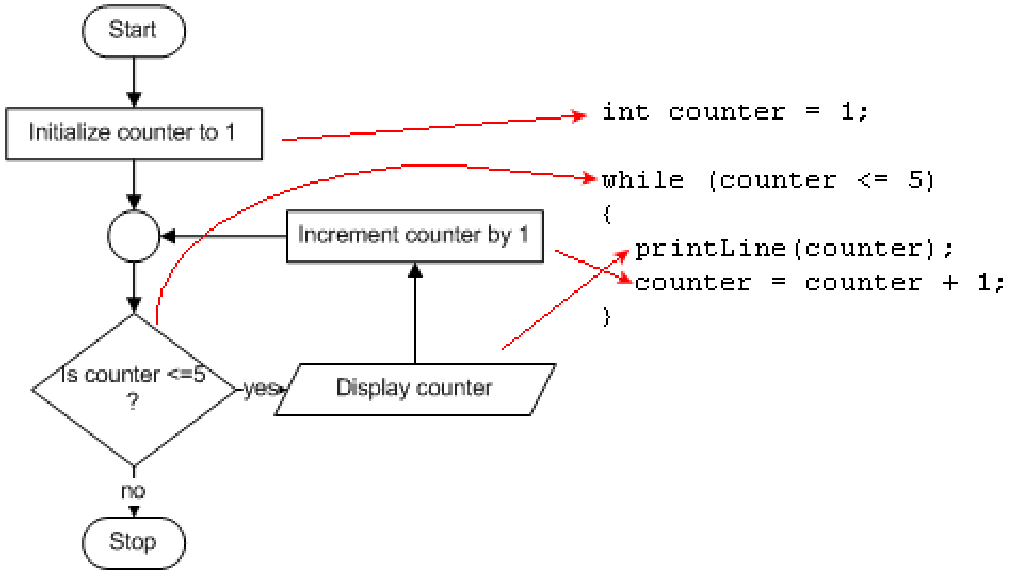
DYNAMIC ONLINE HANDBOOK

Approximately 80 percent of the course readings are consolidated in one online handbook. The benefit of the online handbook is that we can achieve effects that aren't possible with a paper handbook. The current version of the handbook allows flowcharts to be linked to the source code of the implementation, the problem specification, or a web-enabled application that runs the source code. We found that some students that fall into the global learner category comprehend complex flowcharts faster if they are given the chance to see and use the actual implementation of the flowchart during execution. Generally, we won't show them the actual source code until we get to the implementation portion of the course.

Another technique used in the handbook is similar to the old-style cartoons that would have a bouncing ball move along the top of words as a song was playing. We have an applet that highlights each step of a flowchart as it would be executed and displays either comments explaining the current step or the corresponding line of source code.

FLOWCHART TO CODE COMPARISON

We stress throughout the lectures and in the online handbook that the translation from a flowchart to the source code should be relatively mindless. We show our students numerous examples of the translation of flowchart to source code.

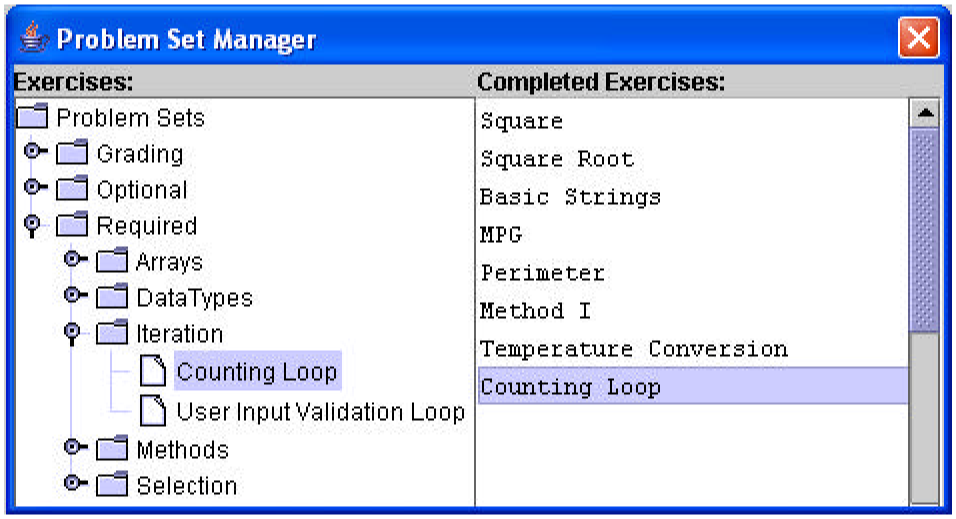


```
int counter = 1;
while (counter <= 5)
{
   printLine(counter);
   counter = counter + 1;
}
```

FIGURE 4
FLOWCHART CODE TRANSLATION

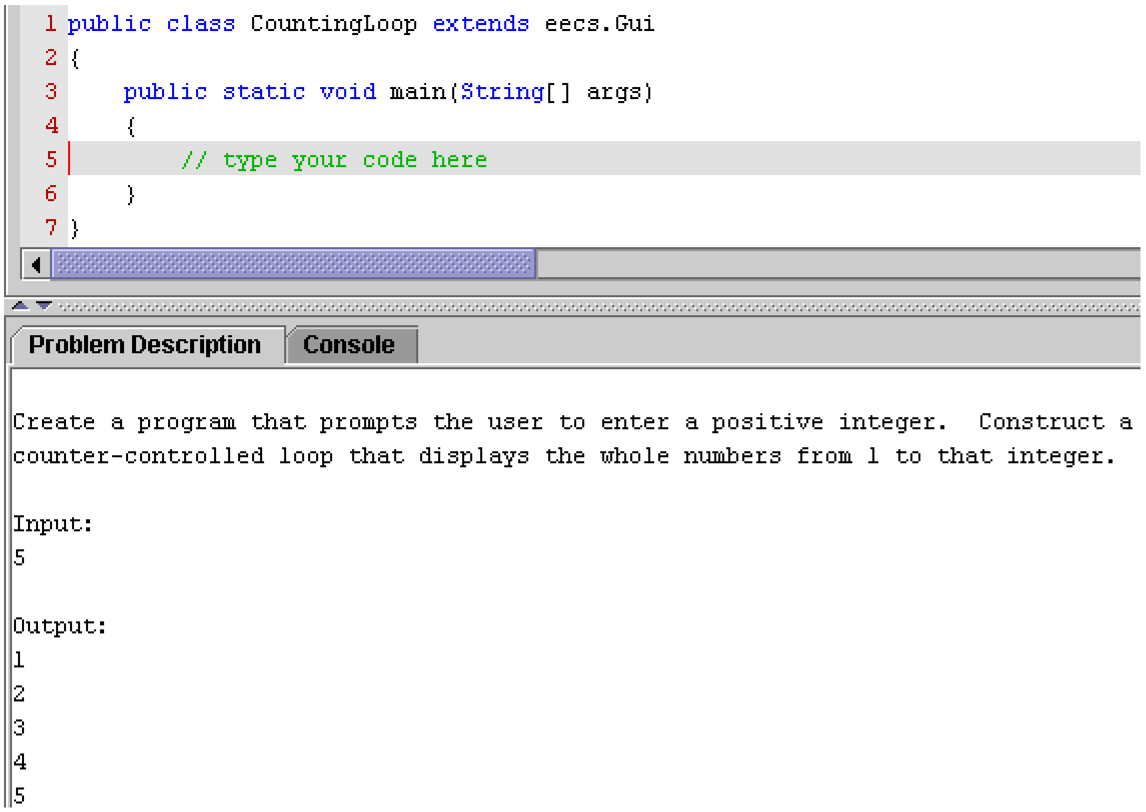
PROBLEM SETS

Our students use an Integrated Development Environment (IDE) that was built and is maintained by our department. This IDE supports a feature that allows us to assign small implementation assignments that require the students to develop a program of 10-20 lines of code. The IDE retrieves the problem set from a repository on the internal local area network. The problem has a description and automatically opens up a flowchart that corresponds to the solution. The cadets are required to take the problem statement, problem specification, and flowchart and develop the small program. The IDE then allows the student to run a battery of automated tests to see if their implementation is correct. The problem sets allow the faculty to move beyond basic syntax in the classroom and reinforce the use of developing a flowchart even for small problems.



```
Square
Square Root
Basic Strings
MPG
Perimeter
Method I
Temperature Conversion
Counting Loop
```

FIGURE 5
PROBLEM SET MANAGER



```
public class CountingLoop extends eecs.Gui
{
    public static void main(String[] args)
    {
        // type your code here
    }
}
```

```
Create a program that prompts the user to enter a positive integer.  Construct a
counter-controlled loop that displays the whole numbers from 1 to that integer.

Input:
5

Output:
1
2
3
4
5
```

FIGURE 6
PROBLEM DESCRIPTION

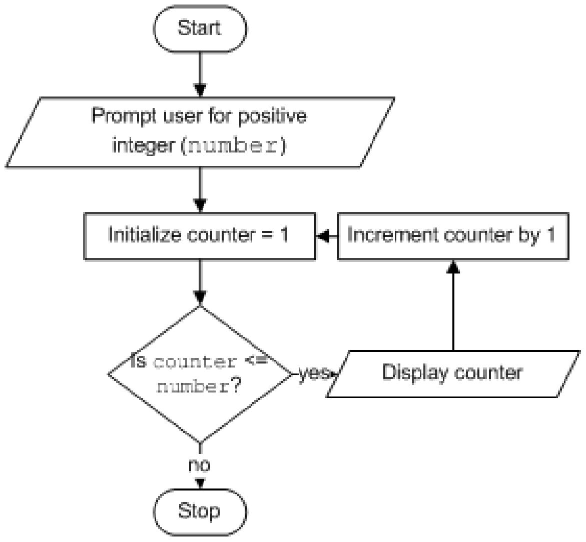
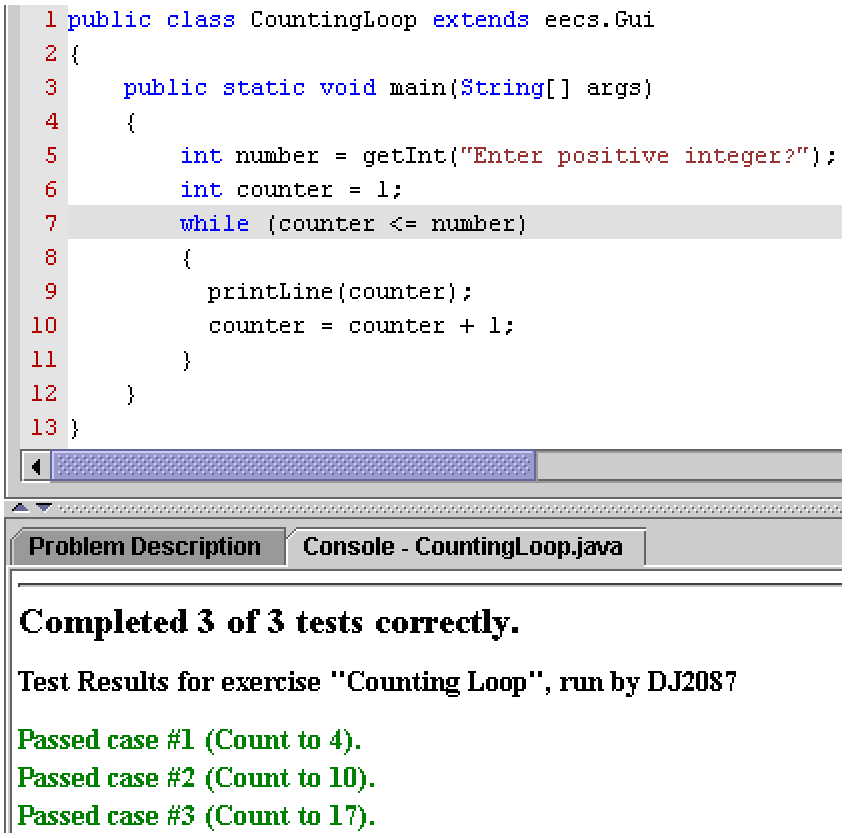


FIGURE 7
FLOWCHART THAT AUTOMATICALLY OPENS WITH PROBLEM DESCRIPTION



```
public class CountingLoop extends eecs.Gui
{
    public static void main(String[] args)
    {
        int number = getInt("Enter positive integer?");
        int counter = 1;
        while (counter <= number)
        {
          printLine(counter);
          counter = counter + 1;
        }
    }
}
```

Completed 3 of 3 tests correctly.

Test Results for exercise "Counting Loop", run by DJ2087

Passed case #1 (Count to 4).
Passed case #2 (Count to 10).
Passed case #3 (Count to 17).

FIGURE 8
AUTOMATIC TEST CASES

OFFICE HOURS REQUIRE USE OF FLOWCHART

Since our course has between 12 and 18 instructors in any given semester, it is imperative that all students receive comparable levels of instruction. Since we do not use graduate student teaching assistants, our office hours tend to be about three to ten hours a week. When conducting office hours, we have a convention that helps reinforce the importance of the design process to our students. We do not provide any help to students who have not worked through a rough draft of their problem specification. Students who come to get help on implementation problems are turned away if they do not have a corresponding design or they are given assistance only on their design. This may sound callous; however, we have found that if we make our expectations explicit on the course web site and reinforce them throughout the semester that most of our students will come to office hours with the a rough draft of the required product.

CONCLUSIONS

As instructors we have seen the course both before and after the design-centric changes were made. We can say that our students are improving with their understanding of how a good design can greatly reduce the time required to produce a working solution. One instructor has estimated that by focusing on design we have reduced office hours by 20%. A survey of instructors over one year indicated that students who developed a good design up front generally encountered 15-40% fewer errors over the project. Another instructor who has taught hundreds of students indicated that "the cost in spending additional lessons on design is more than made up by the reduction of office hours later in the semester."

The most interesting feedback comes from our students. When presented with the statement *I have learned critical thinking processes in this course*, 93% of students responded *agree* or *strongly agree* in an anonymous end-of-course survey. Here is a sampling of student feedback when asked what they would remember most about the course.

- Without a doubt, the problem-solving process. I almost think its use was stressed too much exclusively for programming because it applies to any situation where a solution must be designed and implemented for a particular problem. I intend to apply it to many situations with which I am faced.
- I will remember the problem solving process. I may not remember the exact parts of code, but this class helped me think in a different manner.
- The problem solving process
- How to attack a problem
- This class was one of the better classes in the aspect of challenging me to think creatively and solve problems.
- I will remember how to write programs using java code because I spent so much time on coding.
- How to organize and plan out a soultion [sic] to a problem.
- I will remember the problem solving process and the method we used to approach each problem in a logical manner.

REFERENCES

[1] Bloom, Benjamin S., Taxonomy of Educational Objectives: The Cognitive Domain, Vol 1, 1956