FLORIDA STATE UNIVERSITY

COP 3331

Object-Oriented Analysis and Design

STUDY GUIDE

Spring 2003

http://campus.fsu.edu

Table of Contents

Introduction

COP3331: OBJECT-ORIENTED ANALYSIS AND DESIGN

Course Director: Ian Douglas

Instructor: David A. Gaitros, GAITROSD@CS.FSU.EDU , (850) 644-4055

Welcome to Florida State University's *COP 3331: Object-Oriented Analysis and Design*. This course assumes that you have taken *COP 3330: Object-Oriented Programming* and an introductory course in Computer Science. In particular, we assume that you have already been introduced to an object-oriented programming language and understand the concepts of *encapsulation*, *inheritance*, and *polymorphism*.

Developing a proficiency in programming is a basic skill requirement for both computer scientists and software engineers; however, programming skills alone are insufficient to make you a successful software engineer. For large-scale problems, an understanding of and experience in systems analysis and design are also required. In this course you will begin to develop such knowledge and skill, and increase your familiarity with the principles of software engineering. You will also gain practical skills in using Unified Modeling Language (UML).

Help with FSU Procedures and Technology

A complete learning community and support group will help answer your questions and remedy any problems, from enrolling in courses and developing an academic plan to accessing your course website and submitting assignments via the Web.

For Help with University or Course-Related Problems

Your Learning Community includes the following help:

- See your *academic coordinator* for questions and concerns about:
    - academic requirements of the University and your department or school
    - procedural matters such as course equivalency substitutions, course prerequisites, or graduation checks
- See your *course mentor/instructor* for questions and concerns about:
    - course content
    - your progress in the course

Course-specific questions you might ask a mentor or instructor:

- "Can you help me understand the difference in meaning between the terms *protocols* and *standards* in the Week 9 assignment? They sound like the same thing."
- "What can we do about one of the other students in our group project who isn't doing anything? The rest of us can't continue until she finishes her part of the project."

General questions a student might ask a mentor:

- "When I go to take my proctored exam, will the proctor be able to answer questions I might have about the test?"
- "Will I be at a disadvantage because I live too far away from you to ever meet with you face-to-face?"

*See your **lead faculty** only when your mentor or instructor directs you to do so or when you have a problem that is not adequately addressed by your mentor or instructor.*

For Help with Technical Problems

If your problems or questions concern your computer equipment, software, Internet connection, or the course website, follow the procedures described in one of the following two boxes:

If you have a technical question and are able to connect to the Internet:

1. Check out the *Online Support* website at:

   http://www.fsu.edu/~webhelp/students/index.html

2. You may go directly to the *Student Handbook for Online Learning*, which provides step-by-step instructions on using the course website as well as technical contacts. It is available online at:

   http://www.fsu.edu/~distance/media/studenthandbook.pdf

3. If your question or problem is not addressed by the websites listed above, then send an **e-mail** to:

   oddl@inquiry.fsu.edu

If your problem concerns your course website, please include the following information in your email:

1. The area in the course website at which you are experiencing difficulties (e.g., Discussion Board, Virtual Chat room, or file upload)
2. A detailed description of the problem and exact transcriptions of any error messages
3. Your course, course prefix, section, and instructor's name
4. Your name, e-mail address, and a daytime phone number

And, if you can, include the following technical information:

5. The Web browser your are using (e.g., Netscape Navigator, Microsoft Internet Explorer, AOL)
6. The operating system you are using (i.e., Windows 95, 98, or NT; Mac; Linux; or Unix)
7. Whether you are connecting with a modem or within a network

You will receive a reply to your email on the next business day.

If you have a technical question and are *not* able to connect to the Internet:

Call the ACNS Help Desk at **1-850-644-8502**.

Syllabus for COP 3331

Course Number, Name, and Prerequisite

Course Number: COP 3331

Course Name: Object-Oriented Analysis and Design

Prerequisite: COP 3330 Object-Oriented Programming

Course Description

This course introduces you to software engineering and focuses on the important activities that **precede** the writing of computer code. It will build on many of the concepts and procedures you learned in the previous courses. In the previous courses, you were acquainted with the fundamental concepts of computer science, and began to develop skills in computer programming. You should have learned how to represent problems you encounter and to carefully consider approaches to solutions **before** beginning to write code.

Thus far, you have dealt with small, relatively well-defined problems. In the world of commerce and industry, technological issues and problems are much larger and more complex. The problems are well beyond the scope of a single individual. Software engineers, in collaboration with others, often spend much of their time brainstorming, discussing, and creating documentation of problems and problem solutions (often incorporating complex diagrams), and a relatively small amount of time writing actual code.

Analysis and *design* are two of the key activities involved in software development. These activities can be carried out using a number of different methods and tools. Many modern software development methods are based around the *object-oriented* approach. It is this approach that we will focus upon in this course, and it is assumed you are already familiar with the basic principles of object orientation.

It is important to recognize that there is no one right or wrong way to develop software. Two companies can organize the process of development very differently, yet both may produce successful systems. The overall efficiency of the development process is more critical to the success of the project than the particular method or tool that is used. In order to be efficient, software developers need to give as much consideration to their development process as they do to the product.

In this course, you will gain practical skill in analysis and design that will complement your skills in programming. You will learn about the problems of and approaches to developing large computer systems, and will acquire an understanding of what constitutes a development methodology. You will also experience using computer-aided software engineering (CASE) tools and learn an industry standard modeling language, Unified Modeling Language (UML).

This course is designed to complement the programming course *COP 3330: Object-Oriented Programming*, in which you have gained the programming experience needed to convert completed designs into computer executable code.

This course can be divided into four learning sections. In the first section (Weeks 1–3) you will focus on systems engineering. In the second section (Weeks 4–7) you will focus on analysis. (Week 9 is the midterm exam, and Week 10 is spring break.) In the third section (Weeks 8, 11–12) you will focus on design. In the final section (Weeks 13–14) you will focus on some issues in software development methodology, including quality control. Use Week 15 for exam preparation.

Overall Course Objectives

After you have completed this course, you should be able to:

1. Describe and explain the main stages, methods, tools, techniques, and problems involved in systems and software engineering.
2. Explain the main concepts and goals of analysis and design.
3. Demonstrate practical skill in the use of an analysis and design notation.
4. Demonstrate technical and communication skills required for working in systems development teams and dealing with stakeholders.*

Required Course Materials

You need all of the following to complete this course:

- Textbook
- Study guide
- Access to the course website
- CD-ROM
- UML diagramming software

Textbook:

Bruegge, B., & Dutoit, A. (2000). *Object-oriented software engineering*. NJ: Prentice Hall.

Companion website for this book: **http://cw.prenhall.com/bookbind/pubbooks/bruegge/**

*A *stakeholder* is someone who is not part of the development team but who has an interest in the success of the project.

Article:

Booch. G. (1999). *The software development team.* Available: **http://www.rational.com/products/whitepapers/100580.jsp** [2000, November 30]

(Also reprinted in the appendix to this study guide.)

Recommended Additional Reading:

This course makes use of one textbook and you are also referred to additional readings within this study guide. If you would like to broaden your knowledge, you are encouraged to consult other texts and Web-based sources.

- The textbook we are using looks at software engineering from an object-oriented perspective, but most software engineering texts will take a more general approach. For an example, see:

  Pressman, R. (2000). *Software engineering: A practitioner's approach*. New York: Prentice Hall.

  Sommerville, I. (1996). *Software engineering*. Harlow Essex, England: Addison-Wesley.

- If you wish to learn more about the general process of product development (not restricted to computer systems), consult:

  Reinertsen, D. G. (1997). *Managing the design factory*. New York: The Free Press.

- There are a number of relevant links provided on the course website. These are update regularly. Some of the main sites relating to software engineering and OOAD are:

  The Software Engineering Institute: **http://www.sei.cmu.edu/**

  The Method Engineering Encyclopedia: **http://panoramix.univ-paris1.fr/CRINFO/dmrg/MEE/**

  Rational Software—the market leader in OO computer-aided software engineering (CASE) tools: **http://www.rational.com/index.jsp**

- If you wish to go into more depth on the use of UML in analysis and design, the following are recommended:

  Booch, G., Rumbaugh, R., & Jacobson, I. (2000). *The complete UML training course*. Pearson Interactive.

  Larman, C. (1997). *Applying UML and patterns*. New York: Prentice Hall.

Course Website Features:

On the course website you will find a number of resources, including the following:

- Templates for the weekly assessments and project reports.
- Self-test quizzes
- Lecture presentations
- Examples
- Useful links
- Group working areas

CD-ROM:

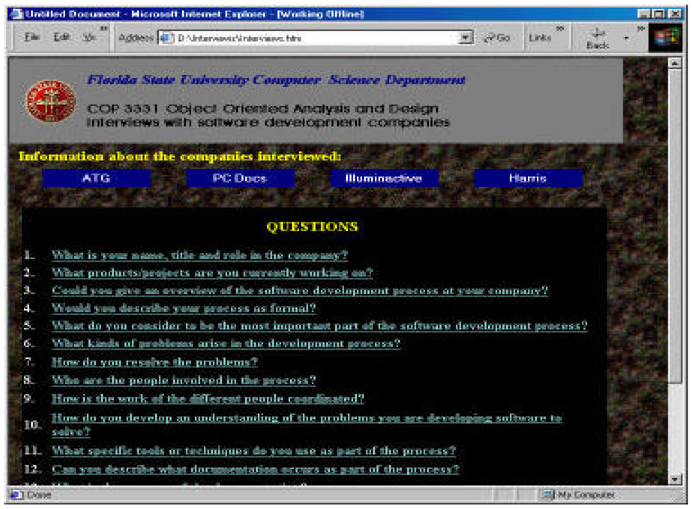
The Multimedia CD-ROM (hereafter referred to as the *CD-ROM*) contains material that will be used in this course. These include the following:

- A series of interviews with practicing software developers
- A student version of Rational Rose visual modeling tool
- The runtime version of Apple's QuickTime video player

While using this learning resource, keep the following in mind:

- Depending on the speed of your machine, there may be some delay when loading video clips.
- You will need to install the QuickTime software in order to play the videos. This software is available on the CD-ROM.
- It is recommended that you use Internet Explorer and a screen resolution of at least 800x600 while accessing the videos.
- The interviews will also be available at the course website in audio-only form.

The video interviews will be utilized in a number of the weekly assessments throughout the course. You can access the video files from your Web browser by selecting **Open** from the file menu and then selecting **interviews.html** from the CD-ROM. You can also click **My computer** on the desktop, select the CD-ROM drive, open the **Interview** folder, and then double-click the **interviews.html** file. Your computer screen should look like that shown on Figure 1.


Florida State University Computer Science Department

COP 3331 Object Oriented Analysis and Design
Interviews with software development companies

Information about the companies interviewed:

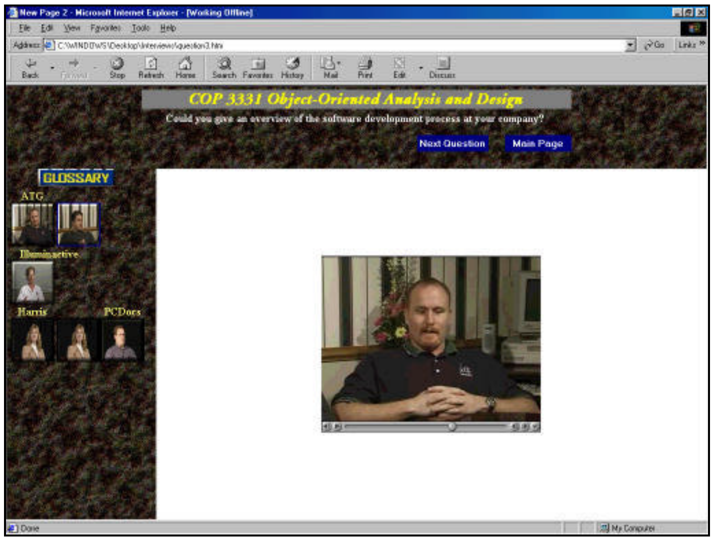
ATG | PC Docs | Illuminactive | Harris

QUESTIONS

1. What is your name, title and role in the company?
2. What products/projects are you currently working on?
3. Could you give an overview of the software development process at your company?
4. Would you describe your process as formal?
5. What do you consider to be the most important part of the software development process?
6. What kinds of problems arise in the development process?
7. How do you resolve the problems?
8. Who are the people involved in the process?
9. How is the work of the different people coordinated?
10. How do you develop an understanding of the problems you are developing software to solve?
11. What specific tools or techniques do you use as part of the process?
12. Can you describe what documentation occurs as part of the process?

Figure 1: Opening screen for the interviews

Selecting one of the numbered questions will take you to a screen like the one shown on Figure 2.


COP 3331 Object-Oriented Analysis and Design

Could you give an overview of the software development process at your company?

Figure 2: Accessing the videos for each question

After you have selected a question, you will note that the next screen contains photographs of people from the various companies. By double-clicking on one of these pictures, you will be able to access a video of that person's answer to the question. A glossary defining some of the technical terms used in the answers may be available. There is also a link to the main page or to the next question in the sequence.

In order to play the videos, you will need a version of Apple's QuickTime installed on your machine. The latest version of this software can be obtained from

http://www.apple.com/quicktime/

You will find version 4.12 (most recent version as of 11/07/00) on your CD-ROM. Double-clicking on this icon will start the installation program.

Course Requirements

To successfully complete this course you must read assigned texts, listen to lectures, and participate in discussions. Completing the assigned readings before their corresponding lectures and discussions will enable you to raise relevant questions and to improve your learning experience and your grade.

Each week of the course you will be given a set of activities to complete. These activities usually involve taking self-tests from the textbook and consulting the website for other learning resources. Some weeks of the course will also provide "Additional Notes" and Web references to cover material that is not dealt with in the textbook.

Most weeks there are assessments to be completed. These are designed to enhance your learning and demonstrate your understanding of the material. Most weeks you will have to submit assessments to your mentor or TA for grading.

In the first half of the course (prior to the midterm exam), you will find that there are more assessments to complete. In the second half of the course you will be completing a course project. Consequently, there will be fewer assessments in the latter half, as you will be expected to spend time on the course project. You should anticipate an average of 10–15 hours for each week's activities and assessments.

Discussion: Active participation in discussions and group work is an important part of the course. Your contributions to the class will enrich your own experience and those of your classmates.

Assessment

The overall course grade is determined as follows:

Weekly Assessments	**35%**
Analysis and Design Project	**15%**
Midterm Exam	**20%**
Final Exam	**30%**

Weekly Assessments

Weekly assessments include assignments that ask you to demonstrate your command of the week's materials. Each assignment includes the total possible points you can earn and the grading criteria on which your submitted work will evaluated. The total number of obtainable points will vary from week to week. Each week, the points earned on the assignments in the assessments are totaled to give the weekly score. The 35% portion of your grade derived from weekly assessments will be determined by dividing the total assignment points earned by the total number of assignment points possible. The Week 13 assignment will not count towards the final score but will be treated as a substitution assignment; i.e., you can substitute the score on this assignment for your lowest score on the previous assessments. If you miss an assignment, this score can be substituted for the missing assignment.

Exams

The midterm and final exams will be proctored. You will be informed when and where to report for these exams. If the maximum obtainable score for an exam is not 100, it will be converted to a 100-point scale.

Analysis and Design Project

You will work in a group of three or four to produce a requirements analysis document for a new software system. Your grade will be assessed both on the quality of the report and the assessment of your effort by your peers and instructor.

Course Schedule

The following calendar delineates the assessments throughout the semester. It is your responsibility, however, to check the **Announcements** and the Course Calendar on the course website for the most up-to-date information and due dates of assessments.

Semester Calendar				
		Readings	**Individual Assessments**	**Group Assessments**
Week 1 Course Introduction		▪ Introductory Notes ▪ Textbook preface	▪ Familiarize yourself with the course website. ▪ Week 1 Assessment (14 pts)	None
Week 2 Systems Engineering		▪ Ch 1 ▪ Week 2 Notes	Week 2 Assessment (36 pts)	None
Week 3 Communication		▪ Ch 3	Week 3 Assessment (39 pts)	Workgroup assigned
Week 4 Modeling		▪ Ch 2	Week 4 Assessment (38 pts)	Group discussion task
Week 5 Requirements		▪ Ch 4 ▪ Week 5 Notes	Week 5 Assessment (41 pts)	Group task
Week 6 Analysis		▪ Ch 5	Week 6 Assessment (22 pts)	Group Project released
Week 7 Project Preparation		▪ Ch 11	Week 7 Assessment (15 pts)	Group Project organization and planning
Week 8 System Design		▪ Ch 6	Week 8 Assessment (15 pts)	None
Week 9 Midterm Preparation		▪ Review first 5 chs.	None	None
Week 10 Project Work/ Spring Break		▪ Read ahead	None	None
Week 11 Design Methods		▪ Extra readings provided	Week 11 Assessment (20 pts)	None
Week 12 Object Design		▪ Ch 7	Week 12 Assessment (35 pts)	None
Week 13 Quality Control		▪ Ch 9	Substitution Assessment (30 pts)	None
Week 14 Issues in Methodology		▪ Ch 12 ▪ Extra readings	None	Group Project due
Week 15 Revision/Exam Preparation		None	None	None

Course Policies

Assessments should be completed using the templates provided on the website. The assessments should be submitted before the due date through the DropBox at the course website using the standard file naming conventions.

Be sure to include the following information on all assessments that you submit:

- Student ID number
- Name
- Section number (for on-campus students) or mentor's name (for distance students)

You will receive assignment grades and feedback through individual email.

NOTE: Please keep a backup copy of all your work. We cannot assume responsibility for lost items.

Late Policy

Descriptions of each assignment and evaluation criteria will be provided through the course. The dates listed for assessments may change as the semester progresses. If they do, you'll be notified through the course bulletin board. However, we do expect assessments to be completed on time.

The policy regarding late assessments is 10% penalty for late assessments turned in within 24 hours of due date and 50% penalty for assessments turned in more than 24 hours after the due date. Assessments submitted later than four days after the due date will not be accepted. We use this system even in the event of excusable situations such as minor sicknesses or other unforeseen conflicts. However, any exceptions to this policy are made at our discretion.

Students who miss graded assessments receive a grade of zero. If you are ill or have a serious problem that prevents you from submitting an assignment on the day it is due, please contact your mentor or instructor at once and we will arrange an alternative date. This should be done at least 12 hours before the submission deadline.

The score for the last assignment will be treated as a substitution assignment, which can replace a missed assignment. Those who do not miss an assignment can use their score on the substitution assignment to replace their lowest score.

Grading

The grading scale for the course will be as follows:

92–100	**A**
90–91.99	**A-**
88–89.99	**B+**
82–87.99	**B**
80–81.99	**B-**
78–79.99	**C+**
72–77.99	**C**
70–71.99	**C-**
60–69.99	**D**
0–59.99	**F**

Example Grading:

Week	**Assessment Score**	**Max Score Attainable**
1	14	14
2	31	36
3	35	39
4	28	38
5	31	41
6	20	22
7	15	15
8	12	15
11	15	20
12	27	35
TOTAL	**228**	**275**
[13	28	30 **]**

- Substitute Assessment 13 for Assessment 8

New Total

246/275 = .8945 = **90%**

Note: All percentages are rounded up.

Midterm	37 out of 50	**74%**
Final	41 out of 50	**82%**
Project		**80%**

Overall Grade Calculation

Assessments	Project	Midterm Exam
90 x 0.35 = 31.5	80 x 0.15 = 12	74 x 0.2 = 15

Final Exam	**Total**
82 x 0.3 = 25	**84%**

Final Letter Grade: **B**

About This Study Guide

This study guide presents each week or unit of this class through the following headings and sections:

- **Overview**

  This section provides a summary of the major concepts or focus of study for the week, as well as a brief description of the assigned readings and activities.

- **Objectives**

  This section describes the knowledge and skills you should master.

- **Focus Questions**

  This section poses questions about the major issues and concepts related to the week's topic. These questions will help you organize and focus your readings.

- **Readings and Presentations**

  This section lists the readings and presentations that you should read or listen to before the week's end.

- **Activities**

  This section lists applications and practices that will help you achieve the week's objectives.

- **Assessments**

  This section lists any assignments for the week that will receive individual grades. You will complete the assignment with a written summation or response to be uploaded to the DropBox. (For detailed instructions on submitting a file to the DropBox, see "How do I use my Tools area" in the *Student Handbook for Online Learning*, available online at: **http://www.fsu.edu/~distance/media/studenthandbook.pdf**)

- **Checklist**

  This section gives you a checklist of the readings, presentations, activities, and assessments that you should have completed by the week's end.

Week 1: Introduction

Overview

Before we focus on the analysis and design stages of the systems engineering process, it is important to have a good understanding of the overall development process and the different ways in which it can be organized.

You will find that it is common to refer to *software engineering* rather than *systems engineering*. However, it is more correct to use the term *systems engineering* because most software is part of a larger system. As the course progresses, you will begin to see that if the software components of a system are engineered in isolation from and without specific consideration to other system components (e.g., hardware, human users), problems are very likely to arise.

In this first week, you will cover the Introductory Notes, familiarize yourself with the course website, and complete your first assessment. You will also have the opportunity to meet the lead faculty and fellow students through the course website. Please ensure that you are familiar with the procedure for submitting assessments and with the late policy.

Objectives

Upon completion of Week 1 you should be able to:

- Find your way around the various sections of the website
- Access the video interviews on the CD-ROM and describe the companies being interviewed
- Distinguish *software engineering* from *computer science* and *programming*
- Distinguish between *software engineering* and *systems engineering*
- Distinguish between *spiral* and *linear* processes
- Identify the main attributes of product design

Focus Questions

- Think about the ways in which you organize the development of a large software development process and then ask: What skills and tasks would be involved?
- When you make a purchase of a technology product (hardware or software), what attributes do you consider when deciding among competing products?

Readings and Presentations

- Read the Introductory Notes and Chapter 1 of the textbook.
- On the course website you will be given some web references to read.

Activities

- Write some notes relating to the bulleted points in the lecture slides.
- Familiarize yourself with the course website, textbook, and CD-ROM. While at the website do the following:
  1. Take the introductory self-test quiz.
  2. Set up a home page in the Student Tools area.
- Install QuickTime on your computer.
- Distance learning students who have not already done so should establish contact with your mentor.

Assessments

The process of developing software systems is complex, and there are many ways of doing it. Academics often differ in what they consider to be the best approach to process.

To give you a sense of the various perspectives on systems development, we have conducted a series of interviews with representatives of four software development companies. These interviews will enable you to hear from a variety of professionals involved in the development of software-based products and services.

Many of the assignments are designed to help you "actively listen" when you watch the video interviews. *Active listening* is the ability to take note of what is said and to consider and compare it with information received from other sources. Active listening is an important skill required of a systems analyst.

The interviews can be found on the CD-ROM, and also can be accessed in audio-only form through a link on the course website.

Submit responses to the assignments below by the due date in the course calendar. The point value of portion of the assignment is indicated in the parentheses.

Assignment 1.1 [Total of 14 points available]

Your first task is to conduct some preliminary research on the four companies taking part in the video interviews. Visit the websites of the four companies. The links can be found on the course website and on the main page of the interviews. Note the kinds of software products each company produces. Then listen to the responses to the following interview questions on the CD-ROM:

- "What are your name, title, and role in the company?"
- "How would you describe the products/projects you are currently working on?"

In your own words and using jargon-free language, write a paragraph that briefly summarizes pertinent information about **two** of these companies. Each summary should be 60–120 words. You should have read the Introductory Notes prior to beginning this assignment. Please note that it is important that you stick to the word limits. An important skill for analysts is to be able to concisely summarize the information available to them. The word limit requires that you consider what is important information and what can be left out.

(7 points per company)

Grading criteria:

- Is each company's primary business clearly stated?
- Are the products of each company identified as either custom or off-the-shelf software?
- Are acronyms defined and technical phrases explained? For example, "The company's main business is software for DTP," should be "The company's main business is desktop publishing (DTP). DTP involves the production of high-quality documents on a computer.
- Are the summaries original expressions, rather than extracts from the company's website?
- Are the abstracts within the word limit?

Checklist

- ☐ Have you read the Introductory Notes?
- ☐ Have you read Chapter 1 in the textbook?
- ☐ Have you read the web references provided on the course website?
- ☐ Have you visited the course website?
- ☐ Have you written notes on the main points in the lecture slides?
- ☐ Have you installed QuickTime on your computer?
- ☐ Have you completed Assessment 1.1?
- ☐ Have you completed the introductory test?
- ☐ Have you met this week's learning objectives?

Week 1: Introductory Notes

Process and Product: An Introduction to Systems Engineering

"In most organizations product development is a mix of black art and esoteric ritual. Most development processes have not been consciously designed, but rather evolve along a bumpy path…like the old joke about making sausages. You may enjoy the results, but you really don't want to know how they do it."

From *Managing the Design Factory*, by Donald G. Reinertsen

"The scientist adds to the store of verified, systematized knowledge of the physical world; the engineer brings this knowledge to bear on practical problems. "

Encyclopedia Britannica

In this course you will look at the process issues in developing large-scale software systems. One approach to this subject is to think of software as a manufactured product. One of the defining features of human civilization is the production of manufactured products, for example, metal tools, wooden boats, etc. Humans have been developing products for thousand of years; however, starting with the Industrial Revolution towards the end of the 18th century, products began to get very complex. Consequently, the process of developing products has become much more dependent on organization and planning.

One of the most significant manufactured products to appear in the latter part of the 20th century is the computer. There are two unique aspects about the computer that distinguish it from all those products previously developed. It acts as a tool to augment our brainpower, and it is an adaptable tool that can be programmed to perform a variety of tasks. In a sense it is many products in one. Recently, the power of this tool has been enhanced by the ability to connect together computers through a worldwide communications network (the Internet).

People tend to think of a computer in the form of a PC (a monitor, box, keyboard, and mouse) and to think of software as what they see on this machine (e.g., Windows). Actually, computers have many different physical forms and most of the computers used in everyday life are embedded in other products, from cars to video recorders to children's toys. However, it is not the computer itself that is useful; what turns the computer into a useful product is software.

Software in many varied forms controls many aspects of our lives. Software controls the power stations that deliver power to our homes, and software is used to calculate the utility bills. It controls the satellites that enable international communications, and it enables a telephone to retrieve a previously dialed number.

Software engineering is concerned with the process of software development from the time that someone determines that there is a need for a new or better software product, to the time that product is no longer used. As a discipline, software engineering seeks to improve the development process in order to increase the efficiency of development and the quality of the product. Software engineering is very much related to computer science, which supplies the theory and technology that software engineers will apply. It also borrows knowledge from other areas, including traditional engineering subjects.

Systems

Although you will find that many people refer to *software engineering*, it is perhaps more accurate to refer to *systems engineering*. The reason for this is that software is seldom developed in isolation; rather, software is usually developed in conjunction with the accompanying hardware. In developing software there is also a need to develop training and support materials, including manuals and online help features. The overall package constitutes a *system*.

It is important to recognize that software will often be used as part of a system for conducting business. Software may utilize data that is collected and processed by another system, which may be other independent pieces of software, or a system involving human operators. Computer systems seldom involve a single isolated program; systems are usually comprised of a collection of programs or program components that work together. The overall success of a system is dependent on how well the different subsystems and components interact.

Problems can arise if software developers concentrate too much on software alone and neglect the many other areas that are part of a system. Exclusively focusing on software may lead to decisions that will have adverse effects on other parts of the system being constructed. This course will emphasize a team approach to systems development that requires everyone involved in systems development to communicate and cooperate. Any changes that software developers make in software design need to be clearly and promptly communicated to all members of the systems development team, because those changes may impact other areas of responsibility.

Systems operate within a particular environment, but are often affected by actions or conditions out of the immediate control of the systems developers. For example, a financial system may have to account for tax rates, yet tax rates can be altered at any time by governments. A well-designed system, therefore, will allow for easy updating of tax code changes.

To more fully understand the process of developing systems, we must distinguish between the overall system development and the development of the subsystems and the components that make up the subsystems. In most computer courses, much of a student's time will be spent leaning how to code at the subsystem or component level. This narrow focus can make it difficult to appreciate the "big picture" problems that occur at the systems level.

The "big picture" of systems development begins with analysis and proceeds to architectural design. Architectural design involves identifying the subsystems and setting up the control, coordination and communication of the subsystem developers. Building architects will coordinate and control the work of plumbers, bricklayers, electricians, etc., to realize their vision for a building as specified in the architectural plan. Systems architects will coordinate and control specialists in hardware, networking, software engineering, testing, documentation, etc., to realize their design vision of the system.

For each subsystem of the architecture, a separate analysis and design will have to be completed. Subsystems are usually not independent; hence, the need for someone to be responsible for coordinating the system development.

Components and Objects

Modern computer systems are seldom made up of one large program. The current trend is to construct large programs from collections of smaller programs, known as *components*. If you look in the systems directory of the Microsoft Windows operating system, you will find many files with the extension DLL (an acronym for *dynamic link libraries*). These are components used by the Windows operating system to perform various functions. A program that is built from smaller components usually produces a program that is easier to maintain. When a problem arises it is easier to identify the component responsible and replace it, rather than updating a whole operating system. Existing components can also be reused in new programs; new components have to be created only to handle new functionality.

There are now a number of different technologies, such as Microsoft's Active X or Java Applets, which software developers can use to create component-based applications. Although the idea of both components and object orientation share similar goals (i.e., facilitating reuse and ease-of-maintenance), these two concepts are not the same. Many components that are used in software systems may be constructed with traditional rather than object-oriented programming. The modern trend, however, is definitely towards object-oriented components.

Software as a Product

As noted in the introduction, software is a product. All products have attributes and developing a new product requires that trade-offs be made among different attributes. The aesthetic value (or visual appeal) of a product is one attribute; cost of manufacturing is another. All other things being equal, a leather-covered chair will look better than a plastic-covered chair. However, leather costs more to manufacture than plastic, and therefore a trade-off has to be made between aesthetic appeal and the manufacturing cost.

In the early days of software, developers did not often have to consider the relative value of different attributes and choose among them. However, many users of software now expect their software to have aesthetic appeal, particularly corporate websites. Thus many software projects have their cost of production raised due to the need to employ specialists in the visual design of software interfaces. In fact, if we look at industrial history this change has also happened for other products. Henry Ford created the first mass-manufactured automobile (the model T) and is quoted as saying, "You can have any color you want as long as it is black." Any modern manufacturer of autos would not be able to survive with this attitude towards customers' color preference. The look of a product can have a great effect on how easy it is to market.

In addition to aesthetics, other product attributes developers need to consider include the following:

Functionality
Functionality is what you can do with a product. Obviously, it is the most important attribute. However, care should be taken that the focus on functionality does not take up all the attention of the developers. Most modern software applications increase functionality with each new version or release of a product. As an example, in the early 1980s no word processor had a spell or grammar checker. Now this feature is fairly common. Functions that are at first innovative and unique, and that people are willing to pay more for, will become, with time, standard on products (e.g., color on TV sets).

Efficiency
Some products have similar functionality, but the way in which the functionality is implemented has an effect on the efficiency. For example, all C++ compilers perform the same function of compiling source code into machine code, but some compilers are much faster than others.

Usability
Usability is a measure of how easy a product is to learn and use. Again, different products can have similar functionality, but interface design can vary greatly and make a big difference on how the product is to operate for the user. A remote control for a TV is an example of a product for which usability can vary greatly according to the design. This attribute is also very important in software development, as customers have come to realize the training and support costs involved in purchasing computer systems. There are additional hidden costs associated with poor interface design and low usability when users make errors because they misunderstand the system they are using.

Reliability
Reliability is the level of continued product functionality. To avoid flaws in the product design, extra effort and cost need to be expended at both the design and testing level of development to ensure product reliability. It is no use for a product to have a high level of functionality, if that functionality is seldom available due to product breakdown.

Maintainability
A product's maintainability is determined by how easily the product can be updated, serviced, or repaired. One of the advantages to both component and object-oriented approaches to software is that they will result in a more maintainable system. If something is not right about a particular function, there are identifiable components and classes related to that function that can be repaired or modified, instead of having to search through a large program to find out which lines of code are responsible.

Safety
Safety is very much related to the reliability attribute, and its importance may be critical for certain applications. Products can be designed to ensure maximum safety, and for some products, there are laws to enforce safety standards. A user may be able to tolerate an occasional breakdown in the reliability of a computer game, but for safety reasons anything less than 100% reliability in flight control software cannot be tolerated.

Cost
This attribute is important for the profitability of the product. The greater the attention that is paid to the other factors, the greater the cost of product design and production. The greater the cost, the fewer the people who can afford to buy the product.

The Importance of Attributes

Designing a successful product requires that the development team consider all of these attributes. Trade-offs among attributes will inevitably have to be made; what these trade-offs are is often dependent on the target customer. There are cost-oriented customers who will accept lower quality in some attributes, and there are quality-oriented customers who are willing to pay more for having a high level of quality in all attributes.

The importance of different attributes often shifts as consumers become more informed. We have already noted that aesthetics has become more important in modern software design. Since the advent of the personal computer, which led to the mass use of computers in developed economies, the attribute of usability has become increasingly significant. Many software developers used to stress in their advertising what you could do with the product; now they stress ease of use. As more people were required to use computers in their jobs, the cost of training and supporting these new users began to increase. These economic pressures helped highlight the importance of usability, because the best way to reduce training and support costs is to have software that is easy to learn and use.

If we were to attempt to rate any product on the basis of the attributes listed above, it would be difficult to find a product that scored highly on all attributes. This is due to compromises that must be made in order to produce a product that will both satisfy a need and make a profit. It is impractical to design a highly efficient, functional, and aesthetically pleasing product that too few people will buy owing to its high cost. Product design teams must balance all of these factors and considerations. This is not always easy, as different members of the development team will have different perspectives on what are the most important attributes.

The importance of any given attribute will depend on the product and its purpose. In the example given at the start of this section, a leather chair is appropriate for use as home furniture, and plastic chairs are preferable for use in a school canteen. In the school dining room, where many chairs have to be supplied and the chairs' users are not particularly careful with the product, durability and cost are the most important attributes. In the home, aesthetics and comfort take precedence. The same kind of give-and-take between attributes is also true for software products. If the software is being designed for computer systems specialists, functionality and efficiency will be much more important than usability or aesthetics. Some experienced computer users have a dislike for graphical user interfaces, which they believe get in the way of the functionality. Conversely, a blank screen, flashing cursor, and cryptic error messages create an unnecessary barrier to novice computer users.

Two Categories of Product

There are two main categories of product, each of which has a slightly different design process.

Commercial Off-the-Shelf (COTS) Products
These products are designed and manufactured to a set standard and are usually purchased through a retail outlet. If you go into a software store, you will see many software products that you can take "off the shelf" and purchase. With these kinds of products, someone first perceives a need, and then works on inventing a product to satisfy that need. No money is made until the product ships (is purchased by retail outlets to be offered for sale), and often considerable investment will have to be made prior to shipping. Once a new product type has been developed, other companies will often begin to develop competing products, and will look for ways to attract more customers by improving the product design, such as adding more functions or improving the usability. Off-the-shelf products are sometimes referred to as *shrink-wrapped* products.

Custom Products
While off-the-shelf products serve the need of a general market, an organization often has unique requirements that cannot be met by off-the-shelf products. In such cases it is necessary for a custom product to be developed. For custom products, developers may receive some up-front or initial payment, but the developers have to carefully balance what they promise the customer against what is charged. A currently popular custom software product is the development of corporate websites. Although there are off-the-shelf products to assist in constructing websites, most websites must be custom-constructed to the exact specifications of the user. For a large interactive site, this will involve a significant amount of effort. Another example of a custom product is software that is designed for

integration into another product, such as a video recorder. While VCRs themselves are off-the-shelf products, the software contained in the VCR had to be custom-developed for that particular model.

Be aware that there is some overlap to these categories. Off-the-shelf software products often have features that can be customized to individual requirements, and customized products often incorporate off-the-shelf products as part of the system. For example, a customized website may utilize an off-the-shelf database product to store information accessed through that website.

Approaches to Process

This introduction has intended to get you to think about software as part of a system that constitutes a product to serve some human need. You will now begin to look at the process by which this product is produced.

Much of computer science tends to be concerned with computer technology and how it is implemented. Systems and software engineering is concerned with the development process, from the initial idea for a new software product to its delivery and beyond. Questions that systems and software engineering ask include: How is this process made efficient? How can we ensure that the end product will be of the highest quality? How can we ensure the product is needed and will be used?

The way software is developed varies from organization to organization. Some organizations have formalized methods for developing software that involve distinct stages, clear division of responsibilities, and well-specified documentation requirements. Other companies have informal development methods, with no clearly definable stages, small teams of people who cover more than one role, and little documentation. There are a number of factors influencing where on the continuum a company will appear, but basically, smaller companies are more likely to have a more informal process. This generalization should be qualified, however, because the specific application for the software will also influence the level of formality in the development process. If a company is creating safety-critical systems, such as for aircraft or nuclear power stations, it is very likely that the company will have a highly formalized process.

In addition to differences in the level of formality in a method, there are also differences in the flow of the process. The process of software development can be broken down into identifiable stages that take place. The main stages are:

Analysis:	Where there is an attempt to understand the problem.
Design:	Where a system is designed to solve the problem.
Implementation:	Where the system design is converted into a real product.
Quality assurance:	Where there is scrutiny of the output of various stages to ensure that a quality product is likely to result. This will incorporate some form of testing.
Installation:	Where the completed product is installed into the working environment.
Maintenance:	Where the product is upgraded and corrected after installation.

All these stages can be further subdivided into activities; for example, *analysis* can be broken down into requirements elicitation, user analysis, business analysis, etc. Different methods will involve different stages, and sometimes use different terms to refer to the various stages.

A process can flow in a linear manner, where one stage of development follows another. This is characterized by the traditional waterfall method. Another approach is to use an iterative method, where the stages are repeated over a number of cycles, the output at the end of each cycle moving closer towards being a final product. This is also referred to as a *spiral* process, which is a more appropriate description, as a spiral suggests that with each loop you get closer towards an end point.

It is also possible to have hybrid processes. For example, an iterative development is used to construct a prototype. The prototype is thoroughly analyzed and tested to determine the requirements for the final product. In the final product development, a linear process is used.

Prototyping involves constructing a working model of the final product to illustrate how it will look and operate. Many companies find this a useful way of removing the ambiguity that surrounds spoken and written descriptions of a design. Rapid development tools such as Visual Basic have increased the ease with which accurate prototypes can be constructed. Sometimes a prototype is used to facilitate understanding and to compare design options, but is then thrown away prior to the development of the final product (*throwaway* prototyping). Sometimes a prototype is incrementally developed to the extent that it becomes the final product (*incremental* prototyping).

Terminology

Many disciplines generate a lot of special terminology (more negatively referred to as *jargon*). This terminology can help experts to refer to complex concepts and processes, thus allowing for more precise and efficient communication. However, such terminology can also cause confusion, especially to those unfamiliar with a subject. Software engineering is no exception, and you will find that a particular term can have different meanings depending on the context in which the word is used, or the expert who is using it.

We have already clarified the difference in terminology between *software* and *systems* engineering, and looked at the term *component*.

In the above section, “Approaches to Process,” the word *method* was used to describe a particular process of software development. Many people use the term *methodology* instead of *method* to refer to the process of development. Strictly speaking, the suffix “ology” denotes the study or science of the root word (*psychology* is the study of the “psyche,” or mind). Thus, methodology more correctly describes the academic study of methods of software development, rather than a specific development process. To further confuse the issue, *method* is also a term used in some object-oriented programming languages to refer to the code controlling a certain behavior within an object. Whatever the strictly correct use of the word *method* is, many people will continue to use “method” and “methodology” interchangeably, just as they will continue to interchange *software* development and *system* development.

Still more potential for confusion exists when communicating with people in other disciplines who also share some of the same terms, but use the terms differently. For example, in some fields the term *design* incorporates the activities that software engineers would identify as “analysis.” You should prepare to be flexible in your understanding of such terms, taking into account the speaker’s frame of reference, the subject under discussion, and the context in which the term is used.

Key points

Systems

The concept of *systems engineering* is distinguished from the concept of *software engineering*. Software is generally a subsystem of a larger system incorporating hardware, support, and training. Most modern software itself is a system comprised of a number of distinct components.

Components

Modern software often consists of a collection of smaller programs that perform a specific set of functions. As with object-orientation in general (which forms the basis of many component technologies), component technologies assist engineering efficiency by facilitating reuse and ease-of-maintenance.

Products

Manufactured products, such as software, are created to serve some human need. Systems engineers should ensure that development efforts pay appropriate attention to the different product attributes such as usability and aesthetic appeal, and not focus on functionality alone.

Process

There are a number of approaches to the process of developing systems. Processes can be characterized as *linear*, *spiral*, or *hybrid*. Processes differ in their level of formality. It is now common to use prototyping as part of the software development process.

Terminology

In many subjects it is often difficult to develop an understanding of the various terms used. You should be aware that the same terms could be used to mean different things in different contexts.

Week 2 : Systems Engineering

Overview

Last week we introduced you in general terms to systems engineering and product development. This week we look at some of these concepts in more detail. In particular, we will examine the concept of a system, and begin to explore the concept of systems modeling. We will also investigate the various stages of the development process, and the management issues involved in software development. In doing so, you will investigate the different roles in the software development team, and some specific tools used in the development process.

In this second week you will read part of Chapter 12 on life-cycles processes. You will also begin to use a modeling tool to illustrate your understanding of a situation. This is a common activity for an analyst.

Objectives

Upon completion of Week 2 you should be able to:

- Categorize a process according to the SEI process maturity levels
- Create a model of a company's development process
- Describe the stages and activities involved in systems development
- Summarize the management activities that occur in software development
- Identify the main problems that can occur in systems engineering
- Identify the different skills involved in systems development
- Describe different ways that systems development projects are coordinated and controlled
- Provide detailed examples of the various kinds of tools and techniques used to support the software development process

Focus Questions

- What do you think are some of the biggest problems in developing a large system?
- If you were in charge of a large software development project working with a number of others, what policies or rules would you institute to ensure the success of the project?

Readings and Presentations

- Read Sections 12.1, 12.2, & 12.3 of Chapter 12 in *Object-Oriented Software Engineering*, by Bruegge and Dutoit.
- When you have completed reading the chapter in the textbook, read the Additional Notes for Week 2 (see the next section).

You are encouraged to take notes on your understanding of these readings, as well noting any clarifications you may require.

Activities

- Attempt the exercises in Chapter 1 of the textbook. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

The following assignments require consulting the software development company video interviews on the CD-ROM. The interviews also can be accessed in audio-only form at the course website.

Submit responses to the assignments below by the due date in the course calendar. The point value of the assignment is indicated in the parentheses.

Assignment 2.1 [Total of 20 points available]

Listen to the responses to the following interview question on the CD-ROM:

> "Could you give an overview of the software development process at your company?"

On the website under "Assessments" there is a diagram that illustrates the development process at one of the companies interviewed.

(A)

1) Identify the company whose process is represented in the diagram

(2 points)

2) Identify one flaw in the diagram.

(2 points)

Grading criteria:

- Is the company represented in the diagram correctly identified?
- Has a flaw in the diagram been identified?

(B) Based on information provided in the video interview, create a similar diagram to illustrate the development process for the company that will be listed in this week's **Course Announcements**. The diagram should be constructed in one of the tools specified on the course website.

(10 points for correctness, 6 points for presentation)

Grading criteria:

- Have all the activities and products of the company been represented?
- Are the activities and products in the correct order?
- Is the diagram neat and clear with appropriate labeling?

Assignment 2.2 [Total of 10 points available]

Do a Web search on the term *software engineering*. Visit at least three sites, list the URLs and titles of the sites, and then write a brief (60–120 words) description of the site you found to be the most interesting.

Grading criteria:

- Have three URLs been listed?
- Has a website been adequately described?
- Is the description in the student's own words?

Assignment 2.3 [Total of 6 points available]

Listen to the responses to the following interview questions on the CD-ROM:

- "What do you consider to be the most important part of the software development process?"
- "How is the work of the different people coordinated?"

(A) List the most important part in the development process as described by each company.

(4 points)

Company	Most important step

(B) Identify two methods of coordinating work that are mentioned by at least two companies.

(2 points)

Grading criteria:

- ▫ Are the steps in the development process identified for each company?
- ▫ Have two work coordination techniques common to two or more companies been identified?

Checklist

- ☐ Have you read Sections 12.1, 12.2, & 12.3 of Chapter 12 in the textbook?
- ☐ Have you completed the exercises at the end of the Chapter 1?
- ☐ Have you read the Week 2 Additional Notes?
- ☐ Have you checked the **Announcements** area of course website for additional instructions?
- ☐ Have you completed Assignments 2.1, 2.2, and 2.3?
- ☐ Have you completed the self-test?
- ☐ Have you met this week's learning objectives?

Week 2: Additional Notes

A Guide to System Stakeholders and Developers: Who are the people involved in the development process?

"…there's a big difference between cutting code and shipping products: deploying quality software is a team sport that requires a group of people with a variety of skills working together towards a common goal."

From Rational Software "white paper," by Grady Booch.

In this course we pay particular emphasis on communication within a development team. Good communication is not only dependent on the actual channels of communication, it is also dependent on psychological factors. One of the keys to good communication is effective leadership by the project/product manager. Different areas of expertise will result in different perspectives and priorities. Personality also plays a part in productive communication. A good project manager will be able to smooth communication through managing the interpersonal relationships of the project development team members.

The production of software-based products has undergone a rapid transition from a limited-market craft to an industry. In the past many software products were produced by individuals. The product designer was also the implementer, and implementation skills were highly regarded. The industrialization of production leads to a division of labor and people begin to specialize in different aspects of the production process. There is a separation between the designers and the constructors of a product; constructors often work on factory assembly lines and are no longer required to be knowledgeable of or skilled in the entire development process. Often, much of the implementation simply becomes automated.

An example of this industrial transition is the development of the automobile. Most of the very early automobiles were designed and handcrafted by small groups or individuals. Today, the design of automobiles is seen as a highly skilled activity, although designers are not expected to actually construct the automobiles. Actual construction (the implementation of the design) of automobiles is now an extremely automated process.

You should be aware that the success of a project depends not only on those immediately involved in the development. There are also other groups of people who have a stake in the outcome of the project, even though they are not directly involved in its development. This group includes the managers that commissioned the system, the senior management of the company doing the development, and the people who will eventually use the system.

It is very difficult to offer a clear and complete list of the job titles and responsibilities of those who will be involved in a software development project. Some companies will use a different job title for people performing essentially the same role. On smaller projects and in smaller companies, one person may perform a number of roles though maintaining only one job title. Companies which develop specialized types of software, such as multimedia, may require specialized job titles and responsibilities (e.g., audio production). Larger companies tend to have a relatively clear division of labor.

Customer
For off-the-shelf products, this is the person or company who is likely to purchase the product; for custom software, it is the person or company who commissions and pays for the product.

End-User
The customer and the end-user are not always the same. For example, if you are producing children's educational software, the end-user is a child but the customer, the one who makes the purchasing decision, is the parent or teacher. In custom products it is usually the management that commissions the system, but they are seldom the end-users. During product development it is important to consider the end-user and to evaluate the product using representative end-users. Many products have failed because although they initially pleased the customer, they created a high level of dissatisfaction in end-users who were not directly considered in the development process.

Marketing Specialist
A marketing specialist should have good knowledge about the geographic and demographic area in which the product is to be sold, the size of the market, competing products, product attributes that are important to customers, and the price customers are willing to pay for the product. Such information will have an important impact on the development process, helping the developers pay attention to the features of a product that will make it commercially viable.

Project/Product Manager
The project/product manager is usually the one responsible for ensuring that the system is completed on time and on budget. Prior to the commencement of a project, a project manager will plan the timeline for the project and arrange for the resources that will be required to complete this plan. Once the project has commenced, the project manager monitors progress against the plan, and identifies and solves any problems that could prevent completion of the project on time and on budget. *Product manager* is the term used in companies that develop off-the-shelf software.

Product Development Director/Operations Manager
Most companies will be working on several projects at the same time. The operations manager functions at a level above the project managers, and works with and assists the project/product managers to prioritize the allocation of resources to the different projects. The operations manager may also monitor the general operation of the support services that the projects make use of.

Technical Writer
Writing is a skill that is often underestimated. Most people can write, so it is difficult for many people to understand why a specialist is needed in this area. If you were to read the manuals for products developed 20 years ago compared with the ones produced by large software companies today, you would soon recognize that specialist writers are really required. Because the developers of a system know the system so intimately, it is hard for them to identify what may be difficult to understand for those who know nothing about the product (new users). A technical writer will work with the system developers and analyze what information a new user of the system will need to know. The writer will then compose, edit, and/or organize this information for manuals and online guides so that it will be clear and easy to understand. Technical writers try to keep the end-user in mind and will attempt to write in a jargon-free manner, or, if jargon cannot be avoided, ensure that the terminology is explained.

Programmer
This is one of the most general job titles. In the strictest sense, a programmer merely implements a design. However, in many situations a programmer will have a more creative role and is expected to perform some analysis and design (especially at the subsystem or component level). When

constructing novel systems, a programmer has to select or create efficient algorithms and often has to experiment with different ways of doing things. It is important to recognize that there are different skill levels within programming. General programmers can produce code in one or more languages. Specialist programmers have knowledge of the tools, techniques, algorithms, and data structures for constructing programs in a specialist areas such as databases, telecommunications, computer games, multimedia, and graphics.

Analyst/Designer
Some companies will differentiate the roles of the analyst and designer of software from that of the programmer. In these companies there usually are defined documentation standards for designs (e.g. the use of UML) whic h programmers then translate directly into program code.

Software Engineer
Software engineer is a relatively general term. In some companies, a software engineer's role will be akin to that of the analyst/designer. In others, the software engineer performs analysis, design, and coding. There are also cases where "software engineer" is used as an up-market job title to describe people who are primarily programmers.

Hardware Engineer
This person, who is an electronic, mechanical, or other type of engineer, will be involved in systems that require specialized hardware.

Human Factors Expert
This is someone who is specially trained in human psychology and can perform analysis and provide testing expertise relating to the usability attribute of a product. A human factors expert who focuses primarily on computers is sometimes referred to as an expert in *human-computer interaction*.

Graphics Designer
This is someone who has artistic skill and can create visual designs that will be visually appealing.

Multimedia Specialist (e.g., sound/video specialist)
In the past, information in computer systems was available only in the form of text. Many current software applications also make use of other media such as sound, graphics, and video to present information. A multimedia specialist will be expert in using one or more of these forms of media.

Instructional Designer
The instructional designer specializes in designing instructional material. Many complex software products require the creation of course and instructional materials to help people learn how to use the product. An instructional designer would assist in developing these instructional materials.

Research & Development (R&D) Specialist
Many companies will have research and development specialists who will evaluate and create new technologies to be integrated into the systems being developed.

Systems Support
Systems development requires a computer and communications support infrastructure. A systems support specialist will set up and support this infrastructure. The efficiency of systems support can seriously affect the efficiency of an entire project.

Financial Officer/Accountant
The overall goal of most businesses is to make a profit. Unless there is someone effectively estimating and monitoring the costs involved in developing a system, there is a danger that the development process may end up costing more than the customer has agreed to pay. The financial officer's responsibility is to be alert to and avoid cost overruns. The officer will also assist in budget planning and cost estimation.

Quality Control Manager
Companies often distinguish quality control and testing from the actual development process. It is sometimes difficult for developers to objectively evaluate their own work, as they will have biases against possible negative aspects of what they have done. A quality control manager will independently monitor the quality of a product (and sometimes the process) and provide constructive feedback on how things may be improved.

Communication

Given all the different job roles and types of expertise that could be involved in a systems development process, good communication is essential to ensure a project actually progresses toward completion. This requires a strong and open communications infrastructure. Some scientists and researchers believe that the use of Internet technology facilitates communications in development groups. In contrast, others believe that the best way to ensure good communication is to have the development team members located physically together. One reason for Microsoft's success may be that the company has until recently concentrated most of its developers in one geographical location. Because this is a technologically based learning course, you have the opportunity to experience Internet-facilitated group development.

Further Reading

"The Software Development Team," a "white paper," by Grady Booch. Available in the appendix of this study guide and at the course website.

Week 3: Communication

Overview

You should by now have developed a good overview of the systems development process. This week you will begin to examine the process in more detail, paying particular regard to people, communication, and tools. Last week you focused on the job roles of the development team members. You learned that maintaining open and clear lines of communication is crucial to the success of any project. In this third week you will continue to build upon these concepts by analyzing how project teams are organized and made efficient.

In this third week your instructor/mentor will assign you to a work group, with which you will collaborate throughout the semester. Your work group will later complete a project involving the analysis and design of a small system. As soon as you have been assigned your work group, send email to the other group members and introduce yourself.

Objectives

Upon completion of Week 3 you should be able to:

- Explain why documentation is necessary in the systems development process
- Distinguish *synchronous* and *asynchronous* communications mechanisms
- Identify the main mechanisms of communication used in a systems development process
- Identify advantages and disadvantages of each communication mechanism
- Describe the steps involved in planning project communications activities
- Distinguish *scheduled* and *event-driven* modes of communication
- Summarize the main types of communication that may occur in a systems development project

Focus Questions

- Email is increasingly being used as the main communication tool. What advantages does using this tool have? What disadvantages does its use have?
- What are the specific communication problems that occur within multi-disciplinary teams?

Readings and Presentations

- Read Chapter 3 of the textbook. In preparation for next week, also read Sections 2.1, 2.2, & 2.3 in Chapter 2. You are encouraged to take notes on your understanding of the material, as well as note any clarifications you may require.

- Read Grady Booch's white paper, "The Software Development Team," which is provided in the appendix to this study guide.

Activities

- Attempt the exercises at the end of Chapter 3. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

The following assignments require consulting the video interviews on the CD-ROM. The interviews also can be accessed in audio-only form at the course website.

Submit responses to the assignments below by the due date in the course calendar. The point value of portion of the assignment is indicated in the parentheses.

Assignment 3.1 [Total of 10 points available]

Listen to the responses to the following interview questions on the CD-ROM:

- "Can you describe what documentation occurs as part of the process?"
- "What is the purpose of the documentation?"

(A) List the types of documentation mentioned in the interview, marking with an asterisk (*) those common to at least three companies.

(5 points)

(B) What do you consider to be the most convincing reason given for extensive documentation? Why?

(3 points)

(C) Listen to the responses to the following interview question on the CD-ROM:

> "What kind of skills/qualities do you consider a good software developer should have?"

Identify two skills of a good software developer that were mentioned by at least two companies.

(2 points)

Grading criteria:

- Is the list of documentation types used by the companies complete?
- Have types of documentation common to three or more companies been identified?
- Has a reason for extensive documentation been identified and justified?
- Have two skill requirements of a software developer been identified?

Assignment 3.2 [Total of 14 points available]

On the course website there will be a reference to some site that provide information on object-oriented computer-aided software engineering (CASE) tools. These tools can facilitate one or more activities in the development process and often complement programming environments such as Microsoft's Visual Studio. You will be guided to select two tools, and write in your own words a summary of each tool's purpose and function. Each description should be 60–120 words.

1. Tool Name:
   Description:

2. Tool Name:
   Description:

(7 points per tool)

Grading criteria:

- Are the descriptions within the stated length?
- Are the descriptions in the student's own words?
- Do the descriptions give an adequate summary of the tools' functionality?
- Does the description identify what part of the process the tool provides assistance with?

Assignment 3.3 [Total of 15 points available]

Last week you received your work group assignments. This week interact with the group to complete this assessment. Distance learning students will do this online in their group discussion area, and on-campus students will use class time. You will conduct a discussion on what you have learned from watching the video interviews. Each group member should rate the companies' level according to Capability Maturity Model (CMM) and then state an opinion on which company appeared to have the most efficient development process, and why. Not everyone agrees with the CMM, so do not feel bound to say that a company on a high level of CMM is the most efficient. You may also want to state which response from an interviewee was the most surprising. Each group member should post at least one follow-up response to another group member's opinion.

This assignment has three phases:

Phase 1: Estimate what level on the CMM each company is on. Submit your estimates and your response to the question, "Which company appears to have the most efficient development process?" and "Which response from an interviewee was the most surprising?" Include justification for your opinion.

Phase 2: Post at least one message constructively responding to another student's post.

Phase 3: Cut and paste the responses you made in Phases 1 and 2 of the threaded discussion. Include the date and time each response was made to the discussion. If you made more than one contribution in Phase 2, include these additional contributions.

Grading criteria:

- Has an opinion been posted for one of the two questions?
- Is the opinion adequately supported?
- Has at least one follow-up message been posted in response to another group member's opinion?
- Is the response clear, polite, and constructive?

Checklist

- ☐ Have you read Chapter 3 of the textbook?
- ☐ Have you read Sections 2.1, 2.2, & 2.3 in Chapter 2 of the textbook?
- ☐ Have you completed the exercises at the end of the Chapter 3?
- ☐ Have you completed Assignments 3.1, 3.2, and 3.3?

- ☐ Have you completed the self-test?
- ☐ Have you met this week’s learning objectives?

Week 4: The Unified Modeling Language

Overview

In the first three weeks of this course you have examined some of the more general aspects of the systems development process, including the stages of production, job titles and roles, and the organization of project teams. You will now begin to focus on practical tools and techniques for analyzing large-scale problems that involve the construction of computer-based systems. You will become familiar with Unified Modeling Language (UML), which has emerged as a standard modeling language for the construction of object-oriented systems.

Analysis and design requires that developers understand the problem area and design a solution for that problem. To accomplish this goal, developers will attempt to model the problem area and the proposed solution. In the early days of computing, when a single person often designed a computer system, analysis and design models were conceived in the head of the developer, or were informally sketched in a non-standard way. Because teams of people build most modern systems, there needs to be a standard way of communicating both about the problem area and the proposed design solution. This is the role performed by a modeling language such as UML.

Using a natural language, such as English, to describe a model is difficult due to the ambiguities and inconsistencies inherent in any natural language. In order to avoid such ambiguities, analysts and designers have found it helpful to use graphical techniques to represent their ideas. This is common approach for many creative activities. For example, musicians will use musical notation, geographers will use mapping notations and architects architectural drawing notations.

UML was developed through the work of three of the most prominent thinkers in software engineering methodology: Grady Booch, Ivar Jacobson, and Jim Rumbaugh. Basically, UML is a notation that allows developers to graphically model a system under development from different perspectives. Different types of diagrams are used for different perspectives on a system.

In this fourth week you will be introduced to UML. You will use the language for the next several weeks, and will also use it in your class project (assigned in Week 6).

Objectives

Upon completion of Week 4 you should be able to:

- Explain the role of modeling in systems engineering
- Describe the role of notation in analysis and design
- Identify the different kinds of modeling that occur in analysis and design
- Distinguish between the functional, object, and dynamic models

- Demonstrate the ability to use UML notations on a simple problem
- Explain the role of the different UML diagrams
- Create a simple use case, class, sequence, and state chart diagram
- Describe the relation between scenarios and use cases
- Distinguish among phenomena, concepts, and abstraction, and be able to provide examples of each

Focus Questions

Some software developers go straight to coding projects without documenting any analysis or design thinking. What do you think are the main problems of this type of approach?

Readings and Presentations

- Last week you were asked to read Sections 2.1, 2.2, & 2.3 in Chapter 2 of the textbook. This week you should complete the rest of Chapter 2 and read the additional notes on UML. You are encouraged to take notes on your understanding of the material, as well as note any clarifications you may require.
- You should also try to read Sections 4.1, 4.2, & 4.3 of Chapter 4 in preparation for next week's topic.

Activities

- Attempt the exercises in the chapter. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

Submit responses to the assignments below by the due date in the course calendar. The point value of each portion of the assignment is indicated in parentheses.

You will be provided with a description of a new product on the course website. As a software engineer, you would be involved in the design and construction of the software program to make the product useful. As a systems engineer, you would model the complete product, including software and hardware. In the following assignments you will analyze the system and create a model of it using UML diagrams.

Assignment 4.1 [Total of 14 points available]

(A) Create a **use case diagram** for the given system similar to that on Figure 2-1 in the textbook.

(7 points)

(B) Create a **class diagram** for the given system similar to that on Figure 2-2 in the textbook.

(7 points)

Grading criteria:

- Does each diagram represent an accurate model of the system?
- Are diagrams constructed according to those illustrated in the textbook?

Assignment 4.2 [Total of 14 points available]

On the website you will be given an example of a use case for the given system.

(A) Create a **sequence diagram** for the given system similar to that on Figure 2-3 in the textbook.

(7 points)

(B) Create a **state chart diagram** for the given system similar to that on Figure 2-4 in the textbook.

(7 points)

Grading criteria:

- Does each diagram represent an accurate model of the system?
- Are diagrams constructed according to those illustrated in the textbook?

Assignment 4.3 [Total of 10 points available]

(A) List three phenomena and three concepts in this course.

(6 points)

(B) Create a **UML class diagram** for the given system similar to that on Figure 2-10 in the textbook that depicts two classes described on the course website.

(4 points)

Grading criteria:

- ▫ Have three phenomena and three concepts of this course been listed?
- ▫ Are diagrams constructed according to those illustrated in the textbook?

Checklist

- ☐ Have you completed your reading of Chapter 2 in the textbook?
- ☐ Have you completed the exercises at the end of the textbook?
- ☐ Have you begun reading Chapter 4 in the textbook?
- ☐ Have you completed Assignments 4.1, 4.2, and 4.3?
- ☐ Have you completed the self-test?
- ☐ Have you met this week’s learning objectives?

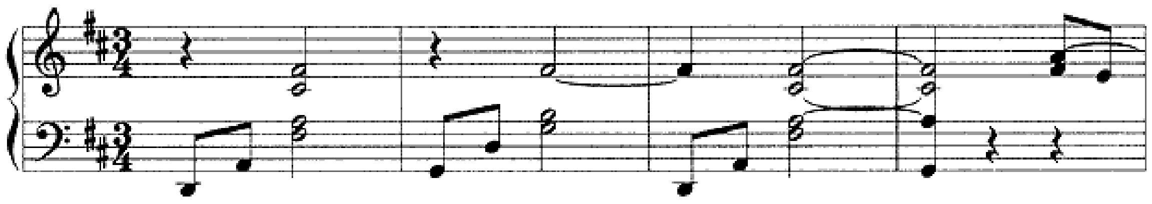
Week 4: Additional Notes

Notations, Modeling, and UML

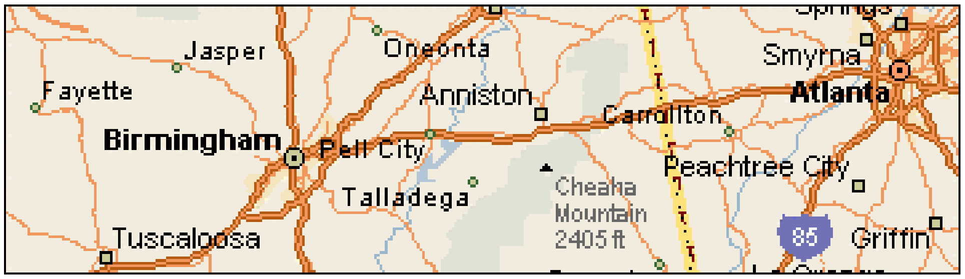
The success of humans as a species is largely built upon their ability to communicate. At first we could only communicate directly by word of mouth. The development of written language greatly spurred the development of civilization as it allowed us to transmit knowledge through space and time. Written language is a notation. It is a set of symbols that we learn to associate with the sounds of spoken language. In a sense we speak through the page to others. Different languages have developed different notations for written communication and some, e.g., Japanese, have more than one notation. Some use alphabets and some use pictograms (e.g., Chinese, Ancient Egyptian). Notations for human language have not solved all our communication needs: for example, if I wanted to communicate how to play a part of Bach's lute suite on the guitar, I could use human language as follows:

Simultaneously pluck the sixth, fourth, third and second strings while holding down the fourth string at the second fret. Sustain this for one second. Pluck the second string while holding down on the first fret, and then pluck it open. Pluck the third string on the second fret, then pluck it open and hammer on then pull off. Pluck the fourth string while holding on at the fourth fret and then at the second fret.

This only describes one small section of the suite and it would be difficult for any musician to read and play this at the same time. Musicians have therefore developed a special notation that has become standardized and communicates the information needed for an experienced musician to play as they read. It also allows composers to efficie ntly describe and edit new music.



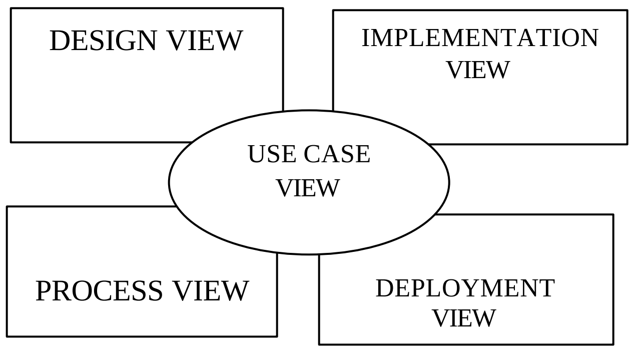
In many other areas of human activity we have developed special notations to communicate information efficiently. Other examples include road signs, icons on computer screens, and mapping notations. Looking at the notation below allows us to instantly answer a number of questions about geography, e.g., what is the nearest town from the highest point between Birmingham and Atlanta?



From its earliest days the world of computing has developed and used notations to represent design information, examples include flow-charting and data-flow diagrams. The importance of UML is that it is a well-defined language supporting all aspects of analysis and design process. The usefulness of any notation is related to how widely accepted it is and the level of standardization. Thus, for example, musical notation is standard and universally accepted. UML was adopted as a standard by the object management group* in 1997 and is well accepted within the software industry (at least in large companies working on large projects).

UML can be used for building models of both the problem domain and the solution domain. In modeling the problem domain we would be building an understanding of an existing system and where it needs to be improved. For example, if we were automating some business process it may help to understand the problem by modeling what currently occurs in the manual system. In modeling the solution domain we would be building a model of one or more proposed solutions to the problem domain that uses computer technology.

UML consists of a number of conventions for creating diagrams. The diagrams and accompanying documentation offer different views of a problem and the proposed solutions. The main views possible in a UML model are as follows:



The Use Case View
The use case view contains the diagrams used in analysis (use case and interaction diagrams), and all the elements that comprise these diagrams (e.g., actors). It is central to the other views as it describes the behavior of the system; if the system built does not behave as specified in the use case view, it will fail. In creating the use case view, we would begin by identifying the categories of users in a system; these would be represented on our diagrams as actors. An actor can be a human user or it can be another existing system. In use case analysis we would be concerned with looking at scenarios of how people work in an existing system (problem modeling) or will work in a proposed system (solution modeling). From these scenarios we can derive use cases that basically describe the functionality of a system. We can also illustrate the flow of control through a system using a sequence diagram.

Design View
The design view contains the diagrams used in object design (class diagrams, interaction and state transition diagrams). It offers a view of objects and their relationships either in the problem domain (e.g. paper forms, reports) or in a proposed solution (e.g., digital equivalent of forms and reports). In solution modeling the design view provides a bridge between the functionality envisaged in the use case view and the implementation. The basic element in this view is the *class*. A class diagram will

*An industry standards group supported by the main computer companies (see **www.omg.org**) often begin by showing the names of the classes of objects and the relationships between the classes. Later in design (during solution modeling) details of class attributes and operations will be added. A detailed class diagram should directly correspond with the class structure of object-oriented program code. The attributes in UML are implemented as data members in C++ or data fields in Java. The operations are implemented as member functions in C++ or methods in Java.* Many CASE tools allow the automatic generation of skeletal code for C++, Smalltalk, Java or Visual BASIC from a class diagram. In some tools you can also reverse engineer program code created in these languages into class diagrams. This allows existing components to be included in documented models, if there is access to the source code. In addition, changes that need to be made during implementation can be reflected in the documentation of the design model.

Process View
The process view models the threads and processes that will occur in the more complex systems. This is required for determining system throughput and scalability. We will not cover this aspect of UML in this course.

Implementation View
The implementation view looks at how the various new and existing components, systems (e.g., a database system) and files will be assembled into the physical system. This includes information about the code libraries, executable programs, runtime libraries, and other software components that comprise the completed systems. Components that do not exist and need to be created by you will have to be designed as part of the design view. The static aspects of this are captured in component diagrams and the dynamic aspects (communication among components) in interaction, state chart, and activity diagrams.

Deployment View
The deployment view illustrates how the completed system will be physically deployed. This view is necessary for complex applications in which a system will have different components located on different machines. For example, interface components may be located on a user machine whereas other components may be located on a network server.

Constructing UML Diagrams
There are now a variety of tools that support the creation of UML diagrams. There are computer-aided software engineering tools (CASE) such as Rational Software Corporation's Rational Rose, which allows the creation and documentation of UML views. In addition, many general diagramming tools now support UML, e.g., Microsoft Visio Professional. CASE tools differ from diagramming tools in that they allow the automatic generation of skeletal program code from UML class diagrams.

If you have access to a diagramming tool and are familiar with and are able to create UML diagrams, feel free to use it. If not, you will be directed to alternatives at the course website.

* Different object-oriented programming languages tend to use different names for the same concept. UML uses the terms *operations* and *attributes*, as it is not tied to any specific programming language.

Week 5: Requirements Elicitation

Overview

Most systems development processes begin by defining the requirements for the system that is to be constructed. To determine these requirements an analyst will use general information-gathering techniques, such as interviews, as well as specifically object-oriented methods, such as use case analysis.

A *requirement* is a statement of what the system must be able to do. If the requirement analysis is not done correctly, a great deal of time and effort may be spent designing a system, only to have it fail to meet the needs of its users. Although it may be impossible to account for all system requirements at the outset of the development process, the success of this stage will have the greatest effect on the overall efficiency of the process. A requirement that has to be amended or added at the design or implementation stage will have a greater cost implication than a requirement that is correctly accounted for at the start. The later it is in the development process, the more difficult it is to accommodate changes in design. Therefore, clear and open communication is crucial to requirements elicitation; without full understanding and cooperation between the software developers and users, this important stage in the development process will not be successful.

UML and in particular scenarios and use cases provide an important tool for formulating and communicating requirements. When requirements are stated in natural language they are often vague and open to different interpretations. This is particularly a problem when the developer and the client understand something different. Use case forces a more precise definition of functionality and provides a clear guide for design and implementation.

Objectives

Upon completion of Week 5 you should be able to:

- Describe the activities included in requirements elicitation
- Describe how an interview should be conducted
- Demonstrate the ability to ask appropriate questions in an analysis interview
- Identify *actors*, *scenarios,* and *use cases*
- Identify what constitutes a system requirement
- Create the documentation for a use case and scenario
- Distinguish the relationships that occur within use case diagrams
- Distinguish between *functional*, *nonfunctional*, and *pseudo* requirements
- Define the concepts of *correctness*, *completeness*, *consistency*, *clarity*, and *realism* as they relate to requirements

Focus Questions

In order to stay in business software development companies have to estimate the cost of development add a reasonable profit margin and then ensure they satisfy the customer without spending more than the cost of development. How does requirements analysis play a role in this?

Readings and Presentations

- Last week you were asked to read Sections 4.1, 4.2, & 4.3 in Chapter 4 of the textbook. This week you should complete the rest of Chapter 4, and the Additional Notes for Week 5 (see next section of this study guide). You are encouraged to take notes on your understanding of the material, as well as note any clarifications you may require.
- You should also try to read Sections 5.1, 5.2, & 5.3 of Chapter 5 in preparation for next week's topic.

Activities

- Attempt the exercises in the chapter. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

The following assignments are based on concepts presented in the textbook and on the software development company interviews on the CD-ROM. Submit responses to the assignments below by the due date in the course calendar. The point value of each portion of the assignment is indicated in parentheses.

Assignment 5.1 [Total of 6 points available]

Listen to the responses to the following interview question on the CD-ROM:

> "How do you develop an understanding of the problems you are developing software to solve?"

(A) Identify three techniques for understanding problems that could be applied to all problem areas.

(3 points)

(B) Briefly describe two difficulties that can arise in understanding the problem.

(3 points)

Grading criteria:

- Have three techniques for understanding problems been identified?
- Have two difficulties in understanding the problem been identified?
- Are the descriptions of the difficulties clear and concise?

Assignment 5.2 [Total of 8 points available]

On the website you will be given the description of a system for which you will write the requirements.

(A) In Section 4.3.1 of the textbook, the functional requirements of "Satwatch" are provided. Using this as an example, write in your own words functional requirements for the given system.

(6 points)

(B) Using the given system, identify an example of each of the following:

1. non-functional requirement
2 pseudorequirement

(2 points)

Grading criteria:

- Have four functional requirements for the system been identified?
- Have correct examples of a non-functional requirement and a pseudorequirement been identified?

Assignment 5.3 [Total of 12 points available]

Consider this course, *COP 3331: Object-Oriented Analysis and Design*, as a system.

(A) Represent the actors for this system in a manner similar to that shown on Figure 4-3 in the textbook.

(2 points)

(B) Describe a scenario from your own experience in this course in the same manner as that on Figure 4-5 in the textbook.

(5 points)

(C) Identify a use case for this system similar to the one depicted on Figure 4-7 in the textbook.

(5 points)

Grading criteria:

- Have the main actors been identified and correctly represented?
- Has a scenario from your own experience been correctly described?
- Has a use case been identified and correctly depicted?

Assignment 5.4 [Total of 15 points available]

(Distance learning students will do this online in their group discussion area, and on-campus students will use class time.) Consult the course website, where you will be given a scenario for a group exercise in requirements analysis. You will be graded on your participation.

Checklist

- ☐ Have you completed your reading of Chapter 4 in the textbook?
- ☐ Have you completed the exercises in Chapter 4?
- ☐ Have you begun reading Chapter 5 in the textbook?
- ☐ Have you read the Week 5 Additional Notes?
- ☐ Have you completed the self-test?
- ☐ Have you completed Assignments 5.1, 5.2, 5.3, and 5.4?
- ☐ Have you met this week’s learning objectives?

Week 5: Additional Notes

Requirements Elicitation: Interviewing Users and Domain Experts

One of the most important tools in the analysis phase of a systems development project is the interview. In order to get a better understanding of the problem, interviews can be conducted with experts involved in the problem area, and with the people who are likely to use the completed system (or in the case of an off-the-shelf product, potential customers).

The most important initial decision is who should be interviewed. It would be a mistake to interview only those who have commissioned the project, especially since they are not likely to use it. Those who commission new systems are usually in positions of management; while they may believe they know enough about the situation to act as the sole point of information, management is not always aware of crucial details and constraints in the working situation. To gain a more accurate perspective a variety of people must be interviewed, especially those who will actually use the completed system. There have been instances of large-scale software systems that had to be abandoned because of adverse user reaction. It is better to deal with any potential adverse user reaction in the analysis phase, rather than encountering it after completion of a system.

On the surface, interviewing may seem like a straightforward technique for gathering information, without need for any special training, and many people conduct interviews without having training. However, interviewing is both an art and a skill. Some people will prove naturally good interviewers, while others may need instruction or tips on how to conduct an interview. Barone and Switzer distinguish the *art* of interviewing from the *skill* of interviewing. Skilled interviewers may be able to elicit the desired information, but those who possess the art of interviewing will be able to accomplish this while also leaving the interviewee in a positive frame of mind. Barone and Switzer quote the example of a physician who, in addition to asking the right diagnostic questions, takes the time to reassure the patient.

People who will to be interviewed as part of a software development process may be wary of outside scrutiny, apprehensive about the effect of a computer system on their jobs, and unmotivated in offering assistance to the interviewer. The ability to both inform and reassure an interviewee during an interview is a highly desirable trait in a computer analyst.

Preparation

Reflecting on the skills and techniques of interviewing will certainly make the information gathering process more efficient. An interview has three main components: the opening, the body, and the conclusion. Reviewing the purpose of each of these three parts will help the interviewer organize the questions, thereby making the interviewing process easier on all concerned.

The Opening
The opening serves to introduce the interview participants to each other, establish a good working relationship, and motivate the interviewee to provide information to the interviewer. How the opening is conducted is very important. Research has shown that people generally form impressions early on in an interview, and once impressions have been formed it is difficult to change them. In order to facilitate an interviewee's full assistance, an interviewer must take great care to appear organized, friendly, and concerned. One mistake that interviewers make is spending too much time on their own

agenda and failing to show concern for the interviewee, who may desire understanding or reassurance. A motivated interviewee is the key requirement to getting good and useful information through an interview.

As part of the opening, the interviewer should inform the interviewee of the following:

1. What is expected from the interview
2. The probable length of time the interview will take
3. The purpose of the interview and how the information will be used
4. How that particular interview fits in with the general analysis effort
5. How the dialogue will be conducted (For example, can the interviewee ask questions any time she wishes? Can the interviewee contact the interviewer if she thinks of additional responses at a later time?)

The Body
The body of the interview is where the information-gathering work is done. In order to determine the requirements of the project, it is very important to have a clear understanding of what information is needed, and the questions that must be asked to elicit that information. Depending on how questions are phrased and the order in which they are asked, it is possible to get different responses and levels of information out of an interview.

There are many approaches to the interviewing process. One effective technique is the funnel sequence. This begins with a series of broad questions and moves to increasingly narrow or specific questions. This technique helps to relax the interviewee by first posing easier questions, in which the interviewee's own concerns can be included, prior to shifting the focus to the specific concerns of the interviewer. A series of funnel sequences can be used for each topic to be dealt with in an interview.

The Conclusion
In the conclusion the interviewer should summarize the interview and state his or her understanding of the information the interviewee has given. This will enable the interviewee to verify that the information is correct, as well as to clarify any additional points. The interview participants should then address and agree upon the issues that are still to be resolved, and set up a procedure for follow-up communication. Finally, there should be a closure, when the interviewer brings the interview to an end by standing, shaking hands with the interviewee, and making a closing statement.

Questioning
Questions can be asked in many different ways, and how they are asked can influence the success of the interview. For example, a question can be asked in a natural conversational manner, or it can be asked in an abrupt, formal tone. In order to build a rapport with the interviewee, a natural manner should be used in an analysis interview and questions should be clear, concise, and free from jargon. Double-barreled questions (two questions incorporated into one) should be avoided.

The wording of the question can greatly influence the answer. Care should be taken not to ask leading questions, for example, "Your supervisor told me that process *X* is the most important activity. What do you think?" Responses will also vary depending on how open or closed a question is. An open question can have a very broad range of response, for example, "What do you believe will be the main benefits of the new system?" A closed question will attempt to restrict the range of responses, for example, “Do you see the main benefit of the system as decreasing cost or increasing quality?” The funnel technique begins with more open questions and moves on to more closed questions. Inexperienced interviewers will sometimes combine open and closed questions, such as, “What do you believe will be the main benefits of the new system…um…like would you say it is decreasing the costs?” These types of questions may result in a simple “yes” or “no” answer that, had the inquiry been phrased as an open question, might have received an entirely different response.

Listening
In addition to being careful about the selection, phrasing, and order of the questions, a good interviewer should be alert and active in dealing with answers. There may be ambiguity in an answer, or the answer may indicate that the question was not understood. Often there will be a need for a follow-up question. A good interviewer will not remain restricted to a predetermined set of questions.

Observation

Direct observation of people working within their environment can also be useful in understanding a domain of work and determining the requirements for a computer system that will operate in that domain. Occasionally this may be done in a laboratory situation using video recording; more often it is done more informally in the workplace. Observation has some advantages over interviewing, as it can reveal the ways people actually work that would not normally be disclosed in an interview. Often people become so habituated to their everyday work tasks that they are unable to clearly describe what it is that they do. Some activities become so automatic that people are unaware that they do them, or are seen as so mundane as to be not worth mentioning.

In conducting workplace observation, there should be an awareness of what is known as the “Hawthorne effect.” This effect is named after a famous study dealing with the psychology of work. In the study, a group of workers at a factory were isolated and a number of their working conditions were altered, in order to discover the effect of these changes on the workers’ efficiency. It was found that no matter what change was made, efficiency increased. It was later determined that the increase in efficiency was not due to the changes in the working conditions. Rather, the motivating factor was that the workers felt special because the professors from the local university were taking an interest in them. The lesson learned was that the act of observation itself could alter the very behavior you hope to observe.

Further Reading

For a more comprehensive treatment on interviewing, consult:

Barone, J. T., & Switzer, J. Y. (1995). *Interviewing*: *Art and skill*. Needhan Heights, MA: Simon & Schuster.

Week 6: Analysis

Overview

Last week you examined requirements elicitation, which involves understanding the problem area and determining what the completed system is to achieve. The next stage in an analysis is to produce a model of the system that will structure and formalize the requirements. This model is used to verify the developer's understanding of the system and to identify any additional requirements or potential problems. Many failed computer systems are the result of incorrect or missing requirements. If an analysis is performed correctly, then the risk of such failures or omissions is greatly reduced.

Objectives

Upon completion of Week 6 you should be able to:

- Identify *entity*, *boundary*, and *control* objects
- Represent and qualify associations
- Use appropriate heuristics to identify model components from natural language
- Determine the association and multiplicity of an association among objects
- Model generalized relationships
- Conduct a review of an analysis model

Focus Questions

Developers sometimes refer to the danger of "analysis paralysis." What do you think this phrase describes?

Readings and Presentations

Finish reading Chapter 5 in the textbook. Although you will not be tested on the material in Section 5.5, you are encouraged to read it. You are encouraged to take notes on your understanding of the material, as well as note any clarifications you may require.

Activities

- Attempt the exercises in the chapter. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

The following assignments are based on concepts covered in the textbook. Submit responses to the assignments below by the due date in the course calendar. The point value of each portion of the assignment is indicated in parentheses.

Assignment 6.1 [Total of 11 points available]

(A) Refer to the system that will be described on the course website under Assessments. Identify two examples of a *boundary*, an *entity*, and a *control* object.

(6 points)

(B) Viewing this course as a system, identify at least one example of the following associations, and represent your examples as illustrated on Figure 5-6 in the textbook.

1. One to one
2. One to many (do not use the example illustrated below in Part C)
3. Many to many

(3 points)

(C) Refer to the example of a one-to-many association on the course website under Assessments. Qualify the given example.

(2 points)

Grading criteria:

- Have two correct examples been given for each type of analysis object?
- Has one example of each type of relationship been identified?
- Are the relationships represented using the correct notation?
- Has the given example been qualified?

Assignment 6.2 [Total of 11 points available]

Refer to the scenario that is contained in a document you will find at the course website.

(A) Referring to the description in the scenario, use Abbot's heuristics to identify at least one of each of the following:

Object, Class, Operation, Inheritance, Aggregation, Constraints, Attribute

(7 points)

(B) Submit an example of a question you might ask to clarify or elicit important missing information from the description given in the scenario.

(2 points)

(C) Using the heuristics for identifying boundary and control objects, give at least one example of each.

(2 points)

Grading criteria:

- Has one correct example been given for each type of model component?
- Have two correct follow-up questions been identified?
- Have examples of both a boundary and a control object been identified?

Checklist

- ☐ Have you completed your reading of Chapter 5 in the textbook?
- ☐ Have you completed the exercises in Chapter 5?
- ☐ Have you completed Assignments 6.1 and 6.2?
- ☐ Have you completed the self-test?
- ☐ Have you met this week's learning objectives?

Week 7: Project Preparation

Overview

This week you will begin to work with your team on the Group Project. The directions and specifications for the Group Project will be available on the course website under **Course Documents**. By this time you should have already made contact with your team members.

The Group Project will involve you working within a team to analyze a problem and produce a detailed requirements document for a system. The Group Project is due Week 12, and teams will be expected to have the project underway by the end of this week. Because communicative planning and collaborative skills are crucial to the success of a project, you should take the time now to discuss with your team members a schedule for project activities. You may wish to appoint a team leader, and assign other roles, e.g., proofreader, diagram creators.

Objectives

Upon completion of Week 7 you should be able to:

- ♦ Create a simple project plan
- ♦ Identify the main issues that occur in project management
- ♦ Describe how your group will communicate over the course of the project

Readings

Read Chapter 11 of the textbook. The content of this chapter will provide you with a guide to organizing and managing your group project.

Activities

- ▪ Review the specifications for the Group Project. Note that the project is to be completed by the end of Week 13.
- ▪ Meet with your project team and discuss the division of responsibilities and the project activity schedule.
- ▪ Listen to the responses to the following interview question on the CD-ROM:
  "How do you ensure you complete on time and on budget?"

Assessments

Assignment 7.1 [Total of 15 points available]

Projects in industry are usually planned using project management software. This software would allow you to produce Gantt charts like the one illustrated on Figure 11-17 of your textbook (p. 437).

(A) Use a diagramming tool to illustrate the plan for your team's project activities over the next few weeks. The plan should be fairly detailed, noting when meetings will take place and when all tasks (e.g., document section creation, document review) start and finish. You will also indicate which group members will be responsible for each task. You should expect to compare your actual progress on the group project with this plan.

(12 points)

(B) After reading the project directions, list the three most important questions you need to ask the customer. These should be phrased in the exact way you would ask them. For example:

"Could you please inform us if you currently have a web server and give us details on its current use and capacity?"

Not: *"I would ask them if they had a server"*

(3 points)

Grading criteria:

- Is the plan in agreement with the rest of the group members?
- Is there an appropriate amount of detail?
- Has it been indicated which group member takes part in each task?
- Have three appropriate questions been identified?

Checklist

- ☐ Have you read Chapter 11 in the textbook?
- ☐ Have you met with your group to discuss the project?
- ☐ Have you submitted a project plan?
- ☐ Have you listened to the responses on the CD-ROM?
- ☐ Have you completed Assignment 7.1?

Week 8: Systems Design

Overview

So far in this course, you have covered UML techniques to formulate requirements and construct an analysis model of a system. You will now examine the next stage of the development process: design. In design, a developer performs a number of activities to determine the details of how the system must be built to meet the requirements. In analysis we were concerned with understanding the problem domain and what is required to effect a better solution for dealing with the problem of the domain. In design we are concerned with the solution domain, and deciding which of the many potential technologies, architectures, algorithms, and pre-existing code will combine to effect an optimal solution.

In systems design we take a high-level view of the design of the whole system. We determine the design goals and the systems architecture, and decide how the system will be deployed into a working situation. We need to decide on such questions as what hardware will be required? What data needs to be stored (will a database be required)? Are there security concerns? How will the system be distributed between servers and client machines? What off-the-shelf solutions can be used?

The answers to some of these questions would have started to form during analysis. There is not always a clear division between different phases of the software development process. The important point about each phase is that there should be a clear and unambiguous documentation at the end of each stage, which addresses the main focus of that stage. Thus at the end of requirements elicitation there should be a requirements analysis document, clearly specifying the requirements of the system to be built. By the end of the systems design phase there should be a clear statement of the structure of the system to be built. There is an analogy to building architecture in that we draw a plan which shows the size of the house, where it will be located, and how it will be divided into rooms, but we do not specify the exact details of how each piece of wood will be cut and how each brick should be laid.

Objectives

Upon completion of Week 8 you should be able to:

- Identify a design goal
- Explain the heuristics that are used to identify subsystems
- Describe the products of a system design
- Define the concepts of a *subsystem*, *coupling*, and *coherence*
- Explain the techniques of *layering* and *partitioning*
- Define *persistent data stores*
- Define *access control*
- Identify boundary conditions
- Describe the factors that necessitate change in a system design.

- List the conditions that determine whether a system design is correct, complete, realistic, and readable
- Map subsystems to processors and components
- Create a UML deployment diagram

Focus Questions

Systems design is often said to be analogous to building architecture. Are there any differences between the two design areas that argue against the analogy?

Readings and Presentations

Finish reading Chapter 6 in the textbook. Although you will not be tested on Section 6.5, you are encouraged to read it. Take notes on your understanding of the material, as well as note any clarifications you may require. This is one of the most conceptually difficult chapters in the book you may wish to consult other reading materials to help in your understanding.

Activities

- Attempt the exercises in the chapter. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.
- You should also try to read Sections 7.1–7.3 in Chapter 7 in preparation for next week's topic.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

The following assignments are based on concepts covered in the textbook. Submit responses to the assignments below by the due date in the course calendar. The point value of each portion of the assignment is indicated in parentheses.

Assignment 8.1 [Total of 15 points available]

Consult the course website. You will be given a scenario that contains a systems design.

(A) Create a UML deployment diagram that illustrates the hardware and represents how you think the software would be deployed. Consider which software needs to be installed on which machine.

(12 points)

(B) For such a system give one example of each of the following:

1. Data that would need to be persistent
2. The access control for a specified actor
3. A boundary condition

(3 points)

Grading criteria:

- Has a deployment diagram been created?
- Are all the machines represented?
- Is all the software represented?
- Are correct connections specified?
- Has a correct example been given for data persistence, access control, and a boundary condition?

Checklist

- ☐ Have you read Chapter 6 of the textbook?
- ☐ Have you completed the exercises in Chapter 6?
- ☐ Have you begun reading Chapter 7?
- ☐ Have you completed Assignment 8.1?
- ☐ Have you completed the self-test?
- ☐ Have you met this week's learning objectives?

Week 9: Midterm Preparation

Overview

In this ninth week, you will review the material covered in the first eight weeks of the course and complete the midterm exam. Consult the course calender on the website for details, including dates, times, and location for this exam. **The midterm will count as 20% of your final grade.**

No assessments will be given this week. After you have reviewed the course material covered in the first seven weeks and completed the midterm, we recommend that that you read ahead in your textbook. Over the next several weeks you will be working on a Group Project in addition to the usual weekly course activities and assessments.

Readings

Prior to the midterm, you should have a good understanding of Chapters 1 through 5, and all the additional notes and readings.

Checklist

- ☐ Have you read everything required for the first eight weeks?
- ☐ Have you written summaries of the key points covered in the readings?
- ☐ Have you reviewed all the learning objectives for the first eight weeks?
- ☐ Have you done additional study for the objectives you have not met?
- ☐ Have you completed the self-test quizzes and reviewed the sample questions?

Week 10: Project Work/ Spring Break

Overview

In this week on-campus students will be on spring break. I would recommend all students take some well-earned rest and recuperation during this period. Do try and take at least some time to read ahead to the next part of the course and do not forget about your project. Distance learning students should use this week to progress in your projects.

Objectives

Upon completion of Week 10 you should:

- Have rested your weary brain and be ready for the second half of the course
- Be well on course for completing your group project

Readings

Read at least Sections 7.1, 7.2, & 7.3 in the textbook.

Checklist

- ☐ Did you take a break?
- ☐ Did you at least read something related to the course?

Week 11: Design Methods

Overview

While you will continue to explore how UML is used in analysis and design, you will also be introduced to other techniques that can be used in OOAD. *CRC cards* are a simple and effective technique for identifying classes in a domain and determining how those classes of objects will interact. Another popular trend in systems design is to build systems based upon pre-existing components built around component technologies such as Microsoft's COM objects (mentioned in the interview with ATG) and Sun's Java Beans. Finally, a popular concept to emerge in recent years, which borrows from building architecture design, is design patterns.

In addition to looking at different design approaches, in the readings we will begin to look at the area of object design. This follows on from the systems design where we determine the high-level structure of the system we will build to meet the requirements. In object design we focus more on the detailed structure of each sub-system and component.

Objectives

Upon completion of Week 11 you should be able to:

- Plan and conduct a CRC card session
- Distinguish between the *analysis* and *design* model of a system
- Describe the process of component-based design
- Define the term *component*
- Explain the concept of *design patterns*
- Distinguish application and solution objects
- Define *type*, *signature*, and *visibility*
- Provide examples of contracts for a class

Focus Questions

What do you think are the relative advantages and disadvantages of CRC cards and UML?

Readings and Presentations

- Read Sections 7.1, 7.2, & 7.3 of the textbook.
- Complete additional readings, which will be available at the course website.

Activities

Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

The following assignments are based on concepts covered in the textbook. Submit responses to the assignments below by the due date in the course calendar. The point value of each portion of the assignment is indicated in parentheses.

Assignment 11.1 [Total of 20 points available]

This will be a team assignment. You will work in the same group you have been assigned for your project. Confer with your group to arrange a mutually agreeable time to organize a CRC card session. You may try to complete the session in a single evening or over a number of evenings.

Consult the course website for the description of a problem area to address using CRC cards. At the end of the session, you will produce a report on the outcome of the CRC card session. The report should include the following:

1. A short introduction to the problem
2. A description of how the group planned the session and what the plan was
3. An overview of the session, including any difficulties encountered and the issues that were discussed
4. A copy of the cards at the end of the session

Your mentor/TA will monitor your involvement in the session.

Grading criteria:

- Did each student actively participate in the CRC card session?
- Was it evident that the group pre-planned for the session?
- Were the outcomes of the session adequately described?
- Was a complete list of classes identified for the problem area?
- Were correct responsibilities and collaborators identified for each class?

Checklist

- ☐ Have you started reading Chapter 7 of the textbook?
- ☐ Have you checked the course website for additional instructions?
- ☐ Have you completed Assignments 11.1??
- ☐ Have you completed the self-test?
- ☐ Have you met this week's learning objectives?
- ☐ Is your project on schedule to be completed on time?

Week 12: Object Design

Overview

We have already looked at systems design, in which the overall architecture for a system is designed and the components are identified. It is rare, however, that a project can be put together entirely from existing components. Usually, there is a need to design and program new components (some of which may be modifications of existing components). The design at the component level requires we determine the classes needed to make an individual component. This level of design requires a high level of understanding of object-oriented concepts and programming languages.

You have already been practicing object design in your programming classes; however, the difference here is that you are required to document your design thinking in UML diagrams. Documenting designs in UML allows developers to reflect upon the design before progressing to coding. For all but very small programs, changing a design diagram is more efficient than changing and recompiling code.

Objectives

Upon completion of Week 12 you should be able to:

- Refine a class diagram to its most detailed level
- Identify missing attributes and operations
- Convert a class diagram into C++ code
- Reverse engineer C++ code into a class diagram
- Specify type signatures and visibility
- Explain the need to specify constraints and exceptions
- Explain the role of class libraries and application frameworks
- Realize associations
- Remove implementation dependencies
- Turn objects into attributes

Focus Questions

What are the disadvantages of using third-party components rather than creating your own code?

Readings and Presentations

Complete the reading of Chapter 7 of the textbook. Although you will not be tested on Section 7.5, it is recommended that you read it. Take notes on your understanding of the material, as well as note any clarifications you may require.

Activities

- Attempt the exercises in the chapter. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.

- You should also try to read Sections 11.1, 11.2, & 11.3 in Chapter 11 in preparation for next week's topic.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

Submit responses to the assignment below by the due date in the course calendar. The point value of each portion of the assignment is indicated in parentheses.

Assignment 12.1 [Total of 10 points available]

On the course web site you will be given a small piece of C++ code. Reverse engineer this code into a refined class diagram.

Grading criteria:

- Is the diagram complete?
- Does the structure of the diagram completely match the structure of the code?

Assignment 12.2 [Total of 25 points available]

Consult the scenario presented at the course website under **Course Documents**.

(A) In the scenario you will be given the basic class structure for a component of a system. It will show the names of the classes and their relationships. You will refine this diagram into a detailed UML class diagram, which is ready to be implemented into code.

(15 points)

(B) Select two of the classes you have designed and implement them as C++ code. One of the classes must be a superclass.

(10 points)

Grading criteria:

- Is the diagram complete?
- Have the attributes and operations been specified for each class?

- Do the classes represented cover all the functionality required by the scenario?
- Is the C++ implementation of the two classes completely consistent with the design specification?

Checklist

- ☐ Have you completed your reading of Chapter 7 in the textbook?
- ☐ Have you completed the exercises in Chapter 7?
- ☐ Have you begun reading Chapter 11?
- ☐ Have you completed Assignments 12.1 and 12.2?
- ☐ Have you completed the self-test?
- ☐ Have you met this week's learning objectives?

Week 13: Quality Control and Testing

Overview

The exceptional importance of testing was emphasized in the video interviews with practicing software development companies. Regardless of the care a developer takes in each stage of software development, errors are still likely to arise. Testing is incorporated within quality control. Quality control must be integrated into the whole development process in order to be effective; it is not something that occurs at the end. Quality control has two functions: 1) to minimize the occurrence of error, and 2) to detect any error that has occurred through product testing (by product we mean all outputs of development including requirements and design models). These two functions must be carefully balanced. As in medicine, prevention is better than cure, so we must be careful not to overemphasize testing. A good system of quality control will look at errors discovered in testing and find ways of preventing the re-occurrence of such errors.

This week's Assessment is a substitution assessment. Your score on this assignment will be used to substitute for your lowest score on the previous week's assessments.

Objectives

Upon completion of Week 13 you should be able to:

- Identify quality control activities that occur during the systems development process
- Describe the activities that are part of testing
- Distinguish between *usability* testing and *functional* testing
- Describe the three main categories of quality control techniques
- Distinguish between *faults*, *errors*, and *failures*
- Provide an example of a test case
- Define a *test stub* and a *test driver*

Focus Questions

What kind of quality control procedures do you have in your project work? How could they be improved?

Readings and Presentations

Finish reading Chapter 9 in the textbook. Although you will not be tested on Section 9.5, you are encouraged to read it. Take notes on your understanding of the material, as well as note any clarifications you may require.

Activities

- Attempt the exercises in the chapter. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.

- You should also try to read Sections 12.1, 12.2, & 12.3 in Chapter 12 in preparation for next week's topic.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

The following assignments require that you consult the video interviews on the CD-ROM. The interviews also can be accessed in audio-only form at the course website.

Submit responses to the assignments below by the due date in the course calendar. The point value of each portion of the assignment is indicated in parentheses.

Assignment 13.1 [Total of 7 points available]

Listen to the following interview question on the CD-ROM:

> "What quality testing is done during and after software development?"

For one of the software companies, describe in 60-120 words or less how its working situation (company size, product type, etc.) affects the testing process.

Company name:
Description:

(7 points)

Grading criteria:

- Are the descriptions within the word limit?
- Does the description address the influence of the working situation on the testing process?

Assignment 13.2 [Total of 15 points available]

(A) Listen to the responses to the following interview question on the CD-ROM:

> "How do you ensure that your products will be usable?"

Based on these responses, indicate two reasons why usability testing is important.

(2 points)

(B) At the course website you will be informed of a site to visit.

1. In 60–120 of your own words, explain what is meant by *[term provided at course website]*?

   (7 points)

2. What are the four types of *[term provided at course website]*?

   (3 points)

3. Name four tools or techniques used *[term provided at course website]*?

   (3 points)

Grading criteria:

- Have two reasons been given for the importance of usability testing?
- Has the *term* been explained?
- Is the explanation in the student's own words?
- Is the explanation within the designated length?
- Are the four types of *term* identified?
- Have four tools or techniques used in the *term* been identified?

Assignment 13.3 [Total of 8 points available]

Listen to the responses to the following interview questions on the CD-ROM:

- "What kinds of problems arise in the development process?"
- "How do you resolve the problems?"

Complete the table provided by listing one problem experienced by each of the companies, and the solution used by the company.

(8 points)

Grading criteria:

- Has a problem in the development process and the solution utilized been identified for each company?
- Are the problems and solutions an accurate reflection of what is expressed in the interview?

Checklist

- ☐ Have you completed your reading of Chapter 9 in the textbook?
- ☐ Have you completed the exercises in Chapter 9?
- ☐ Have you begun reading Chapter 12 in the textbook?
- ☐ Have you completed Assignments 13.1, 13.2, and 13.3?
- ☐ Have you completed the self-test?
- ☐ Have you met this week’s learning objectives?

Week 14: Process Issues

Overview

In this course we have looked at how some real software companies develop software. By now, you should have come to the realize that there is no single right or wrong way to develop software, and arguments can be made for the relative efficiency of different methods of development. Many academics have put a great deal of thought and research into different methods of organizing the development process. We will conclude our study with a look at some different life-cycle models.

We will also take a brief look at some recently emerging contentious issues relating to the software development process. The first issue emerges form a backlash against the complexity and what some would see as the bureaucratic nature of many software development methodologies (such as those encouraged by the CMM). It has resulted in a movement towards “lighter,” more informal, methodolgies. The second issue relates to the predominance of closed software development where companies develop software with the internal workings hidden from customers. The open software movement promotes the development of a product where the source code is available for anyone to collaboratively alter and improve.

Please note that your Group Project is due this week.

Objectives

Upon completion of Week 14 you should be able to:

- Describe the IEEE 1074 standard for developing lifecycle process
- Contrast *traditional* and *light* methodologies
- Describe the main principles of extreme programming
- Explain the philosophy of *open source* development
- Compare and contrast the *Waterfall*, *V-model*, *Spiral*, *Sawtooth*, *Issue based*, and *Sharktooth* models
- Describe the benefits of rationale and configuration management
- Provide an overview of the unified development process

Focus Questions

What is your feeling on the issues of open versus closed development and light versus heavy methodologies? If you favor one perspective, what would you say is the strongest argument supporting the opposing perspective?

Readings and Presentations

- Finish reading Chapter 12 in the textbook. Although you will not be tested on Section 12.5, you are encouraged to read it. Take notes on your understanding of the material, as well as note any clarifications you may require.

- Read at least Sections 1 and 2 in Chapters 8 and 10.
- You will be provided with additional introductory readings and materials on light methodology and open source development.

Activities

- Attempt the exercises in the chapter. You will not be asked to submit your work, but try to answer the exercise questions. This will provide an excellent review of the key points of the chapter, and an opportunity to check your knowledge and understanding of the material.
- Check the **Announcements** area of the course website for additional instructions. Consult any learning materials listed, and then complete the weekly self-test quiz.

Assessments

There are no assessments for this week. You should take advantage of this extra time to begin preparing for the final exam.

Checklist

- ☐ Have you completed your reading of Chapter 12 in the textbook?
- ☐ Have you read the first parts of Chapters 8 and 10 in the textbook?
- ☐ Have you read the additional materials provided?
- ☐ Have you completed the exercises in Chapter 12?
- ☐ Have you completed the self-test?
- ☐ Have you met this week's learning objectives?

Week 15: Exam Preparation

Overview

You should use this week to continue your review of the material covered during the course (with an emphasis on the material covered since the midterm exam) and to complete the final exam. Consult the course calendar on the course website for details of this assessment. **The final exam will count as 30% of your final grade.**

We hope that in this course you have learned some essential concepts and ideas in software engineering, and that you have gained a good understanding of the process issues in software development. We also hope that you have been able to develop practical skills in analysis and design that you will practice and continue to improve on in the future.

We look forward to having you register for subsequent courses in Computer Science and Software Engineering.

Checklist

- ☐ Have you produced notes summarizing clearly and concisely the main point in each of the required readings?
- ☐ Have you reviewed all the learning objectives to ensure you have met them?

Week 16: Course Evaluation

Overview

Your mentor or instructor, and the course developers would like to get feedback from you on this course, *COP 3331: Object-Oriented Analysis and Design*. This information will be used to improve this course, as well as other courses in this program. You will be asked to evaluate your mentor or instructor, course content, assessments, etc. Your mentor or instructor will give you further instructions on how to complete a course evaluation.

Appendix

Required Reading

The Software Development Team

By Grady Booch

Rational Fellow

A Rational Software White paper

Developing quality software in a repeatable and predictable fashion is a job that requires the coordinated activity of a team of developers. Although individual productivity is still important, as systems get larger and more complex, the productivity of the team as a whole becomes a much more important factor in the success or failure of a project. This paper examines the importance of teams, their organization and tools, and pragmatic ways to grow a productive team so as to unify, optimize, and simplify their work.

TABLE OF CONTENTS

The Importance of Teams

As Gerald Weinberg pointed out over 25 years ago in his seminal work, *The Psychology of Computer Programming*, “computer programming is a human activity.”[1] When you are up at 2 AM trying to stamp out an elusive bug, programming does indeed look like an *isolated* human activity. However, there’s a big difference between cutting code and shipping products: deploying quality software is a team sport that requires a group of people with a variety of skills working together toward a common goal. Associated with his work on the Capability Maturity Model (CMM) for development organizations, Watts Humphrey observed that “the history of software development is one of increasing scale. Initially, a few individuals could hand craft small programs; the work soon grew beyond them. Teams of one or two dozen professionals were then used, but success was mixed. While many organizations have solved these small-system problems, the scale of our work continues to grow. Today, large projects require the coordinated work of many teams. Complexity is outpacing our ability to solve problems intuitively as they appear.”[2]

There are perhaps two major forces that drive this complexity. First, there is technology push, wherein the ever-declining cost of hardware and the growing availability of high-speed networks makes it possible to develop automated solutions that, even a year ago, would not have been economically feasible. Second, there is the societal pull, wherein individuals and organizations have come to rely upon such automation, and in so doing have developed an insatiable demand for systems that are better, faster, and cheaper. Combine that with a world-wide shortage of skilled software developers, the net result is that the typical development team is being asked to do more with less: more features and more quality with less resources and less time.

Weinberg goes on to note that “for the best programming at the least cost, give the best possible programmers you can find sufficient time so you need the smallest number of them.”[3] That is certainly sound advice: all things being equal, it’s better to have a small team than a large one, and to be relatively unconstrained by schedule. But, as Fred Brooks points out, “the dilemma is a cruel one. For efficiency and conceptual integrity, one prefers a few good minds in doing design and construction. Yet for large systems one wants to find a way to bring considerable manpower to bear, so that the product can make a timely appearance.”[4]

Most problems are sufficiently complex that they simply require a lot of hard work and sustained labor, more than can be carried out by a single developer working in isolation.[5] However, you cannot just expect a project to succeed by staffing it with superstars and arming them with powerful tools. Walker Royce points out that, although hiring good people is important, it’s far more important to build a good team.[6] He goes on to explain that balance and coverage are essential characteristics of such a team, and describes this by analogy. “A football team has a need for diverse skills, very much like a software development team. There has rarely been a great football team that didn’t have great coverage; offense, defense, and special teams, coaching and personnel, first stringers and reserve players, passing and running. Great teams need coverage across key positions with strong individual players. But a team loaded with superstars, all striving to set individual records and competing to be the team leader, can be embarrassed by a balanced team of solid players with a few leaders focused on the team result of winning the game.”[7]

In the context of software development, "winning the game" means developing and deploying quality software in a predictable and sustainable fashion. As Barry Boehm demonstrates in COCOMO (a model for software cost estimation), the capability of the team has the greatest impact upon productivity.[8] Tom DeMarco and Tim Lister go on to note that, within a team, an organizations most productive people will tend to outperform its least productive by a factor of 10:1.[9]

Historically, most advances in software development languages and tools have focused on improving the productivity of the individual developer. This is not to say that such advances are unimportant: I'd rather have a fast compiler than a slow one. However, given the importance of teams to modern software development, such advances in individual productivity have diminishing returns relative to winning the game. As such, it makes sense to turn our attention to the software development team, and ways to prove its productivity.

Organizing the Software Development Team

Drawing upon his experience inside Microsoft, Steve McConnell notes that "it takes more than just a group of people who happen to work together to constitute a team. In their book *The Wisdom of Teams*, Katzenbach and Smith define a team as 'a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable.'"[10] He goes on to enumerate the characteristics of a hyperproductive team:[11]

- a shared, elevating vision or goal
- a sense of team identity
- a results-driven structure
- competent team members
- a commitment to the team
- mutual trust
- interdependence among team members
- effective communication
- a sense of autonomy
- a sense of empowerment
- small team size
- a high level of enjoyment

There are many different ways to organize such a team: the business team, the chief-programmer team, the skunkworks team, the feature team, the search-and-rescue team are among those that McConnell identifies.[12] Although no one organization is optimal for every team or problem domain, the surgical team (as Brooks calls it[13]), also known as the chief programmer team, is perhaps the most common and most effective.

In this organization, the central player in the development organization is the architect, who is responsible for the "conceptual integrity of all aspects of the product perceivable by the user."[14] Don't confuse the role of the architect and the project manager, however, they encompass very different activities.[15] A team's architect is responsible for designing the system, where as the project manager is responsible for designing the team, and making it possible for them to do their job.

Within a team, you should choose an architect who possesses a vision large enough to inspire the project to great things, the wisdom born from experience that knows what to worry about and what to ignore, the pragmatism necessary to make hard engineering decisions, and the tenacity to see this vision through to closure.[16] On small projects, one architect is sufficient. On larger projects, architects are a recursive feature: different subsystems will require their own architect, with the overall conceptual integrity of the system being maintained by a single architect or, more commonly, a small team of architects.

Surrounding these architects at each level in the system are application developers, people who love to code and who are able to turn abstractions into reality.

Thus, the center of gravity of the development team should be formed by a tight grouping of the organization's project manager, its architect, and its application developers. Jim Coplien, in his study of hyperproductive organizations, has found that many such teams exhibit this common pattern or organization.[17]

These three skill groups are necessary but, as it turns out, insufficient. The successful deployment of any complex system requires a variety of other skills: toolsmith, quality assurance, system integration, build and release, and analyst are some of the identifiable roles you'll need to fill on most projects. Capers Jones notes that "the software industry reached the point of needing specialists as long ago as twenty years.… For some large enterprises, there may be more than one hundred different occupational groups within an overall software organization.[18] In large projects, you'll have specific individuals for each such role; in smaller teams, each individual will end up playing multiple, simultaneous roles.

Improving Team Productivity

Staffing a project with the right people who have the right skills is important, but that alone does not explain the differences in productivity one sees among such teams. In this context, DeMarco and Lister speak of a "jelled team" which they define as "a group of people so strongly knit that the whole is greater than the sum of the parts. The productivity of such a team is greater than that of the same people working in unjelled form. … Once a team begins to jell, the probability of success goes up dramatically."[19]

Essential to the formation of jelled teams is this precondition: a project must honor and respect the role of every one of its developers. This means that each project must recognize that its developers are not interchangeable parts, and that each brings to the table unique skills and idiosyncrasies that must be matched to the needs at hand and calibrated within the organization's development culture. This is one of the five basic principles of software staffing that Boehm describes: "fit the tasks to the skills and motivation of the people available."[20]

DeMarco and Lister suggest a simple formula for creating a jelled team:[21]

- get the right people
- make them happy so that they don't want to leave
- turn them loose

They go on to note ways to develop an organizational culture that encourages jelled teams to develop and flourish:[22]

- make a cult of quality
- provide lots of satisfying closure
- build a sense of eliteness
- allow and encourage heterogeneity
- preserve and project successful teams
- provide strategic but not tactical direction

Teams and Tools

When projects were simple and teams were small (often involving teams of one), an organization could get by with only the simplest of tools: an editor, a compiler, and a linker would be quite sufficient for many problems. Add a debugger, and you were really rocking.

Amazingly, lots of organizations still get by with only this minimalist set of tools. However, in the context of contemporary software development, that's akin to banging rocks together to light a fire: you can do it, but it's not particularly efficient. Rather, as the complexity of your project grows, you must consider other, more targeted tools, such as tools for requirements capture, visual modeling, configuration management and version control, performance analysis, and testing.

Personal integrated development environments are important in making the individual programmer productive. Indeed, there's been a long history of maturation of such tools, and their advances have been essential in enabling the creation of larger and more complex systems. However, there are limits to the cool features you can provide to the individual developer, and it would appear that we as an industry are getting close to those limits. Indeed, there is only such much you can do to help an individual developer bang out code faster.[23]

A recent trend in the industry has been the integration of such tools, tools that individually address point problems but that collectively cover the full spectrum of life cycle activities. That's certainly a predictable advance, but it's not necessarily the most important one. Rather, a greater improvement in the overall productivity of a project will only come from tools that empower the team as a whole. In other words, you must integrate your team, not just your tools.

Ed Yourdon points out that "the only way such a tool could be a silver bullet is if it allows or forces the developers to change their processes."[24] In other words, while it's important to supply your development team with good tools, you'll only see a state change in your team's productivity if you apply tools that encourage and enforce a sound development process.

The Next Generation of Team Tools

Elaborating upon Yourdon's comments, that suggests three trends that will likely drive future software development tools.

First, there is the need to unify a team's tools around a common process and a common set of artifacts. Development teams create and modify artifacts such as requirements, plans, models, code, components, tests, and so on. They employ tools to create and modify those artifacts. Insofar as those tools are clumsy to use or that get in the ways of manipulating those artifacts, they detract from the primary focus of the development team, namely, deploying quality systems in a predictable and sustainable fashion. That means that, across the set of tools used by the development team, they must share and preserve a uniform understanding of the semantics of these artifacts and the activities that manipulate them.

Second, there is the need to optimize a team's tools to the different skill sets that exist within the team, since many of a project's artifacts will be created and manipulated by different stakeholders. For example, a requirement may be created by an end user, elaborated upon by the project's architect, and referenced by the project's quality assurance personnel. Each one of these stakeholders has a different view into the artifacts of the project. As such, it would be suboptimal to provide a one size fits all development environment. Rather, its far better to provide a tool set that permits different, simultaneous views into a project's artifacts optimized to the needs of each individual stakeholder.

Third, there is the need to simplify the delivery of these tools, especially for development teams that are distributed in time or in space. Practically, this means we'll likely see such tool sets become Web-centric, since the Web is the quintessential common vehicle for providing access to and visualization of information. Already, we find projects that use the Web as a repository for all of their project's artifacts. Open communication is a key enabler in forming a jelled team, and making all such artifacts visible to the project facilitates that kind of communication.

Growing a Team

All that being said, how do you grow a team? Even if you've loaded up your team with all the latest and greatest tools and techniques, what must you do next to turn a disjoint set of individuals into a jelled team whose productivity is greater than the sum of its parts?

There are three pragmatic techniques that I've seen work.

First, identify clear roles and responsibilities suitable to your development culture and necessary for your particular domain, and then match individuals with the right skills to those roles.[25] For example, perhaps the most important role you need to identify is that of architect: an architect is the person or persons responsible for establishing the significant design decisions for the system and for creating and validating a suitable architectural style. An architect may not be your fastest or most clever programmer, but he or she must certainly be your most wise.

Second, consider the essential artifacts of your project, and organize your team around the set of activities that produce and manipulate those artifacts. Clearly, the most important artifact of the software project is the running system itself, but that alone is insufficient: the team must create a scaffold of other artifacts around that system in order to build it in an efficient and predictable fashion. This means selecting what other work products you need—such as software architecture documents, test plans, releases, and so on—and then establishing a plan for growing and evaluating those artifacts. For example, in an iterative style of development, it's important to drive each iteration according to essential use cases (which specify the desired behavior of the system and additionally serve as test cases) and according to project risk (which changes with each iteration, and may be manifest as technical, economic, or business risk). By focusing on artifacts such as these in a controlled and measured way, you create an environment that encourages your team to drive their work to closure and to focus on the most important things they can do at each moment to mitigate the risks of the project.

Third, leverage tools that let each individual manipulate these artifacts in a manner appropriate to their specific role and consistent with other roles, and in a manner that reduces the interference between individuals. This is what the emerging generation of team tools is all about: providing an environment that encourages individual skills and enables those individuals to work productively and cooperatively.

Wrapping Up

These are indeed interesting times. The challenges of software development are certainly not going to go away, for we as an industry are continually being driven to do more with less. Methods and processes help; so do languages, frameworks, and point tools. However, software development is ultimately a human endeavor, and as such its ultimately the efforts of the software development team that enable us to deliver quality systems in a predictable and sustainable fashion. Tools that unify, optimize, and simplify the work of that team represent the next state change in helping create jelled teams.

Grady Booch, Rational Fellow

Grady Booch is one of the leading software development methodologists in the world. He is recognized internationally for his work on software architecture, modeling, and software engineering processes, all of which have contributed to improving the effectiveness of developers worldwide. Now serving as a Rational Fellow, Grady was a Chief Scientist for Rational Software Corporation from 1981-1999, where he was responsible for the development of many Rational tools, including Rational Rose, the world's leading visual modeling tool. Grady was one of the original developers of the Unified Modeling Language (UML), the industry standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. He is the author of six best selling books, has published several hundred technical articles on software engineering, and has lectured and consulted worldwide. In addition to Grady's continued affiliation with Rational, Grady is Chief Technical Officer and Vice President of Catapulse:
www.catapulse.com

About Rational Software Corporation

Rational Software Corporation (Nasdaq: RATL), the e-development company, helps organizations develop and deploy software for e-business, e-infrastructure, and e-devices through a combination of tools, services and software engineering best practices. Rational's e-development solution helps organizations overcome the e-software paradox by accelerating time to market while improving quality. Rational's integrated solution simplifies the process of acquiring, deploying and supporting a comprehensive software development platform, reducing total cost of ownership. International Data Corporation (IDC) has recognized Rational as the leader in multiple segments of the software development life-cycle management market for three years in a row. Founded in 1981, Rational, one of the world's largest Internet software companies, had revenues of $572 million in its fiscal year that ended in March, 2000 and employs more than 3,000 people around the world.

Notes

1. Weinberg, p. 3.
2. Humphrey, p. vii.
3. Weinberg, p. 69.
4. Brooks, p. 31.
5. Booch, p. 190.
6. Royce, p. 43.
7. Royce, p. 43.
8. Boehm, p. 642.
9. DeMarco, p. 45.
10. McConnell, p. 275.
11. McConnell, pp. 278 279.
12. McConnell, pp. 304 313.
13. Brooks, p. 29.
14. Brooks, p. 256.
15. Booch, p. 200.
16. Booch, p. 197.
17. Booch, p. 207.
18. Jones, p. 215.
19. DeMarco, p. 123.
20. Boehm, p. 669.
21. DeMarco, p. 93.
22. DeMarco, p. 151.
23. As an aside, I've often said that the best way to accelerate software development is simply by writing less software. That's why object-oriented techniques and component-based development are useful: the former encourages you to write less code via inheritance and abstractions that form a balanced set of responsibilities, and the latter encourages you to reuse and adapt existing components and frameworks rather than writing your own from scratch.
24. Yourdon, p. 183.
25. For example, the Rational Unified Process specifies a common set of roles, which it calls workers.

Bibliography

Boehm, B. 1981. Software Engineering Economics. Englewood Cliffs, New Jersey: Prentice-Hall.

Booch, G. 1996. Object Solutions: Managing the Object-Oriented Project. Menlo Park, California: Addison-Wesley.

Brooks, F. 1995. The Mythical Man-Month: Essays on Software Engineering, anniversary edition. Reading, Massachusetts: Addison-Wesley.

DeMarco, T. and Lister, T. 1987. Peopleware: Productive Projects and Teams. New York, New York: Dorset House.

Gilb, T. 1988. Principles of Software Engineering Management. Wokingham, England: Addison-Wesley.

Glass, R. 1998. Software Runaways: Lessons Learned from Massive Software Project Failures. Upper Saddle River, New Jersey: Prentice Hall.

Humphrey, W. 1989. Managing the Software Process. Reading, Massachusetts: Addison-Wesley.

Jones, C. 1996. Patterns of Software Systems Failure and Success. London, England: Thomson Computer Press.

McCarthy, J. 1995. Dynamics of Software Development. Redmond, Washington: Microsoft Press.

McConnell, S. 1996. Rapid Development: Taming Wild Software Schedules. Redmond, Washington: Microsoft Press.

Royce, W. 1998. Software Project Management. Reading, Massachusetts: Addison-Wesley.

Shneiderman, B. 1980. Software Psychology: Human Factors in Computer and Information Systems. Cambridge, Massachusetts: Winthrop Publishers.

Weinberg, G. 1971. The Psychology of Computer Programming. New York, New York: Van Nostrand Reinhold.

Yourdon, E. 1997. Death March. Upper Saddle River, New Jersey: Prentice Hall.