IPRO 328

Spring 2008

Testing and Improving a New Text for Teaching Computer Science

Result of IPRO Deliverables

Project Plan	Grade: 14/14
Midterm Report	Grade: 16/16
Code of Ethics	Grade: 16/16

Work Results

Book Results

Revised Chapters	7
Completely New Chapters	4
Total Chapters	11

Book Changes

Insertions	497
Deletions	518
Total Changes	1015

Lecture Slide Results

Chapters w/ Slides	5
Chapters w/o Slides	5
Total Slides	152

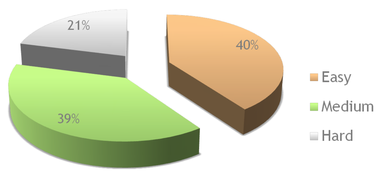
Exercise Results

total time spent (min)	5285
average time spent (min / prob)	18
average grade (out of 5)	4.15

Exercise Changes

Changed	23
Added	35
Deleted	14

Exercise Difficulty



Chapter Comments Score Board

Team Member	Major	Chapters	Average/Chapter
Allen, David	Non-CS	10	12.2
Bathum, Nicholas	CS	8	4.8
Jeong, Seon	Non-CS	10	7.9
Johnson, Leland	CS	10	4.3
Kwak, Noh Hyup	Non-CS	10	8.5
Patel, Vivek	Non-CS	10	6.8
Schmitz, Peter	CS	8	4.1
Tilatti, Michael	Non-CS	10	11.0
Hammes, Katherine	Non-CS	2	5.0
Kofman, Roman	CS	2	5.0
Tran, Harry	Non-CS	2	6.5
Rymek, Phil	CS	2	5.5

Code of Ethics

Overarching Standard

All book-contributors will:

- **Fulfill all requirements set forth by the client**
- **Submit only original work**
- **Never lose sight of the best interest of the consumers**
- **Never act unfairly toward fellow members**

Paradigm of Code of Ethics

- **The book uses multicultural examples and exercises such as driving or making sandwiches.**
- **All of our content is original and created with our audience in mind.**

Meeting Structure

1. Two consecutive class hour meeting
2. Thursday IRC online chat
3. Wednesday Editing Subteam meeting

Project Plan

IPRO Deliverable	Start	Finish
Project Plan	1/25/2008	2/22/2008
Midterm Report	3/3/2008	3/14/2008
Midterm Presentation	3/3/2008	3/14/2008
Final Report	4/21/2008	5/2/2008
IPRO Day Presentation	4/20/2008	5/25/2008
IPRO Day Poster	4/21/2008	4/25/2008
Code of Ethics	2/15/2008	3/7/2008
Website	4/18/2008	4/25/2008
Abstract	4/21/2008	4/25/2008
Meeting Minutes	1/25/2008	4/18/2008
CD-ROM	4/25/2008	5/2/2008
Team Debriefing	5/5/2008	5/26/2008

Milestones	Start	Finish
Chapters 1 - 4 and Midterm report	2/4/2008	3/16/2008
Chapters 5 - 9	3/14/2008	4/28/2008
Final Report and Deliverables	3/14/2008	5/2/2008

Time Spent

Name/Week	1/28-2/3	2/4-2/10	2/11-2/17	2/18-2/24	2/25-3/2	3/3-3/9	3/10-3/16	3/17-3/23	3/24-3/30	3/31-4/6	4/7-4/13	4/14-4/20	total
David Allen	7.0	6.0	6.8	9.6	13.5	12.3	3.0	5.5	6.0	7.5	9.0	7.0	93.2
Nicholas Bathum	7.3	3.5	7.0	11.5	10.0	10.0	7.0	1.0	6.0	3.0	8.0	6.0	80.3
Katherine Hammes	3.5	12.0	9.0	11.0	10.0	10.0	5.0	10.0	12.0	9.5	11.0	9.0	112.0
Seon Jeong	2.5	10.5	9.0	9.0	10.0	10.0	9.0	2.0	11.0	6.0	7.0	10.0	96.0
Leland Johnson	5.0	8.5	8.3	10.0	10.0	6.0	6.5	6.5	6.0	6.5	7.0	4.0	84.3
Roman Kofman	7.1	6.0	3.0	6.0	8.5	12.0	7.0	0.0	8.0	8.0	8.5	7.8	81.9
Noh Hyup Kwak	3.5	7.5	8.5	11.0	8.5	12.0	7.0	3.5	6.5	10.0	9.0	9.0	96.0
Vivek Patel	3.5	9.8	9.5	6.5	10.0	10.0	7.5	0.0	6.5	6.0	9.0	11.5	89.8
Phillip Rymek	5.0	10.0	10.0	10.0	8.0	9.0	8.0	0.0	9.0	8.5	9.0	10.0	96.5
Peter Schmitz	5.9	6.3	4.0	4.8	8.0	8.8	5.0	3.0	4.0	3.5	5.0	14.0	72.3
Michael Tilatti	5.0	10.0	6.5	6.0	10.0	10.5	6.0	0.0	9.0	6.0	9.0	8.0	86.0
Harry Tran	2.6	10.0	8.0	10.0	10.0	10.0	10.0	10.0	10.0	11.0	11.0	10.0	112.6
week total	57.9	100.1	89.6	105.4	116.5	120.6	81.0	41.5	94.0	85.5	102.5	106.3	1100.9
week average	4.8	8.3	7.5	8.8	9.7	10.1	6.8	3.5	7.8	7.1	8.5	8.9	91.7

IPRO 328

Spring 2008

Testing and Improving a New Text for Teaching Computer Science

IPRO 328 Members

David Charles Allen, Nicholas Bathum, Katherine Hammes, Seon Jeong, Leland Johnson, Roman Kofman, Noh Hyup Kwak, Vivek Patel, Phillip Rymek, Peter Schmitz, Michael Tilatti, Harry Tran, Yacin Nadji (Student Advisor), David Grossman (Advisor)

Editing Team 1: Katherine Hammes, Roman Kofman, Phillip Rymek, Harry Tran

- Created new chapters 1 through 8
- Re-edited chapters 1 through 8 based upon others' comments
- Commented on chapters 9 and 10

Editing Team 2: Nicholas Bathum, Peter Schmitz

- Created new chapters 9 and 10
- Edited chapters 9 and 10 based upon others' comments
- Commented on chapters 1 through 8
- Initially helped the exercise team

Exercise Team: David Allen, Nicholas Bathum, Seon Jeong, Noh Kwak, Vivek Patel, Peter Schmitz, Michael Tilatti

- Responsible for learning Ruby and all concepts presented in the book.
- Completed all homeworks to test proficiency of the book.
- Commented on every chapter of the new book.

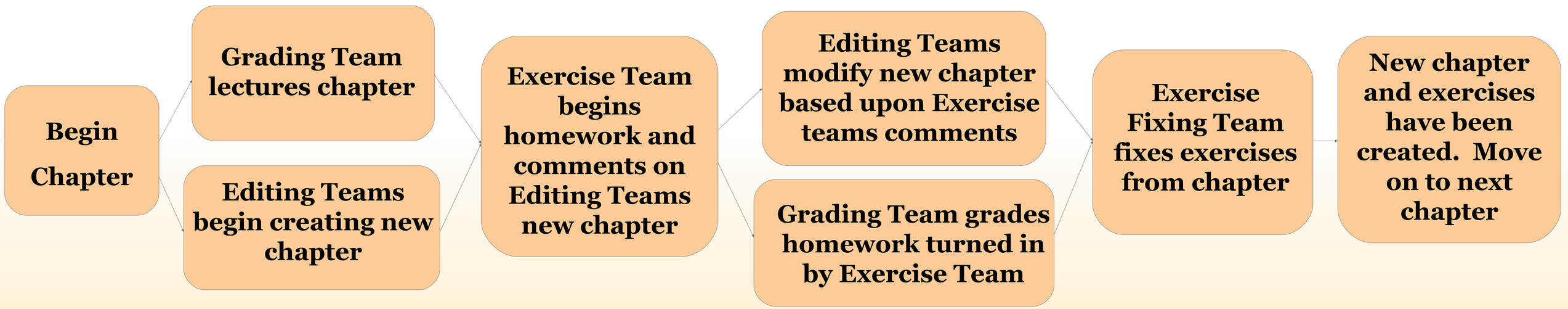
Grading Team: Leland Johnson, Phillip Rymek

- Created lecture slides
- Taught the book in a lecture format
- Graded all the homeworks
- Commented on every chapter

Exercise Fixing Team: David Allen, Seon Jeong, Noh Kwak, Vivek Patel, Michael Tilatti

- Modified, deleted, and created problems for every chapter of the book.
- Populated new chapters with problems
- Commented on every chapter

Development Cycle



IPRO 328

Spring 2008

Testing and Improving a New Text for Teaching Computer Science

Background

Most Traditional Computer Science Texts are:

- **Long and Imposing**
- **Out of touch with student needs**
- **Out of touch with industry**

Problem

Create an effective text:

- **Assess and improve the text**
  - **Create teaching tools**
  - **Create new problem sets**

Objectives

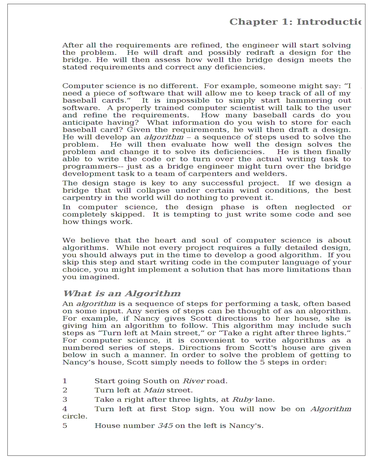
- **Text itself**
  - **Revise or rewrite all 8 Chapters**
- **Teaching Tools**
  - **Create and test slides**
- **Problem Sets**
  - **Test and revise for all 8 Chapters**

Results

- **One entirely new book**
- **A lot of edits based upon real results**
- **An almost complete set of lecture slides**
- **Problems rearranged, edited, and recreated based on student feedback**

Phase One (Create original book draft)

- **Last semester's IPRO was given a bare bones draft of a Ruby textbook provided by Prof. Frieder & Prof. Grossman**


Chapter 1: Introductio

After all the requirements are refined, the engineer will start solving the problem. He will draft and possibly redraft a design for the bridge. He will then assess how well the bridge design meets the stated requirements and correct any deficiencies.

Computer science is no different. For example, someone might say: "I need a piece of software that will allow me to keep track of all of my baseball cards." It is impossible to simply start hammering out software. A properly trained computer scientist will talk to the user and refine the requirements. How many baseball cards do you anticipate having? What information do you wish to store for each baseball card? Given the requirements, he will then draft a design. He will develop an *algorithm* – a sequence of steps used to solve the problem. He will then evaluate how well the design solves the problem and change it to solve its deficiencies. He is then finally able to write the code or to turn over the actual writing task to programmers– just as a bridge engineer might turn over the bridge development task to a team of carpenters and welders.

The design stage is key to any successful project. If we design a bridge that will collapse under certain wind conditions, the best carpentry in the world will do nothing to prevent it.

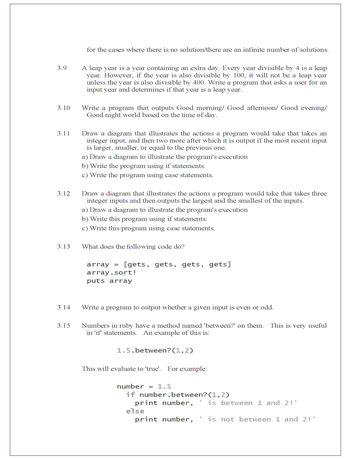
In computer science, the design phase is often neglected or completely skipped. It is tempting to just write some code and see how things work.

We believe that the heart and soul of computer science is about algorithms. While not every project requires a fully detailed design, you should always put in the time to develop a good algorithm. If you skip this step and start writing code in the computer language of your choice, you might implement a solution that has more limitations than you imagined.

What is an Algorithm

An *algorithm* is a sequence of steps for performing a task, often based on some input. Any series of steps can be thought of as an algorithm. For example, if Nancy gives Scott directions to her house, she is giving him an algorithm to follow. This algorithm may include such steps as "Turn left at Main street," or "Take a right after three lights." For computer science, it is convenient to write algorithms as a numbered series of steps. Directions from Scott's house are given below in such a manner. In order to solve the problem of getting to Nancy's house, Scott simply needs to follow the 5 steps in order:

1 Start going South on *River* road.
2 Turn left at *Main* street.
3 Take a right after three lights, at *Ruby* lane.
4 Turn left at first Stop sign. You will now be on *Algorithm* circle.
5 House number *345* on the left is Nancy's.


for the cases where there is no solution/there are an infinite number of solutions.

3.9 A leap year is a year containing an extra day. Every year divisible by 4 is a leap year. However, if the year is also divisible by 100, it will not be a leap year unless the year is also divisible by 400. Write a program that asks a user for an input year and determines if that year is a leap year.

3.10 Write a program that outputs Good morning/ Good afternoon/ Good evening/ Good night world based on the time of day.

3.11 Draw a diagram that illustrates the actions a program would take that takes an integer input, and then two more after which it is output if the most recent input is larger, smaller, or equal to the previous one.
a) Draw a diagram to illustrate the program's execution
b) Write the program using if statements.
c) Write the program using case statements.

3.12 Draw a diagram that illustrates the actions a program would take that takes three integer inputs and then outputs the largest and the smallest of the inputs.
a) Draw a diagram to illustrate the program's execution
b) Write this program using if statements.
c) Write this program using case statements.

3.13 What does the following code do?

```
array = [gets, gets, gets, gets]
array.sort!
puts array
```

3.14 Write a program to output whether a given input is even or odd.

3.15 Numbers in ruby have a method named "between?" on them. This is very useful in 'if' statements. An example of this is:

```
1.5.between?(1,2)
```

This will evaluate to 'true'. For example:

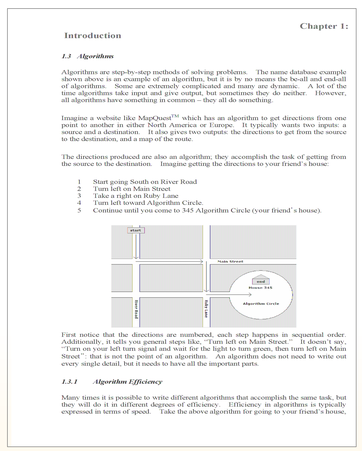
```
number = 1.5
  if number.between?(1,2)
    print number, " is between 1 and 2!"
  else
    print number, " is not between 1 and 2!"
```

- **They accomplished the following tasks:**
  - **Improved & edited the text itself**
  - **Developed 20~30 exercises per chapter and solutions for every exercise.**
  - **Developed 5-6 programming examples**
  - **Creating figures & accompaniments**
  - **Created two Model Eliciting Activities**

- **The book was not perfected and needed a large scale revision phase, Phase 2 – this semester of IPRO 328**

Phase Two (Test and Revise)

- **Returning members from last years IPRO taught the class the contents of the book.**
- **Homework was assigned to class and graded by whoever lectured that week.**
- **Grades and class input were tracked using Google Docs and was used to drive changes to the book.**


Chapter 1:

Introduction

1.1 Algorithms

Algorithms are step-by-step methods of solving problems. The name database example shown above is an example of an algorithm, but it is by no means the be-all and end-all of algorithms. Some are extremely complicated and many are dynamic. A lot of the time algorithms take input and give output, but sometimes they do neither. However, all algorithms have something in common – they all do something.

Imagine a website like MapQuest™ which has an algorithm to get directions from one point to another in either North America or Europe. It typically wants two inputs: a source and a destination. It also gives two outputs: the directions to get from the source to the destination, and a map of the route.

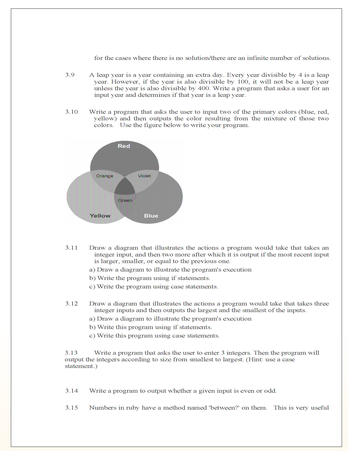
The directions produced are also an algorithm; they accomplish the task of getting from the source to the destination. Imagine getting the directions to your friend's house:

1 Start going South on River Road
2 Turn left on Main Street
3 Take a right on Ruby Lane
4 Turn left toward Algorithm Circle.
5 Continue until you come to 345 Algorithm Circle (your friend's house).

First notice that the directions are numbered, each step happens in sequential order. Additionally, it tells you general steps like, "Turn left on Main Street." It doesn't say, "Turn on your left turn signal and wait for the light to turn green, then turn left on Main Street": that is not the point of an algorithm. An algorithm does not need to write out every single detail, but it needs to have all the important parts.

1.3.1 Algorithm Efficiency

Many times it is possible to write different algorithms that accomplish the same task, but they will do it in different degrees of efficiency. Efficiency in algorithms is typically expressed in terms of speed. Take the above algorithm for going to your friend's house,


for the cases where there is no solution/there are an infinite number of solutions.

3.9 A leap year is a year containing an extra day. Every year divisible by 4 is a leap year. However, if the year is also divisible by 100, it will not be a leap year unless the year is also divisible by 400. Write a program that asks a user for an input year and determines if that year is a leap year.

3.10 Write a program that asks the user to input two of the primary colors (blue, red, yellow) and then outputs the color resulting from the mixture of those two colors. Use the figure below to write your program.

3.11 Draw a diagram that illustrates the actions a program would take that takes an integer input, and then two more after which it is output if the most recent input is larger, smaller, or equal to the previous one.
a) Draw a diagram to illustrate the program's execution
b) Write the program using if statements.
c) Write the program using case statements.

3.12 Draw a diagram that illustrates the actions a program would take that takes three integer inputs and then outputs the largest and the smallest of the inputs.
a) Draw a diagram to illustrate the program's execution
b) Write this program using if statements.
c) Write this program using case statements.

3.13 Write a program that asks the user to enter 3 integers. Then the program will output the integers according to size from smallest to largest. (Hint: use a case statement.)

3.14 Write a program to output whether a given input is even or odd.

3.15 Numbers in ruby have a method named "between?" on them. This is very useful

- **This innovative method allowed us to accomplish the following:**
  - **We completely revised 7 of the chapters and created three entirely new chapters.**
  - **8 of the 10 chapters have lecture slides, and they have all been successfully tested.**
  - **Problems from every chapter were revised and many new problems created.**